ip infusion™

ZebOS®
Network Platform
Version 7.8.4

Open Shortest Path First Architecture Developer Guide

August 2012

IP Infusion Proprietary

© 2012 IP Infusion Inc. All Rights Reserved.

This documentation is subject to change without notice. The software described in this document and this documentation are furnished under a license agreement or nondisclosure agreement. The software and documentation may be used or copied only in accordance with the terms of the applicable agreement. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or any means electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's internal use without the written permission of IP Infusion Inc.

IP Infusion Inc.
1188 East Arques Avenue
Sunnyvale CA 94085
(408) 400-1900- main
(408) 400-1863 - fax

For support, questions, or comments via E-mail, contact:
support@ipinfusion.com

Trademarks:
ZebOS is a registered trademark, and IP Infusion and the ipinfusion logo are trademarks of IP Infusion Inc.
All other trademarks are trademarks of their respective companies.

Table of Contents

©2012 IP Infusion Inc. Proprietary

©2012 IP Infusion Inc. Proprietary

©2012 IP Infusion Inc. Proprietary

CHAPTER 1 OSPF Fundamentals

Network administrators and developers who install and configure ZebOS® Network Platform IP routing software should use this developer guide. This guide contains the following information for ZebOS OSPFv2 (IPv4) and OSPFv3 (IPv6):

- An overview of the OSPF architecture.
- Detailed information about the ZebOS OSPF module.
- Details about these features:
  - OSPF optimization
  - OSPF multiple instance
  - Graceful restart
  - CSPF

Overview

The OSPF (Open Shortest Path First) routing protocol uses the Djikstra algorithm to calculate the shortest path between a source and destination. A typical OSPF network is an Autonomous System (AS) comprised of areas. An area contains routers and host devices. Routers that interconnect different areas are Area Border Routers (ABRs), and these are part of the backbone of the AS. Each link in an area is assigned a value that represents its cost (which could be a measure of bandwidth, distance, QoS, or a combination of these and other factors).

For communication to occur between areas, each ABR announces reachability data from one area to other areas. Each ABR does an SPF calculation and connects to the backbone. Routers within an area form a Link State database. OSPF routers use Hello packets (Keep-alive messages) to maintain adjacency. OSPF routers do not lose adjacency while doing an SPF calculation, because the calculation thread can be suspended to allow hello packets to be sent and received.

Basic Functionality

In the figure below, for Host6 to reach Host4, there are two possible paths: one hop through R6 or two hops through R4 and R5. Depending on the sum of the link costs, the one-hop path may be less preferable.

For Host1 to reach Host5, an inter-area transfer, the packets must travel across the backbone, beginning at R3 through ABR2, R5, ABR3; then into AREA C through R7 to R8 to the access point and wirelessly to Host5.

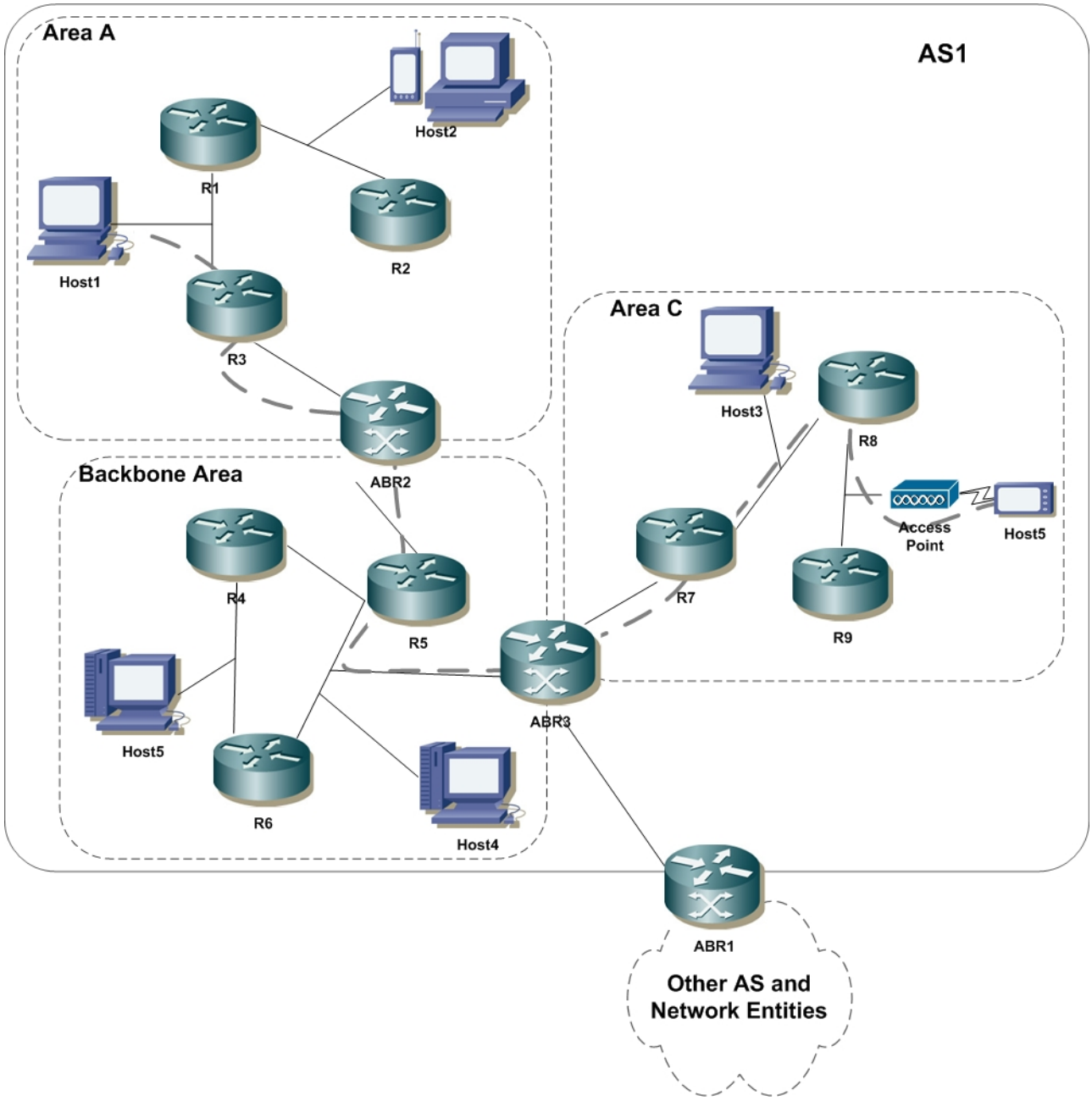


Figure 1: Basic OSPF Functionality

When the network changes, due to either a dropped or added path, or a router goes down or one comes online, neighboring routers adjust their routing tables. Other routers collect this data and change their tables, in a process called convergence. Because only the affected parts of the tables are shared, OSPF areas converge very quickly, compared to a RIP network that sends out entire routing tables. OSPF networks utilize less bandwidth for network maintenance, leaving more bandwidth available for payload.

©2012 IP Infusion Inc. Proprietary

OSPF does not limit the number of hops per path. Path cost is not determined by the number of hops, though this is a valid basic measurement. Path cost is determined by cost attributes assigned to each path by network personnel.

OSPF routers periodically send Hello and Refresh packets, as a way of keeping the routing tables accurate and up to date. If a Hello packet is not received from a router within the defined time, the router is assumed to be offline, and the tables are recalculated.

OSPF and PPVPN

ZebOS uses a simpler method of Provider Edge (PE) to Customer Edge (CE) router communication in BGP MPLS VPNs than that proposed by VPN-BGP-OSPF. Instead of using Type-3 LSAs, ZebOS propagates VPN routes as Type-5 LSAs, while using multiple instances of OSPF in conjunction with standard BGP/OSPF route redistribution mechanisms. This is compliant with `draft-ishiguro-ppvpn-pe-ce-ospf-01.txt`.

Because the PE-CE feature requires VRF support, only environments with stacks that support VRF can take advantage of the feature. The environments in which this is acceptable are:

Environment/OS	overlap-vrf	ospf-pece
linux	no	yes
netbsd	no	no
vxworks	no	no
vxworks_ipnet	yes	yes

The ZebOS implementation of the draft is platform independent, requiring VRF support in the stack.

ZebOS BGP VPN is a fully-scalable implementation of IETF 2547-bis. ZebOS BGP can be used either as a PE/P node in the Service provider network, or as CPE to connect to Service provider edge using EBGP. Other ZebOS routing protocols, such as RIP, are also extended to work as CE-PE routing protocols.

ZebOS BGP uses multiple forwarding tables to provide network Isolation of VPN traffic inside the service provider network. A separate default forwarding table can be used to contain public Internet routes and site-based VRF tables for VPN routes. The PE router maintains a VRF table, per site, that shares the same routes. When an IP packet is received on a PE-CE interface, the corresponding VRF is used for the destination IP address lookup.

CE-PE Interaction

The CE devices learn and advertise routes to other sites in its VPNs. ZebOS BGP, OSPF, and RIP can be used for CE-PE route exchange. BGP in PE installs the routes learned in the corresponding VRF table of the site, based the incoming interface.

PE-PE Interaction

ZebOS BGP installs the VPN routes, at the PE device, that were learned from the attached CE, in VRF tables. ZebOS BGP uses Multiprotocol BGP (MP-BGP) to distribute VPN routes to another PE BGP, across the SP network. Routes that are exported to BGP from VRF tables are converted into unique addresses in the VPN-IP address family using Route Distingushers (RDs). The VPN-IP route is a 12-byte quantity that consists of an RD and the IPv4 address. ZebOS BGP uses route targets to advertise by PE devices with the Router Target attribute. Every VRF is associated with a set of route targets. The import route target list specifies the routes that can be installed in a VRF. Routes that are advertised to other PE devices are tagged with the export route targets.

©2012 IP Infusion Inc. Proprietary

Process Flow

The figure below illustrates the OSPF process flow.

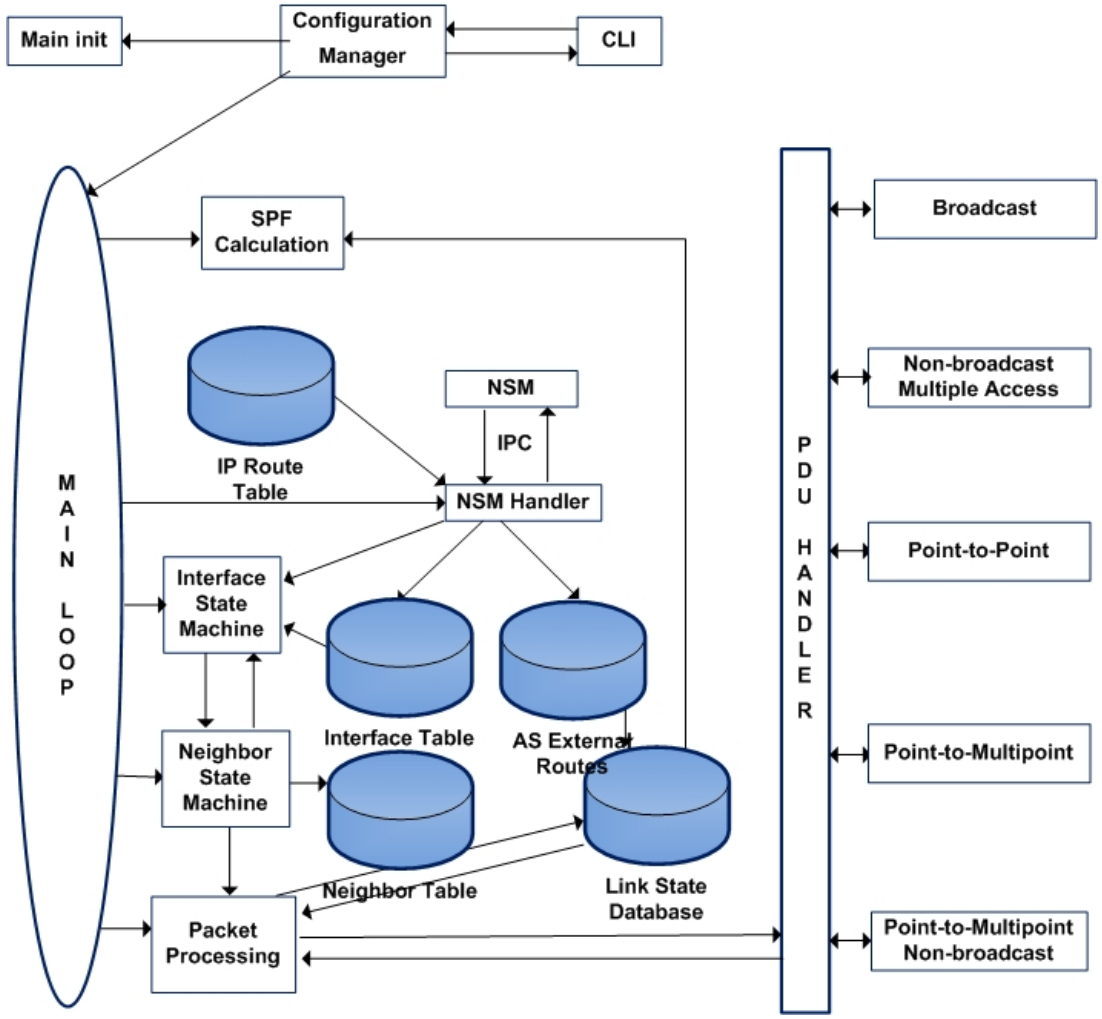


Figure 2: OSPF Processing Flow

Source Code

The following illustrates an overview of OSPF source code.

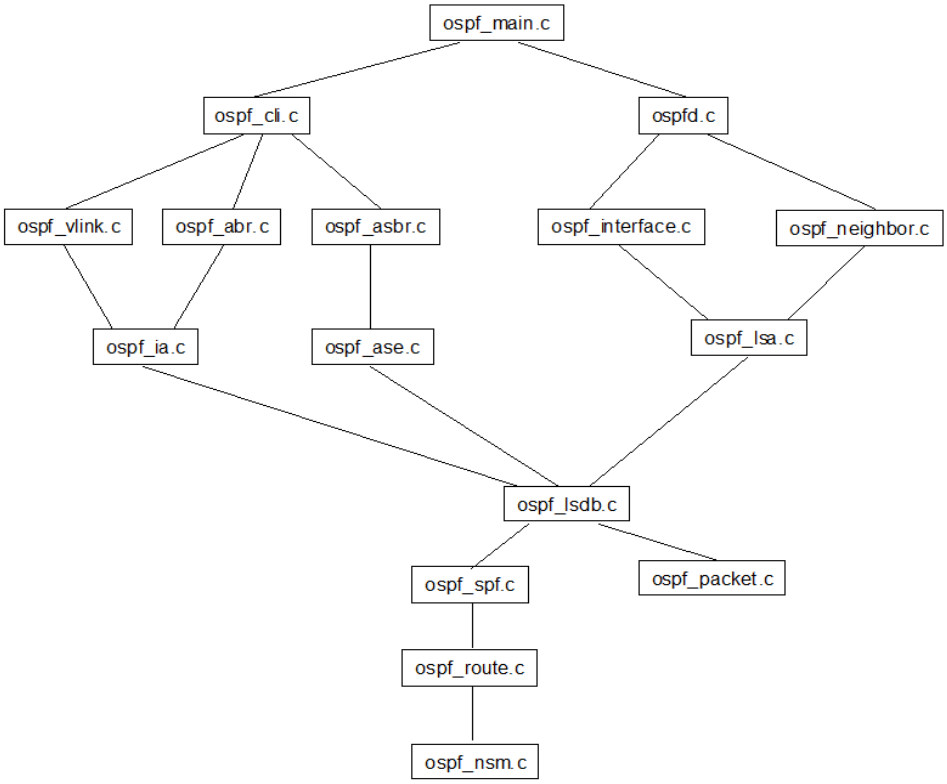


Figure 3: OSPF Source Overview

©2012 IP Infusion Inc. Proprietary

Core Modules

The following lists and describes OSPF core modules.

Module	Description
ospf_main.c	Contains main routine to start OSPF
ospfd.c	OSPF instance handling routines
ospf_vlink.c	OSPF virtual link handling routines
ospf_abr.c	OSPF router's behavior as area border router
ospf_asbr.c	OSPF router's behavior as autonomous system boundary router
ospf_interface.c	Contains OSPF's logical interface handling functions
ospf_ifsm.c	Contains OSPF's interface state machine
ospf_neighbor.c	Contains OSPF's neighbor handling functions
ospf_nfsm.c	Contains OSPF's neighbor state machine
ospf_ia.c	Contains routines for handling OSPF's inter-area routes
ospf_ase.c	Contains routines for handling OSPF's AS-External routes
ospf_lsa.c	Contains OSPF's Link State Advertisement (LSA) handling functions
ospf_lsdb.c	OSPF's link state database handling functions
ospf_spf.c	Contains routines that carry out SPF calculation
ospf_route.c	Handles manipulation and updating of routes; notifies NSM
ospf_nsm.c	Message processing for ZebOS NSM and external routing information handling

©2012 IP Infusion Inc. Proprietary

©2012 IP Infusion Inc. Proprietary

CHAPTER 3 OSPF Optimization

ZebOS OSPF optimization includes support for the following features:

- SPF calculation optimization
- Congestion detection and avoidance
- MaxAge Walker optimization
- Controlled incremental SPF calculation
- SPF exponential hold-time backoff

These features are described in the following sections.

SPF Calculation Optimization

SPF calculation optimization reduces total SPF calculation time to enhance scalability and provide faster convergence.

SPF calculation time in OSPF is optimized by implementing the candidate list using a binary-heap data structure, instead of a linked-list data structure. The binary-heap data structure is located in the ZebOS common library module. This data structure is used compactly for implementing the candidate list in the SPF calculation to significantly improve the SPF computation in OSPF, particularly for large networks.

SPF calculation in ZebOS is based on the Dijkstra algorithm. Previously, the linked-list data structure was used to store the candidate list of vertices. Operations, such as, insert and delete, performed with this linked list in the SPF calculation, required more amortized cost. By using the binary-heap structure, instead of the linked-list data structure, to store the candidate list, the amortized cost for all SPF calculation operations are reduced, resulting in the optimal implementation of the Dijkstra algorithm.

Another advantage of the binary-heap data structure is that it can be stored compactly: No space is required for pointers; instead, the parent and children of each node can be found by simple arithmetic on array indices.

Congestion Detection and Avoidance

When network congestion occurs, the major source for the continuity of the congestion is the repeated Link State Advertisement (LSA) retransmissions. ZebOS solves this problem by controlling LSA generation and retransmissions, thus, controlling LSA flooding when an OSPF network becomes congested. This solution improves the scalability and stability of large OSPF networks.

LSA Retransmission Interval

Congestion occurs when the percentage of unacknowledged LSAs to a neighbor is higher than 75 percent. If congestion is detected for a neighbor, the retransmission interval for all LSAs to the neighbor is exponentially increased using the following exponential back-off algorithm:

R(1) = Rmin

R(i+1) = Min(KR(i), Rmax) for i >=1

where:

R(i) represents the LSA retransmission (Rxmt) interval value used during the i-th retransmission of an LSA.

K, Rmin, and Rmax are constants

Min(.,.) represents the minimum value of its two arguments

The default Rxmt interval is always taken as Rmin. The default Rxmt interval is 5 seconds, if not configured using the `ip ospf retransmit interval` command.

Rmax is 8 times the value of Rmin. If the Rmin value is changed using the `ip ospf retransmit interval` command, the Rmax value will also be changed, however, the Rmax value cannot exceed 65535.

If the percentage of unacknowledged LSAs drops below 75 percent, but is more than 25 percent, the same retransmission interval is used as the interval used during congestion. The retransmission interval is reduced to the default value (Rmin) only when the percentage of unacknowledged LSAs drop below 25 percent.

Congestion Detection

Congestion is detected at the neighbor, based on the number of unacknowledged LSAs using the following algorithm:

nbr->rxmt_count = x;

nbr->ack_count = y;

then unack_count = x – y;

if (unack_count == 75% of x)

then nbr has congestion;

where:

nbr = neighbor

rxmt = LSA retransmission

Retransmission Lists

ZebOS maintains multiple retransmission lists for each neighbor. LSAs are added to the retransmission list, depending on the value: (current time)%(number of retransmission list). This ensures fair distribution of LSAs to multiple retransmission lists.

The retransmission interval is equally divided among the retransmission lists. For example, if the retransmission interval is 5 seconds, and 5 retransmission lists are used, LSAs from one retransmission list are flooded every second.

If congestion occurs, the LSA retransmission interval is increased, and the retransmission interval is correspondingly increased. For example, if the retransmission interval is increased to 20 seconds, LSAs from every retransmission list are flooded every 4 seconds.

 ©2012 IP Infusion Inc. Proprietary

MaxAge Walker Optimization

The OPSF MaxAge Walker Optimization process increases processing efficiency by distributing the CPU load and allowing CPU time for other processes. This optimization avoids unnecessary scans of the entire Link State Database (LSDB), and provides enhanced processing efficiency and improved scalability.

The OSPF_MaxAge_Walker routine was originally called every 10 seconds, scanned through the entire LSDB, and determined which LSAs Max-Aged: This was a very CPU intensive process, especially with LSAs in the order of several hundreds of thousands. Also, frequent scans of the LSDB affected other CPU processes. To optimize the processing efficiency, two solutions are provided:

1. Reduce the number of LSDB scans: Instead of scanning the LSDB every 10 seconds, ZebOS tracks the maximum age of all LSAs in the LSDB. For example, if the maximum age is 30 minutes, the LSDB is not scanned for another 30 minutes. Also, during each LSA addition, the maximum age value is checked and updated.
2. Scan a limited number of LSAs in a single scan: Instead of scanning the entire LSDB, scan in multiple steps. After scanning a certain number of LSAs, the event is suspended, and the next scan event is scheduled. This allows CPU time for other processes.

Controlled Incremental SPF Calculation

ZebOS intelligently determines when not to do the incremental SPF calculation to improve CPU utilization of SPF, thus providing enhanced scalability when a router is heavily loaded under transient conditions.

This is accomplished by controlling SPF calculation for Summary, AS-External, or NSSA LSA types, if there is a high rate of LSAs when a topology change occurs.

SPF Calculation Functionality

OSPF uses a link state in each individual area that makes up the hierarchy and calculates the shortest path tree inside each area using the LSDB.

The LSDB is constructed from information from other routers gathered using LSAs. The SPF algorithm gleans information from these LSAs to prepare the SPF tree. The SPF algorithm is calculated every time there is a change in the topology within an area.

Installing a new LSA in the database, either as the result of flooding or a newly self-originated LSA, may cause the OSPF routing table to be recalculated.

The SPF algorithm is a very CPU intensive process: frequent SPF calculation for each change can affect the other processes running on the same CPU. To prevent frequent SPF calculation after a change in topology is received via the Router or Network LSA, the system waits for a default period of time, called the hold time, before calculating the SPF.

After the SPF tree is calculated for the area, routes to all inter-area destinations are calculated by examining the summaries of the area border routers. Inter-area routing is analogous to forcing a star configuration on the Autonomous System (AS), with the backbone as the hub and each of the non-backbone areas as spokes.

As the packet exits the IGP domain, the correct ABR to use is chosen in exactly the same way area border routers advertising summary routes are chosen. A similar rule applies to Not-So-Stubby Area (NSSA) routes.

Incremental SPF can be used to calculate summary, external, or NSSA route changes, so that there is no requirement to recalculate the entire SPF tree.

Controlled Calculation Solution

Previously, whenever a topology change summary, external, or NSSA LSA was received, incremental SPF was calculated independently for each LSA, which resulted in increased CPU utilization under transient conditions.

To solve this problem, if the rate of these LSAs is high, and the SPF has been scheduled, the incremental SPF calculation is not done: this reduces CPU utilization.

This is possible because the SPF calculation computes all routes from the beginning. If it is known the SPF has been scheduled, route calculation can be deferred for these LSAs.

SPF Exponential Hold-Time Backoff

SPF exponential backoff provides the ability to make the hold time variable and changing, based on the frequency of topology changes. This enables increased scalability for frequent topology changes. It also provides for faster convergence when topology changes occur at a slower rate.

OSPFv2 and OSPFv3 use an exponential back-off algorithm for hold time. In addition, the hold time maximum and minimum periods can be configured using the `timers spf exp` command.

This feature provides variable intelligent delay for SPF, depending on the load, to increase scalability and convergence speed.

Hold Time Calculation Algorithm

The hold time delay calculation uses an exponential mechanism, as follows.

1. The CurrHold value indicates the current hold time to use. At the start of processing, the CurrHold value is initialized to the value of the MinHold.
2. If a topology change notification is received before the CurrHold time, the SPF calculation is delayed by either the MinHold time, or the CurrHold time since the last SPF calculation, whichever is less. The CurrHold value becomes either the SPF incremental value of its current value, or MaxHold, whichever is less.
3. If the topology change notification is received after the CurrHold time into the SPF incremental value, the CurrHold time becomes the value of the MinHold time.

In these cases, the SPF is delayed by the MinHold time, or CurrHold time since the last SPF calculation, which ever is lesser.

Configuring Hold Times

The `timers spf exp` command allows configuration of maximum and minimum hold times. The timers have a granularity in milliseconds to allow the user to more effectively control the SPF hold time delays. The hold time changes exponentially when a topology change occurs.

The `no timers spf exp` command resets the exponential backoff timers to the default exponential backoff timer values. The default values of the maximum hold (MaxHold) and minimum hold (MinHold) times are 50 seconds and 50 milliseconds, respectively.

©2012 IP Infusion Inc. Proprietary

CHAPTER 4 OSPF Multiple Instance

ZebOS supports multiple OSPFv2 and OSPFv3 instances. Multiple OSPF instances assist in creating administrative separation in a large network to segregate customer traffic and associated settings. Multiple instances are used to create overlay networks in which separate services are routed only towards routers participating in that service, such as voice. The overlay network isolates routes belonging to one service from another service, by exporting routes, applying tags, and filters routes based on tags. Each protocol instance contains a routing table, applied routing policies, a routing table group, and interfaces that belong to that instance.

OSPF Multiple Instance Functionality

This section describes how ZebOS handles multiple OSPF instances.

- A separate Routing Information Base (RIB) is created for each protocol instance.
- Because the administrative distance is the same for all OSPF instances, NSM additionally considers the metric of the route for the route preference rule.
- NSM passes the OSPF instance ID while sending redistribution route information to OSPF.
- The redistribution of one OSPF instance is allowed into another OSPF routing table.
- The redistribution of other Layer 3 protocols into a specific OSPF instance is allowed.
- CLI support is provided for redistribution between OSPF instances and redistribution to a particular OSPF instance.

Multiple OSPFv2 Instances on a Single Interface

ZebOS supports multiple OSPFv2 instances in a single interface by differentiating packets for the various instances sent and received on the same interface. The packet header format is modified so that the authentication type field is divided into an instance ID and authentication type.

Features

ZebOS support for multiple OSPFv2 instances on a single interface includes the features listed below.

- A command option to enable multiple OSPF instances on a single link
- 16-bit authentication-type field is split into an 8-bit instance ID and an 8-bit authentication type
- The ability to compare an incoming instance ID with its own instance ID to bring up the adjacency
- Debugging statements to display instance-ID-related information
- The `show ip ospf interface` command displays information related to all OSPF processes running on the same subnet. This information also contains the instance ID OSPF process
- The `show ip ospf neighbor` displays instance-ID-related information

System Overview

When an OSPF process is enabled in a subnet, it is associated with an instance ID, in the range of 0 to 255. The instance ID is set through the CLI. Similarly, when multiple OSPF processes are enabled in a single subnet, every process has its corresponding instance ID. By default, the instance ID is zero. Instance IDs are encoded in the respective OSPF packet headers when they are sent out. The 16-bit authentication-type field is divided into an 8-bit instance ID and 8-bit authentication type.

Before bringing up the adjacency, the instance IDs are compared: Received packets with an instance ID not equal to one of the configured OSPF instance IDs on the receiving interface are discarded. To support this feature, the `--enable-ospf-multi-inst` configuration option must be selected while compiling ZebOS. This feature is disabled by default. It is enabled using the `enable ext-ospf-multi-inst` CLI command.

System Architecture

Packet Structure

Packets are differentiated for different instances sent and received on the same interface. Each protocol instance is assigned a separate Instance ID. This instance ID is encoded in the packets associated with the OSPF process. The packet structure is illustrated below.

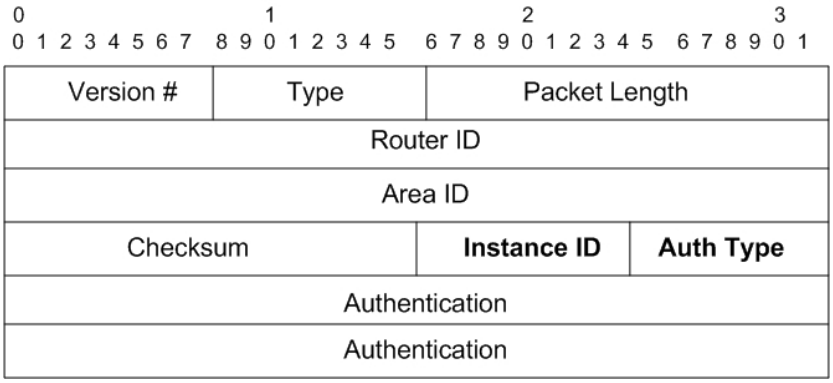


Figure 4: Instance ID Packet Structure

Example Configuration

In the following example, OSPF processes 1, 2, and 3 are enabled on a single link with different instances IDs.

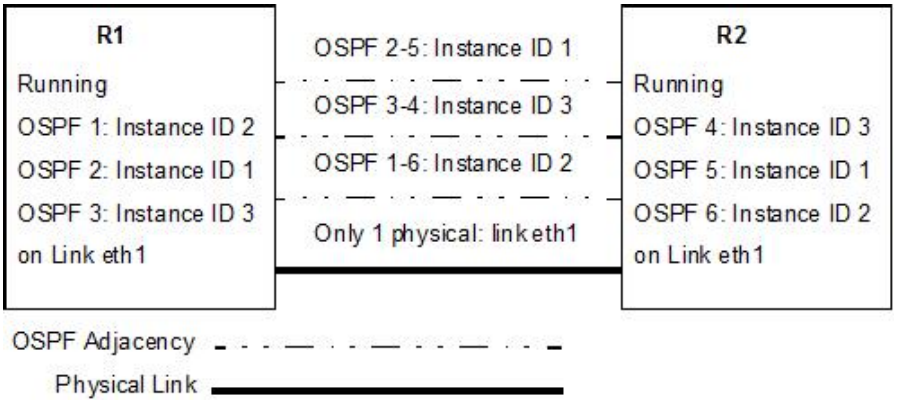


Figure 5: Sample Configuration

©2012 IP Infusion Inc. Proprietary

According to the configuration above, there are the following three OSPF adjacency sessions:

R1		**R2**
OSPF 1	----------	OSPF 6 because instance ID matches
OSPF 2	----------	OSPF 5
OSPF 3	----------	OSPF 4

R1 receives packets from OSPF process 4, 5, and 6 on eth1, but routes packets only from OSPF 6 to OSPF process 1. Similarly, packets from OSPF 5 will be routed only to OSPF 2, and packets from OSPF 4 will be routed only to OSPF 3.

©2012 IP Infusion Inc. Proprietary

CHAPTER 5 OSPF PE-CE for BGP/MPLS VPNs

Using OSPFv2 as the provider-edge/customer-edge (PE-CE) protocol for BGP MPLS VPNs (per RFC 4577) provides easy transition of a service provider's network from the OSPF backbone to the VPN backbone. OSPF routes are redistributed from the remote site of the same area or domain into the CE as Type-3 link-state advertisement (LSA).

Functionality

This feature provides the following:

- CLI support for the OSPF domain identifier and enhancement of the existing CLI for route tagging.
- OSPF and BGP communication via NSM as per section 4.1 of RFC 4577.
- Sets the DN bit for the Type-3 and Type-5 LSAs sent from the PE to the CE.
- Processes the down (DN) bit for LSAs received from the CE to the PE (RFC 4576).
- Use of the route tag to avoid loops for external routes (Type-5 LSA).
- BGP can process and send the OSPF domain identifier type and OSPF router-type extended community attributes, and can store the attribute, along with VRF routes, when present.

System Overview

Prior to version 7.7, ZebOS supported OSPF as the PE-CE protocol, however, routes were redistributed from the remote site of the same area or domain into the CE as a Type-5 LSA (AS external route). Beginning with Release 7.7, these routes are delivered in Type-3 LSAs (as inter-area routes), so that they can be distinguished from any AS external routes that may be circulating in the VPN (that is, they can be distinguished by OSPF from routes that do not come from within the VPN).

Traditional OSPF-BGP Routing

The following is an example of traditional OSPF-BGP distribution.

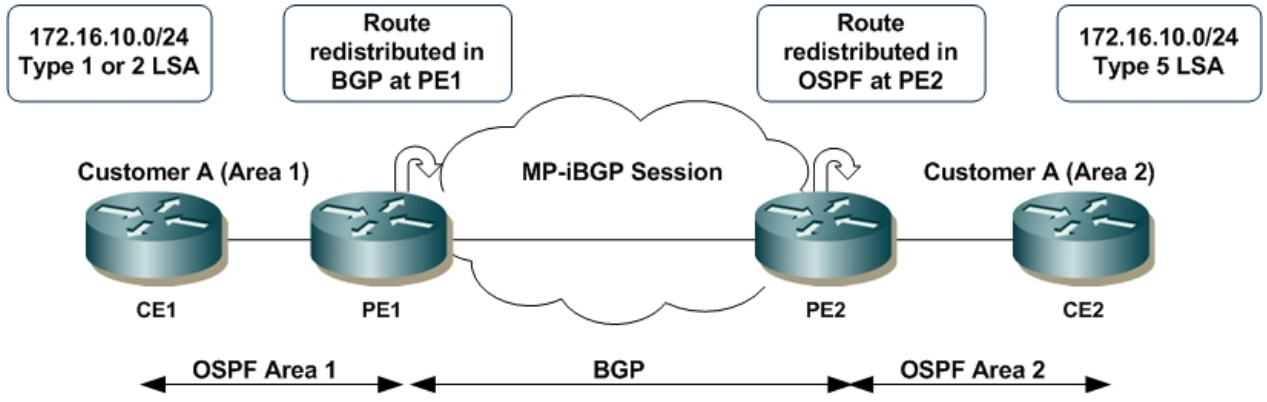


Figure 6: Traditional OSPF-BGP Routing

In an MPLS VPN environment, customer networks are connected to an MPLS VPN enabled provider backbone. As shown above. Customer A areas, Areas 1 and 2, are now connected to an MPLS VPN enabled provider network. Areas 1 and 2 have routers CE1 and CE2 running the OSPF routing protocol. MP-iBGP is used between PE1 and PE2 to propagate routes between Site 1 (Area 1) and Site 2 (Area 2). Traditional OSPF-BGP redistribution is performed at PE routers, PE1 and PE2. The figure above depicts the following sequence that occurs in traditional OSPF-BGP redistribution:

1. Network 172.16.10.0/24 is advertised to the PE1 router by CE1 as a Type-1 or Type-2 LSA.
2. Traditional OSPF-BGP route redistribution occurs where 172.16.10.0/24 is redistributed into BGP at PE1. This route is then propagated as a VPNv4 route to PE2.
3. At PE2, the 172.16.10.0/24 BGP VPNv4 prefix is redistributed in OSPF.
4. This redistributed route (172.16.10.0/24) is propagated as an external LSA Type-5 OSPF route.

Therefore, the OSPF route type, or LSA type, is not preserved when the OSPF route for 172.16.10.0 is redistributed into BGP, when traditional OSPF routing rules are used in an MPLS VPN environment. Also, the following characteristics of OSPF external routes do not allow a smooth transition for a customer attempting to migrate from traditional OSPF routing to the MPLS VPN routing model:

- Internal routes, regardless of their cost, are always preferred over external routes
- External routes cannot be summarized automatically
- External routes are flooded throughout all OSPF areas
- External routes could use a different metric type that is not comparable to OSPF cost
- External LSA Type-5 routes are not inserted in stub areas or not-so-stubby areas (NSSAs)

Routing with OSPF as PE-CE Protocol for BGP MPLS VPNs

The following is an example of using OSPF as the provider-edge/customer-edge (PE-CE) protocol for BGP MPLS VPN.

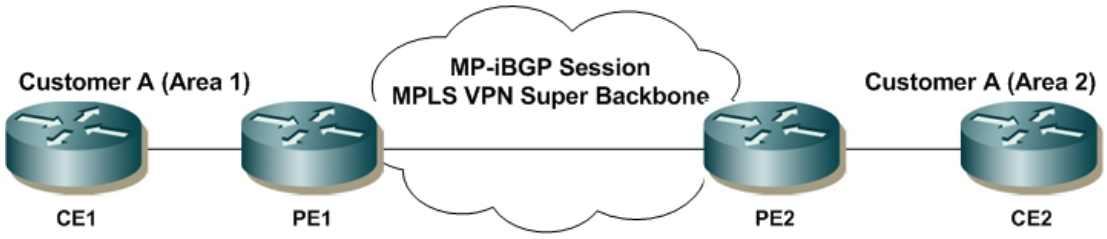


Figure 7: OSPF as PE-CE Protocol for BGP MPLS VPNs

To circumvent the issues posed by the traditional OSPF routing model, the MPLS VPN architecture for OSPF PE-CE routing is expanded to allow transparent customer migration from traditional OSPF routing to the MPLS VPN routing model, by introducing another backbone above the OSPF Area 0. This backbone is called the OSPF or MPLS VPN super backbone.

The non-backbone areas, Area 1 and Area 2, are directly connected to the MPLS VPN super backbone that functions as an OSPF Area 0.

The PE routers, PE1 and PE2, which connect OSPF areas in the customer domain to the super backbone, appear as OSPF Area Border Routers (ABRs) for the devices in the customer OSPF domains. CE routers CE1-A and CE2-A are unaware of any other OSPF areas beyond the MPLS VPN super backbone, because it is transparent.

The MPLS VPN super backbone is implemented using MP-iBGP between PE routers. OSPF information is carried across the MPLS VPN super backbone using BGP extended communities. These extended communities are set and used by PE routers.

©2012 IP Infusion Inc. Proprietary

System Architecture

The domain ID to be assigned to the OSPF instance should be determined so that the route is sent to the other PE router, along with the domain ID value. On the remote PE router, based on the matching domain ID, the route is redistributed as Type-3 LSA (if the domain ID matches) or Type 5-LSA (if the domain ID does not match) to the customer network.

For this purpose, the OSPF domain ID must be assigned for each OSPF instance in the PE router. This information, along with the route type, is carried over to the other PE, and this information is used to identify the exact route type to be sent to the customer network.

The ZebOS modules modified for this requirement are OSPF, BGP, and NSM. The figure below follows the communication between OSPF to NSM and BGP to NSM.

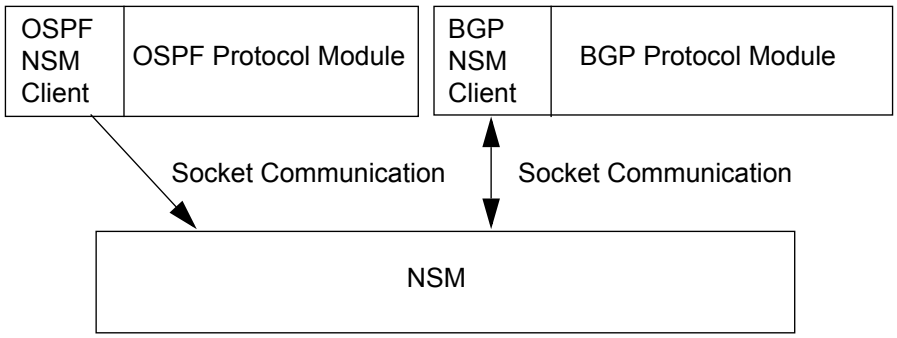


Figure 8: Communication Model

The OSPF domain ID is unique for each VRF instance. The OSPF instance is bound to a VRF using the `router ospf vrf` command.

The OSPF module sends a message to the NSM module to update the NSM RIB with a domain ID value.

When the `redistribute ospf` command is given in BGP at the ingress PE, the OSPF route information, along with the domain ID and route-type information, is sent by NSM to BGP, which further sends this VPN route information to the remote PE router. The domain ID value and route type are sent in the form of BGP extended-community attributes to the remote PE. Similarly, when the `redistribute bgp` command is given inside OSPF at the egress PE, the VPN routes received with the matching domain ID are sent with route type received (either Type-3 or Type-5 LSA). Routes with non-matching domain IDs are sent as Type-5 LSA to the customer routers.

Routing loops are prevented in OSPF with the down bit (DN bit) and route tags.

One of the options bit in the LSA header is allocated to be the DN bit. When a Type-3, -5, or -7 LSA is sent from a PE to a CE, the DN bit is set. The DN bit is clear in all other LSA types (per RFC 4576).

The following illustrates the Options field with the DN bit.

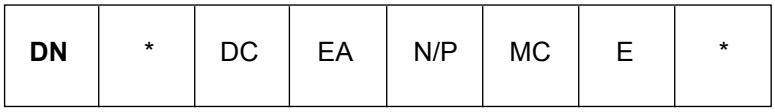


Figure 9: Options Field

The DN bit is used between the PE routers to indicate which routes were inserted into the OSPF topology database from the MPLS VPN super backbone, and thus, do not allow redistribution of the route back into the MPLS VPN super backbone. The PE router that redistributes the MP-BGP route as an OSPF route into the OSPF topology database sets the DN bit. Other PE routers use the DN bit to prevent this route from being redistributed back into MP-BGP.

The DN bit stops the routing loops between MP-BGP and OSPF. It cannot, however, stop the routing loops when redistributing the routes between multiple OSPF domains (that is, in the case of external routes). These routing loops are solved using route tags.

The following is an example of using the `redistribute` command with the `tag` value to avoid routing loops in OSPF.

```
ZebOS# configure terminal
ZebOS(config)# router ospf 10
ZebOS(config-router)# redistribute bgp tag 355
```

CLI Usage

The OSPF domain can have multiple domain IDs. Configure one as the primary domain ID and the remaining as secondary domain IDs. The primary domain ID is sent to the remote PE with the BGP extended community attributes.

If a secondary domain ID is configured without a primary domain ID, an error occurs, and this message is displayed: “configure primary ID first”.

The domain ID value is NULL in two cases:

- When the domain ID is configured as NULL through the CLI
- When no domain ID is configured, its default value will be NULL

When the domain ID chosen is NULL, OSPF checks the previous configuration of domain ID values in the list. If present, an error occurs, indicating: “non-zero domain-id values exists”. Otherwise, the primary domain ID is set to NULL.

For example: When router PE1 receives a route from remote router PE2, with domain ID, d1, the PE1 router compares this route’s domain ID, d1, with the set of domain IDs present with its OSPF instance. If it matches, it sends the route as the type specified in the Route-Type extended-community. If it does not match, it sends the route as Type-5 LSA.

The `domain-id` CLI should be configured before bringing up an OSPF-VRF session with the CE router. In this way, the OSPF routes received from the CE are stored in NSM, along with domain ID value. In turn, when BGP redistributes these OSPF routes into the VPN cloud, it takes the routes along with the domain ID.

©2012 IP Infusion Inc. Proprietary

CHAPTER 6 OSPF-TE Extensions for GMPLS

Traffic engineering (TE) extensions for Open Shortest Path First (OSPF) provide a way to describe the traffic engineering topology and distribute this information within a given OSPF area. Usually, traffic engineering topology exactly matches the physical topology.

Overview

Generalized MPLS (GMPLS) now provides complete separation between the control and data planes for various networking layers, and presents a more general view of TE links. A TE link between a pair of LSRs does not imply the existence of an IGP adjacency between the LSRs. Therefore, OSPF needs to maintain both control links (over which an OSPF adjacency is formed) and traffic engineering links that are used to carry LSPs.

Traffic engineering links are announced in a specific area and the information is flooded over the control links in that area. Traffic-engineering information is stored in a traffic engineering database (TED) for the area. The TED can be used by the Constrained Shortest Path First (CSPF) algorithm to compute a path for an LSP.

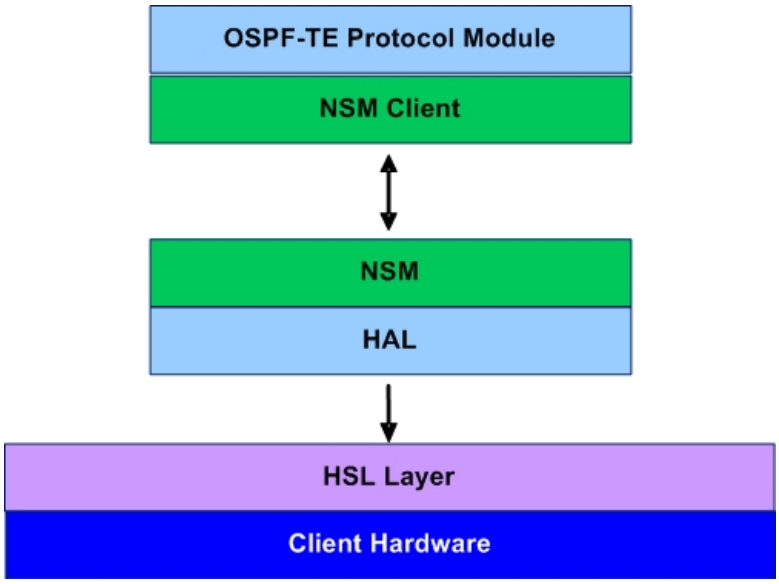


Figure 10: ZebOS OSPF-TE and NSM interaction

Operational Architecture

A TE link is advertised as an adjunct to a standard OSPF or IS-IS link, which means an adjacency is brought up on the link. When the link is up, both the regular IGP properties of the link (the SPF metric) and the TE properties of the link are advertised. OSPF registers for the Interface Service that allows OSPF to get interface information along with the TE information associated with the interface.

Because GMPLS clearly separates the of control and data planes, OSPF cannot get TE link information directly from Interface Service. In order to get TE link information, OSPF-TE registers for TE link services from NSM, in addition to interface services. OSPF also provides callback routines for adding, deleting and updating TE links with NSM. OSPF-

TE supports the ability to updated remote link identifiers for TE links to NSM that it learns from its neighbor through a Link-Local TE LSA.

OSPF-TE LSA Extensions

TE links are announced to an OSPF instance in an area. An OSPF instance and area can be configured for each TE link. This information is stored in the TED for the area. If a TE link is UP, it is flooded in areas configured for that TE link. If property changes are communicated from NSM, OSPF updates the TE LSA and floods it again throughout the area. When a TE link is deleted or its status is DOWN, it is withdrawn from the area.

ZebOS supports the capability to enable MPLS and GMPLS at the same time; therefore it is possible to have a traffic engineering database consisting of MPLS links and GMPLS TE links.

Graceful Restart Extensions

During Graceful restart, TE links are advertised with an unreserved bandwidth of 0 and TE metric of 2^{32} to avoid establishing a new LSP through the restarting router. Neighbors of the restarting node continue advertising the actual unreserved bandwidth of the TE links from the neighbors to that node. Since control and TE topology is separate and LSPs are carried on TE links, graceful restart is not impacted if there is any change to properties of a TE LSA; as a result a new TE LSA is generated or received.

Interfaces

The OSPF-TE protocol module interacts with:

- Management (CLI-API)
- Service APIs, so the OSPF-TE module can interact with NSM

©2012 IP Infusion Inc. Proprietary

API Calls

The following APIs are called by the CLI commands used to configure OSPF TE links.

ospf_telink_te_metric_set

This call is used to set the traffic engineering metric for a TE link.

API Call

```
int ospf_telink_te_metric_set (u_int32_t vr_id, char *tlink_name, u_int32_t metric);
```

Input Parameters

`vr_id` The Virtual Router ID

`tlink_name` The TE link name

`metric` The metric value to set

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the call is successful

OSPF_API_SET_ERR_PROCESS_ID_INVALID when process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the process cannot be found

OSPF_API_SET_ERR_VR_NOT_EXIST when the VR cannot be found

OSPF_API_SET_ERR_ABR_TYPE_INVALID when the ABR type is not valid

ospf_telink_te_metric_unset

This call is used to set the traffic engineering metric for a TE link to default value.

API Call

```
int ospf_telink_te_metric_unset (u_int32_t vr_id, char *tlink_name)
```

Input Parameters

`vr_id` The Virtual Router ID.

`tlink_name` The TE link name

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the call is successful

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the process cannot be found

OSPF_API_SET_ERR_VR_NOT_EXIST when the VR cannot be found

OSPF_API_SET_ERR_ABR_TYPE_INVALID when the ABR type is not valid.

ospf_telink_flood_scope_set

This call is used to set the flooding scope (area and OSPF instance) of a specified TE link

API Call

```
int ospf_telink_flood_scope_set (u_int32_t vr_id, char *tlink_name,
                                 int proc_id, struct pal_in4_addr area_id);
```

Input Parameters

`vr_id` The Virtual Router ID.

`tlink_name` The TE link name

`proc_id` Process ID

`area_id` The area into which TE links should be flooded

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the call is successful

OSPF_API_SET_ERR_PROCESS_ID_INVALID when process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the process can-not be found

OSPF_API_SET_ERR_VR_NOT_EXIST when the VR cannot be found

OSPF_API_SET_ERR_ABR_TYPE_INVALID when the ABR type is not valid

ospf_te_link_flood_scope_unset

This call is used to set the flooding scope of a specified TE link to a default value.

API Call

```
int ospf_te_link_flood_scope_unset (u_int32_t vr_id, char *tlink_name, int proc_id,
struct pal_in4_addr area_id);
```

Input Parameters

`vr_id` The Virtual Router ID.

`tlink_name` The TE link name

`proc_id` Process ID

`area_id` The area into which TE links should be flooded.

Output Parameters

None

 ©2012 IP Infusion Inc. Proprietary

Return Value

OSPF_API_SET_SUCCESS when the call is successful

OSPF_API_SET_ERR_PROCESS_ID_INVALID when process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the process cannot be found

OSPF_API_SET_ERR_VR_NOT_EXIST when the VR cannot be found

OSPF_API_SET_ERR_ABR_TYPE_INVALID when the ABR type is not valid

ospf_opaque_te_link_local_lsa_enable

This call is used to enable exchange of TE link local LSA for a specified TE link.

API Call

```
int ospf_opaque_te_link_local_lsa_enable (u_int32_t vr_id, int proc_id,
                                          char *tlink_name,)
```

Input Parameters

`vr_id` The virtual router ID

`tlink_name` The TE link name

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the call is successful

OSPF_API_SET_ERR_PROCESS_ID_INVALID when process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the process cannot be found

OSPF_API_SET_ERR_VR_NOT_EXIST when the VR cannot be found

OSPF_API_SET_ERR_ABR_TYPE_INVALID when the ABR type is not valid

ospf_opaque_te_link_local_lsa_disable

This API is used to disable exchange of TE link local LSA for a specified TE link.

API Call

```
int ospf_opaque_te_link_local_lsa_disable (u_int32_t vr_id, int proc_id
                                           char *tlink_name,)
```

Input Parameters

`vr_id` The virtual router ID

`tlink_name` The TE link name

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the call is successful

OSPF_API_SET_ERR_PROCESS_ID_INVALID when process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the process cannot be found

OSPF_API_SET_ERR_VR_NOT_EXIST when the VR cannot be found

OSPF_API_SET_ERR_ABR_TYPE_INVALID when the ABR type is not valid

©2012 IP Infusion Inc. Proprietary

CHAPTER 7 Passive Interface

The passive-interface feature simplifies the configuration of distribution routers and allows network managers to obtain routing information from the interfaces in large Internet Service Provider (ISP) and enterprise networks.

Overview

The `passive-interface` command can put all interfaces, or a particular interface, into passive mode. If no interface name or IP address is used with this command, all interfaces are put in passive mode. If an interface name or IP address is used with this command, a particular interface is put in passive mode. The `no` form of this command removes all interfaces from passive mode if no interface name or IP address is specified, or removes a particular interface from passive mode if an interface name or IP address is specified. For details on this command, refer to the *OSPF Command Reference*.

Issue the `passive-interface` command to stop routers from becoming OSPF neighbors on a particular interface. In ISP and large enterprise networks, many of the distribution routers have a large number of interfaces. Configuring passive interface on each of the interfaces can be difficult. The `passive-interface` command (without any interface name) can be used to solve this problem by configuring all the interfaces as passive using a single command.

System Architecture

The following provides a configuration example, then describes configuration without and with the default option for passive interface.

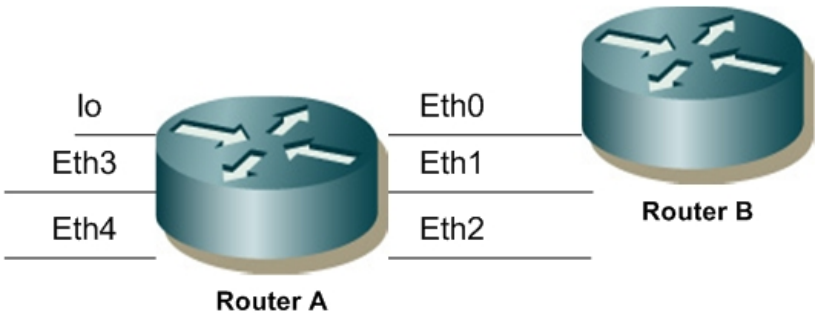


Figure 11: Default Passive Interface

In the illustration above, Router A and Router B are connected with interface Eth0, and the other interfaces are connected to other networks.

Without Default Passive Interface

Before the default passive-interface feature, there were two methods to obtain routing information of connected interfaces of Router A:

- Configure OSPF on Router A and redistribute connected interfaces. As a result, a large number of Type-5 LSAs can be flooded over the domain.
- Configure OSPF on all interfaces and manually set most of them as passive. As a result, large Type-1 link-state LSAs might be flooded into the area. The Area Border Router (ABR) creates Type-3 LSAs, one for each Type-1

LSA, and floods them to the backbone. It is possible, however, to have unique summarization at the ABR level, which injects only one summary route into the backbone, thereby reducing processing overhead.

The solution to this problem was to configure OSPF on all interfaces and manually set the `passive-interface` command, with the interface name on the interfaces where adjacency was not desired. In the above illustration, the user had to run the `passive-interface` command with the interface name for all interfaces (Eth1, Eth2, lo, Eth3, Eth4), except Eth0.

With Default Passive Interface

With the default passive-interface feature, this problem is solved by setting all interfaces to passive by default using a single `passive-interface` command, then configuring individual interfaces where adjacencies are desired using the `no passive-interface` command with selected interface names or IP addresses.

Functions

The following sections list functions added or modified in the OSPFv2 and OSPFv3 modules to support default passive-interface.

OSPFv2

- `ospf_config_write_instance` in `ospf_cli.c` is modified to display the passive-interface option
- `ospf_nsm_if_add` in `ospf_nsm.c` is modified to call the routine that puts the dynamically added interface into passive mode if `passive-interface` is set
- `ospf_passive_if_add_by_interface` in `ospfd.c` is added to put the dynamically added interface into passive mode if `passive-interface` is set
- `ospf_proc_init` in `ospfd.c` is modified to initialize the `passive_if_default` option to PAL_FALSE
- `ospf_passive_interface_default_set` in `ospf_api.c` is added to put all interfaces into passive mode
- `ospf_passive_interface_default_unset` in `ospf_api.c` is added to allow routing updates on all interfaces

OSPFv3

- `ospf6_config_write_instance` in `ospf6_cli.c` is modified to display the passive-interface option
- `ospf6_nsm_if_add` in `ospf6_nsm.c` is modified to call the routine that puts the dynamically added interface into passive mode if `passive-interface` is set
- `ospf6_passive_if_add_by_interface` in `ospfd6.c` is added to put the dynamically added interface into passive mode if `passive-interface` is set
- `ospf6_proc_init` in `ospfd6.c` is modified to initialize the `passive_if_default` option to PAL_FALSE
- `ospf6_passive_interface_default_set` in `ospf6_api.c` is added to put all interfaces into passive mode
- `ospf6_passive_interface_default_unset` in `ospf6_api.c` is added to allow routing updates on all interfaces

©2012 IP Infusion Inc. Proprietary

CHAPTER 8 VLOG Support in OSPF

In older implementations of ZebOS, there were several limitations related to debugging Virtual Routers (VR):

- Debugging could be enabled on a VR, but debug information could not be viewed on the VR.
- Debugging could not be enabled for protocols like IS-IS, OSPF, or RIP on individual VRs.
- A Privileged VR (PVR) user could view their own VR debug messages, or all VR debug messages, but could not differentiate per-VR messages.
- VR users could not forward debug output to a log file, because logging was disabled on all VR, and could only be enabled for PVR.
- Non-PVR users could view debug logs, but could not distinguish what message(s) belonged to which VR.

VLOG Features

- A non-PVR user may enable debugging on a VR where the user is logged in
- A non-PVR user can view debug information for the VR where the user is logged in
- A PVR user may enable debugging in the PVR context, and other global debugging that is not VR-specific
- A PVR user is able to view all ZebOS debugging, including debugging information generated in the context of non-privileged VRs
- Excludes log throttling (duplicate debug messages are not handled)
- Excludes user permissions for Log file

Debug Support for Protocol Modules

- A VLOG build allows operators to enable debugging in a specific VR context for BGP, OSPFv2, OSPFv3, IS-IS, RIP and RIPng.
- The commands entered in global VR configure mode allow a VR user to configure a system for a specific type of debugging.
- VR-specific debug output can be written to the terminals where a VR user has logged in.
- A PVR user may view all debug output generated by ZebOS, including that generated in the context of specific virtual routers.
- ZebOS debugging for protocol modules OSPFv2, ISIS, OSPFv3, RIPng, RIP and BGP is now VR context-sensitive.

VR Builds

When it comes to the use of VLOGD, the following builds and operational modes are possible:.

Non-VR Build

A non-VR build uses the debugging and logging functionality already present in ZebOS.

VR Build with VLOG Disabled

A VR build with VR disabled also uses the debugging and logging functionality already present in ZebOS.

VR Build with VLOG Enabled

To generate this build, two options are required to be enabled: `--enable-vr` and `--enable-vlogd`.

Note: For details on ZebOS Build Options, refer to the *ZebOS Network Platform Installation Guide*.

VLOG Users

There are two types of VLOG users, VR users and PVR users.

VR Users

VR users may:

- Enable or disable VR debugging
- View the debug messages of its own VR
- Log the debug messages to a specified log file
- Specify a VR log file name; otherwise a default name is used.

PVR Users

PVR users may:

- Enable or disable PVR and VR (after logging in) debugging
- From a PVR terminal session, view the PVR and any VR debug output
- Log the debug output for both the PVR and all VRs, to a log file
- Specify a local or global log file name, otherwise, a default name is used

©2012 IP Infusion Inc. Proprietary

VLOG Support for OSPFv2

Set Virtual Router Context

To make all VR debug commands context-specific, the following macro is called:

```
#define LIB_GLOB_SET_VR_CONTEXT(LIB_GLOB, VR_CXT) \
  do {                                            \
    ((LIB_GLOB)->vr_in_cxt) = (VR_CXT);           \
  } while (0)
```

The macro is defined in lib.h. It passes arguments in `lib_global (ZG)` and the pointer for binding VR from `ospf_master`. This sets the VR context (`VR_CXT`) in lib_globals (`LIB_GLOB`).

To set VR context for all internal or external events, it is necessary to identify specific events originating from the OSPF task. OSPF functions that need to be modified as a result of IFSM or NFSM events and timers include:

- OSPF packet processing
- Execution of ISFM or NSFM event processes
- OSPF SPF calculation
- Updating ASBR status
- Installing LSA
- Timers

When a specified event occurs, a determination is made as to whether a particular debug option is enabled in the ospf_master database. If the debug option is enabled, the debug message (error, warning, informational) is displayed using either the `zlog_info`, `zlog_err` or `zlog_warn` function, which redirect the message to `VLOGD`. The VLOG module is fully responsible for displaying debug messages to the terminals or log files for VR-specific debugging.

Example

In VR1, the administrator enables a debug message, in this case, `ospf ifsm timer`, by entering the commands shown below:

```
ZebOS# login virtual-router vr1
localhost.localdomain>debug ospf ifsm timers
localhost.localdomain>terminal monitor
```

When the Hello interval configured in OSPF expires, the debug output message is displayed on the terminal, and the VR context is set in lib_global by calling the following macro `(LIB_GLOB_SET_VR_CONTEXT)`:

```
#define IS_DEBUG_OSPF(A, B)                      \
        (om->debug.term.A & OSPF_DEBUG_ ## B)
```

The last line of the macro, for this example, is replaced with `om->debug.term.ifsm & OSPF_DEBUG_IFSM`. When the macro returns a value of `1`, the debug flag is set, otherwise it is unset.

Once VLOG is enabled, when `zlog_info` is called, the VLOG module writes the output message to the terminal where the VR administrator is logged in. VLOG manages the VR context by using the `lib globals->vr_in_cxt->id` (where `id` is the virtual router number) for every debug message.

Debug Commands Per Virtual Router

All debug commands documented in the *ZebOS Network Platform OSPF Command Line Interface Reference Guide* are available to VR administrators.

Debug Flags Per Virtual Router

The ospf_master maintains system-wide configurations and variables for OSPFv2. Debug flags for configuration and terminals are also maintained in the ospf_master database. Each VR maintains an instance of the ospf_master database, therefore the debug flags enabled on one terminal or VR are distinguishable from another.

The diagram depicts VR 01, VR 09 and a PVR. The context of each VR can be seen in its relationship to the ospf_master database, with the debug flags set for it.

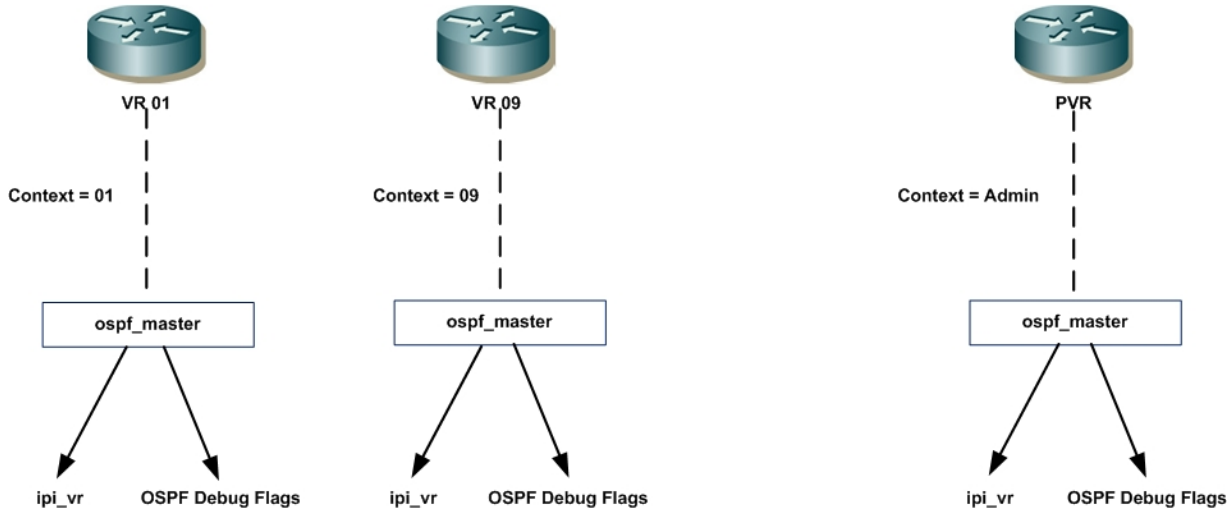


Figure 12: VLOG Support for Virtual Routers

VLOG Support in OSPFv3

Set Virtual Router Context

To make all VR debug commands context-specific, the following macro is called:

```
#define LIB_GLOB_SET_VR_CONTEXT(LIB_GLOB, VR_CXT) \
  do {                                            \
    ((LIB_GLOB)->vr_in_cxt) = (VR_CXT);           \
  } while (0)
```

The macro is defined in lib.h. It passes arguments in `lib_global (ZG)` and the pointer for binding VR from ospf6_master. This sets the VR context (`VR_CXT`) in lib_globals (`LIB_GLOB`).

To set VR context for all internal or external events, it is necessary to identify specific events originating from the OSPF task as a result of timers, packet processing, route calculation, internal operations of LSAs or other events. An example of an external event is a command issued in the CLI.

When a specified event occurs, a determination is made as to whether a particular debug option is enabled in the ospf6_master database. If the debug option is enabled, the debug message (error, warning, informational) is displayed

©2012 IP Infusion Inc. Proprietary

using either the `zlog_info`, `zlog_err` or `zlog_warn` function, which redirect the message to `VLOGD`. The VLOG module is fully responsible for displaying debug messages to the terminals or log files for VR-specific debugging.

VLOG manages the VR context by using the `lib globals>vr_in_cxt>id` (where `id` is the virtual router number) in every debug command.

Debug Commands Per Virtual Router

All debug commands documented in the *ZebOS Network Platform OSPF Command Reference* are available to VR administrators.

Debug Flags Per Virtual Router

In OSPFv3, the ospf6_master maintains system-wide configurations and variables for OSPFv3. Debug flags for configuration and terminals are also maintained in the ospf6_master database. Each VR maintains an instance of the ospf6_master database, therefore the debug flags enabled on one terminal or VR are distinguishable from another.

For a visual representation of the VR in relationship to the ospf6_master file and the OSPF Debug Flags, refer to the diagram above.

©2012 IP Infusion Inc. Proprietary

CHAPTER 9 Constrained Shortest Path First

The Constrained Shortest Path First (CSPF) algorithm calculates an optimum explicit route (ER), based on the specified constraints, using the TED (Traffic Engineering Database) and pre-existing Label Switched Paths (LSP). The resulting ER is used by a signaling protocol (RSVP-TE or CR-LDP) to set up LSPs.

For the set up of traffic-engineered LSPs, ZebOS Network Platform requires these modules:

- IGP-TE (IGP (interior gateway protocol) routing protocol which supports traffic engineering extension (ZebOS uses OSPF-TE)
- TED (Traffic engineering Database)
- CSPF

This document describes the basic architecture of the CSPF module. The ZebOS Network Platform CSPF module is a library that is linked to either the OSPF-TE module or the IS-IS-TE module.

System Architecture

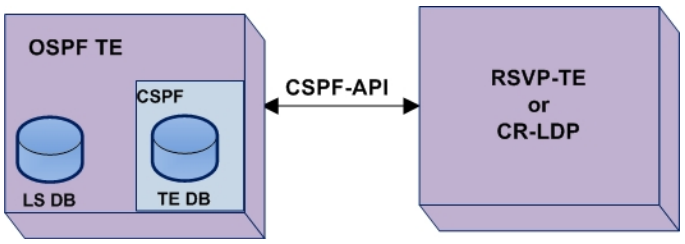


Figure 13: CSPF Related to OSPF

IGP-TE

The algorithm to calculate the CSPF requires router data, network data and traffic engineering data (see IGP-TE data below) for each link. For OSPF, Type-1 (Router-LSA), Type-2 (Network-LSA) or Type-10 (Opaque-LSA with TE information), fill these roles respectively.

In IGP-TE (OSPF or IS-IS), routers exchange the following data and store it in the TED:

Table 1: TED Data

Link type (1 octet)	Link ID (4 octets)
Local interface IP address (4 octets)	Remote interface IP address (4 octets)
Traffic engineering metric (4 octets)	Maximum bandwidth (4 octets)
Maximum reservable bandwidth (4 octets)	Unreserved bandwidth (32 octets)
Resource class/color (4 octets)	

Traffic Engineering Database

Link attributes are exchanged between neighboring OSPF routers using OSPF-TE with Opaque LSA extensions. To calculate CSPF based LSP, router and network information are needed. In the case of OSPF-TE, Router-LSA and Network-LSA are used.

LSP Attributes

The following attributes comprise an LSP:

Egress address. Specifies the address of the egress endpoint of the LSP tunnel: the interface address or the router id of the egress router.

Ingress address. Specifies the address of the ingress endpoint of the LSP tunnel. If no address is specified, the router id of the local router is used. This attribute is optional

Bandwidth. Specifies the bandwidth (in bps) to be reserved for the LSP.

Hop-Limit. (optional) It specifies the maximum hop count between the ingress and egress routers (including egress). The value ranges from 1 to 255 (1 meaning the LSP tunnel end-points are directly connected). The default hop limit is 255. This attribute is optional.

Priority. Control LSP setup and pre-emption. IP Infusion recommends that these two values be the same.

- **Setup Priority**. Sets the relative importance of LSP setup. This attribute sets the order in which CSPF computes the paths for a given set of LSPs. The valid range of values 0-7 (0 is the highest priority). It also sets whether the setup of a new LSP preempts the setup of an LSP already in the queue; an LSP with a higher setup priority preempts LSPs with lower priorities. The default value of 7 means that this LSP cannot preempt the setup of other LSPs. This attribute is optional.
- **Hold Priority**. Determines the priority of an LSP session for holding resources. Hold priority is useful for deciding whether an existing LSP can be preempted by a new LSP. The value ranges from 0 to 7 where 0 is the highest priority level. The default value 0 means that no LSP can preempt this LSP. This attribute is optional.

Administrative Group. (optional). Specifies resource class filters to enforce admission control policies. Each resource (link) is assigned one or more class (color) attributes. The class filters can be of two types:

- **Include Filters**. Specify all groups to include while computing CSPF routes. Only links which match the specified color are included in path computation.
- **Exclude Filters**. Specify the groups to exclude from path computation.

  These filters are represented by a 32-bit vector (one each for include and exclude).

Path Attribute. (optional). Specifies the routers included in CSPF route computations. Each router address is designated as either *strict* or *loose*. A *strict* address in a path signifies (default) that there is no other router between the current router and the specified router. A *loose* address signifies there can be any number of routers between the current router and the specified router.

Tie-Break method. (optional). Tie break method is used to choose a path among equal cost multipaths. There are three types of tie-break methods:-

- **Least-Fill** A path which is least-filled is chosen.
- **Most-Fill**. A path which is most-filled is chosen.
- **Random**. A path is chosen at random. This is the default.

Adaptability. This attribute determines whether an LSP is subject to re-optimization when there is a change in network conditions i.e. new resources become available etc. Re-optimization means that the LSP is rerouted through a better

©2012 IP Infusion Inc. Proprietary

path (defined by some established criteria) in case of network changes. This attribute is disabled by default. There is an associated reoptimization timer which determines how often a cspf computation is undertaken to discover a new route.

LSP Connection Retry Timer. This timer determines how often CSPF makes an attempt to compute a path for a given LSP in case of route computation failures.

LSP Connection Retry Count. This count sets an upper limit to the number of times CSPF does path computation for a given LSP in case of route computation failures.

Shared LSP. This attribute specifies a set of LSPs sharing resources with the LSP for which route computation is desired. This set of LSPs must have the same setup and hold priorities. CSPF takes all such LSPs into consideration while computing a desired route and ensures that on shared links the bandwidth is not counted twice. This helps in conservation of network resources on shared links.

TE Class. This attribute is only available when the Diffserv-TE feature is enabled. It specifies the TE class of the LSP for which a path computation is desired. This feature must be present for successful route computation for DS-TE enabled routers.

CSPF Communications

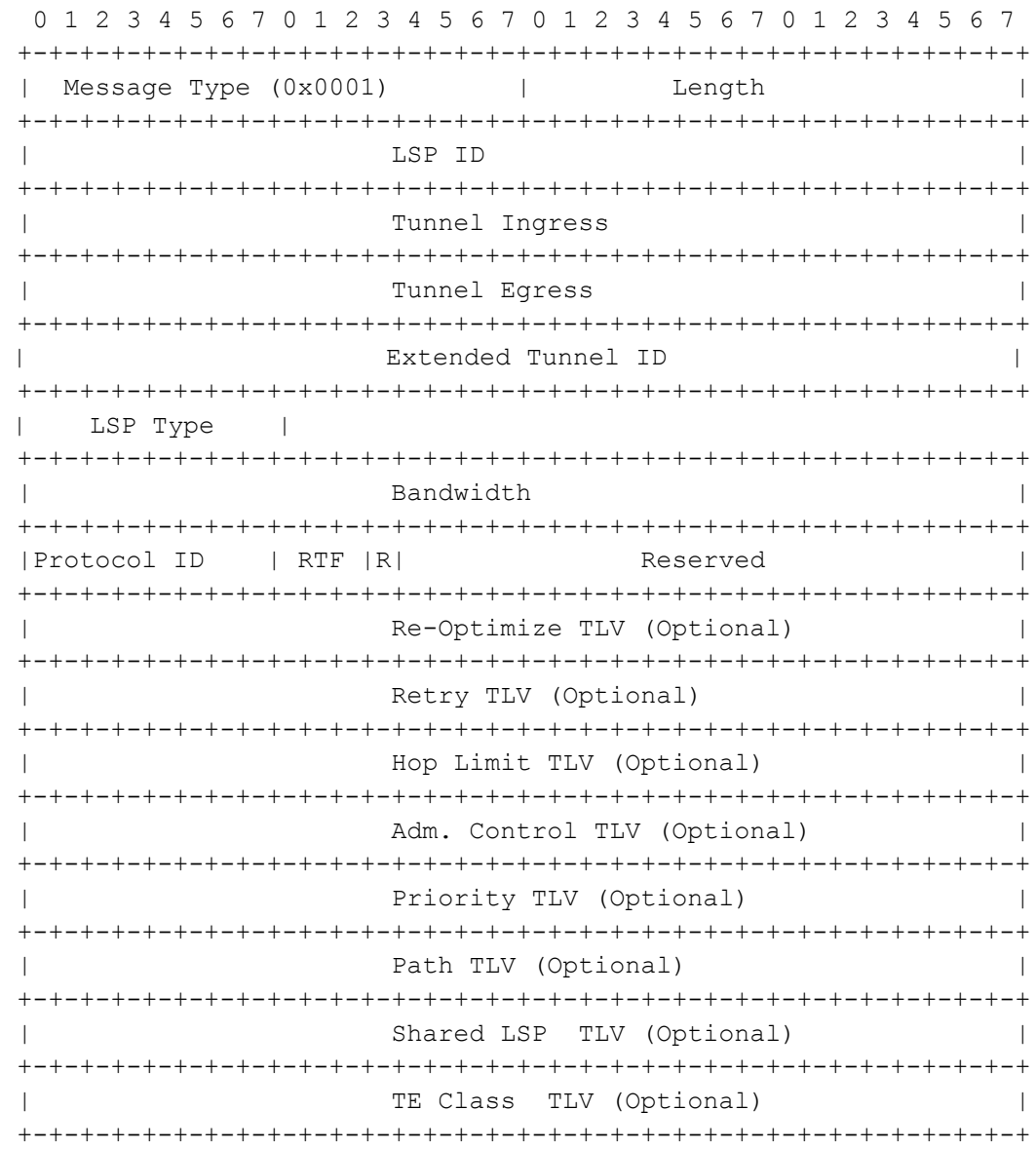
CSPF communicates with a *signaling module* (RSVP-TE/CR-LDP) using the following messages. All the messages are aligned on 32-bit boundaries. The following message structure definitions stipulate how to structure communications with the CSPF module. IP Infusion provides no function calls for this purpose.

Route Request Message

This message is sent from a signaling module to CSPF to request a route computation based on a set of specified constraints. CSPF returns a Route Message is returned when the calculation can be completed; it returns a Notification message when an error occurs. This message should be sent under the following conditions:

- Compute a new route
- Re-compute an old route (possibly with changed attributes)
- Update route attributes which may lead to route recomputation

Message Structure



Field Descriptions

Message Type. This two-byte, unsigned integer identifies the type of message. See Appendix A, CSPF Message Values

Length. This two-byte unsigned integer specifies the length of data excluding the message type and length field. It takes into account all the required fields as well as the optional (type length value) TLV objects.

LSPID. This four-byte unsigned integer is generated by the signaling module to uniquely identify a route request. This value is used to change attributes, request route re-computations and delete given LSPs. This value is also used by CSPF to respond to route requests and for any route updates for a given LSP.

Note: If some signaling module uses 2 octets to identify LSPs, then it should use implementation specific padding bytes to generate a four octet value.

Ingress. This four octet unsigned integer identifies the address of the local router that is used to create an LSP tunnel. It defaults to the router ID if no value is specified.

©2012 IP Infusion Inc. Proprietary

Egress. This four octet unsigned integer specifies the address of the tunnel endpoint. It could be set to the router ID of the egress router or one of the interface addresses.

Note: Care should be taken while using interface addresses to ensure that the interface, corresponding to the egress address, belongs in the same area as the ingress.

Extended Tunnel ID. This four octet unsigned integer specifies the extended tunnel identifier used by signalling protocols.

LSP Type. This octet specifies a primary or a backup LSP. The value 0 signifies primary LSP and the value 1 signifies backup LSP.

Bandwidth. This four octet unsigned integer specifies the bandwidth (in bytes per second) as a constraint for path computation.

Protocol ID. This identifies the signaling protocol (RSVP-TE or CR-LDP). Refer to Appendix A for protocol ID values.

RTF bits. This is a 3-bit flag which specifies the type of route request as discussed previously. The following types are supported.

- **New Route Request**. This specifies a new route computation. The RTF bits should be set to 001.
- **Route Re-computation Request**. This specifies that a re-computation is desired for a pre-existing LSP. It can also be used in conditions where attributes need to be changed for an existing LSP. If no new attribute values are specified, then existing attributes will be used for route computation. To delete an attribute, the corresponding TLV should be specified with the length field set to zero and no TLV data. The RTF bits should be set to 010.
- **Update priority** –

Note: Flag bits should be set appropriately for each condition. A signaling module expects either a route message or a notification message (in case of computation failure or other exceptional conditions).

R bit. This bit specifies the resilience attribute of the LSP. It should be set to 1 if path resilience is required.

Reserved.This 20-bit field should be set to zero.

Note: The remaining fields are all optional; they may be included in the message in any order.

Re-Optimize TLV. This is an optional TLV which indicates that route re-optimization feature is required for the given LSP. A re-optimization timer value can be specified for cspf path recomputation; otherwise a default value is used.

Retry TLV. This is an optional TLV which specifies the number of times CSPF should try to compute a route for the given LSP in case of route failure. It also specifies the interval between successive attempts to compute routes. The TLV length should be set to zero and no TLV data should be set if default values for retry interval and retry limit are desired. See section “TLVs” for encoding.

Hop Limit TLV. This optional TLV specifies the number of maximum hops that the LSP can traverse. This hop limit does not include ingress router. See section “TLVs” for encoding.

Adm. Control TLV. This optional TLV specifies the set of resource classes to be included and excluded for path computation. See section “TLVs” for encoding.

Priority TLV. This optional TLV specifies the setup and holding priority of the given LSP. See section “TLVs” for encoding.

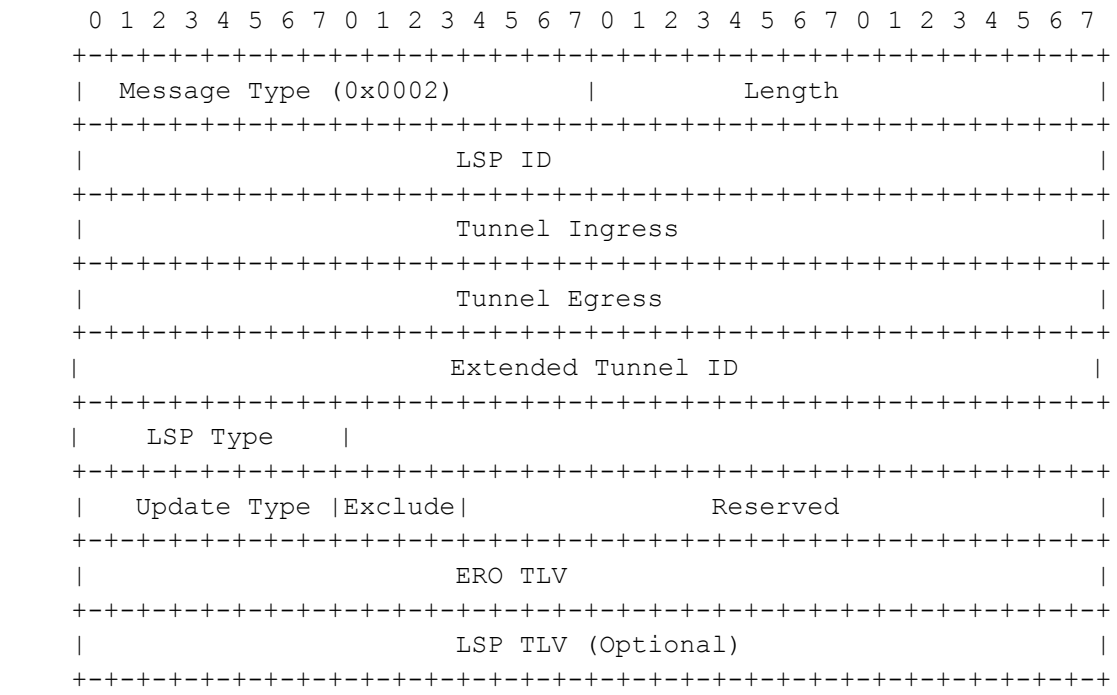
Path TLV. This optional TLV specifies the routers that must be included in the CSPF path computation. See section “TLVs” for encoding.

Shared LSP TLV. This optional TLV specifies a list of LSPs sharing network resources with the LSP for which a route computation is desired. See the *CSPF Communications* section for encoding.

TE Class TLV. This optional TLV is only available when Diffserv-TE feature is enabled. It specifies the TE class of the given LSP. See the *CSPF Communications* section for encoding.

Route Message

This message is generated by CSPF module in response to a route request message, if a route is successfully computed. It contains the explicit path to be used for LSP setup.



- **Message Type** – This two-byte unsigned integer identifies the type of message. Refer to Appendix A for message types.
- **Length** – This two-byte unsigned integer specifies the length of data excluding the message type and length field. It takes into account all the required fields as well as the optional TLV objects.
- **LSPID** – This four-byte unsigned integer identifies the route request message corresponding to this route message.
- **Ingress** – This four octet unsigned integer identifies the address of the local router that is used to create an LSP tunnel.
- **Egress** – This four octet unsigned integer specifies the address of the tunnel endpoint. It could be set to the router ID of the egress router or one of the interface addresses.
- **Extended Tunnel ID** - This four octet unsigned integer specifies the extended tunnel identifier used by signalling protocols.
- **LSP Type** - This octet specifies a primary or a backup LSP. The value 0 signifies primary LSP and the value 1 signifies backup LSP.
- **Update Type** – This is a 1 byte field which can take the following values:-
  - **Route Response** – 0x00 This indicates that the route message is in response to a route request message.
  - **Route Optimization** – 0x01 This indicates that the route message is due to automatic route optimization performed by CSPF.
  - **Reroute on Failure** – 0x00 This indicates the existing path for the LSP is not feasible anymore. CSPF checks the feasibility of a given LSP route if the resilience feature is enabled for the same route.

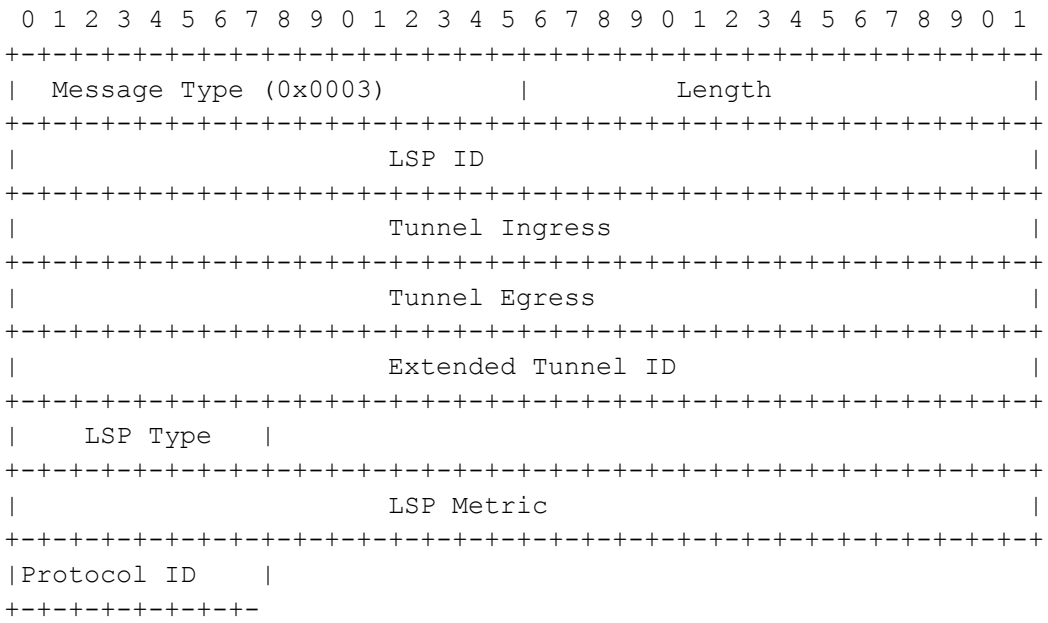
©2012 IP Infusion Inc. Proprietary

- **Exclude Type** - This nibble specifies whether requested node/link protection is available for the given LSP path computation request.
- **Reserved** – This three-byte field should be set to zero.
- **ERO TLV –** This TLV specifies the explicit path for LSP setup. This path has been calculated based on the specified constraints. See section “TLVs” for encoding.
- **LSP TLV** – This is an optional TLV which specifies one or more LSPs which need to be pre-empted for the current LSP setup to be completed. See section “TLVs” for encoding.

  Note: The signaling module may use protocol specific methods to preempt LSPs and create the new LSP or may employ other methods for setting up a new LSP.

LSP Established Message

This message is sent to CSPF to confirm the setup of a signaled LSP. This message should be sent for LSPs which are setup using CSPF (for computation) as well as LSPs which are manually setup (without using CSPF).



Message Type – This two octet unsigned integer identifies the type of message. Refer to Appendix A for message types.

Length – This two-byte unsigned integer specifies the length of data excluding the message type and length field. It takes into account all the required fields as well as the optional TLV objects.

LSPID – This four-byte unsigned integer is generated by the signaling module. This should be the same as the LSPID used in corresponding route request message (for CSPF based LSP setup) or it should be a new unique 32-bit value generated by the SM.

Ingress – This four octet unsigned integer identifies the address of the local router that is used to create an LSP tunnel.

Egress – This four octet unsigned integer specifies the address of the tunnel endpoint. It could be set to the router ID of the egress router or one of the interface addresses.

Extended Tunnel ID - This four octet unsigned integer specifies the extended tunnel identifier used by signalling protocols.

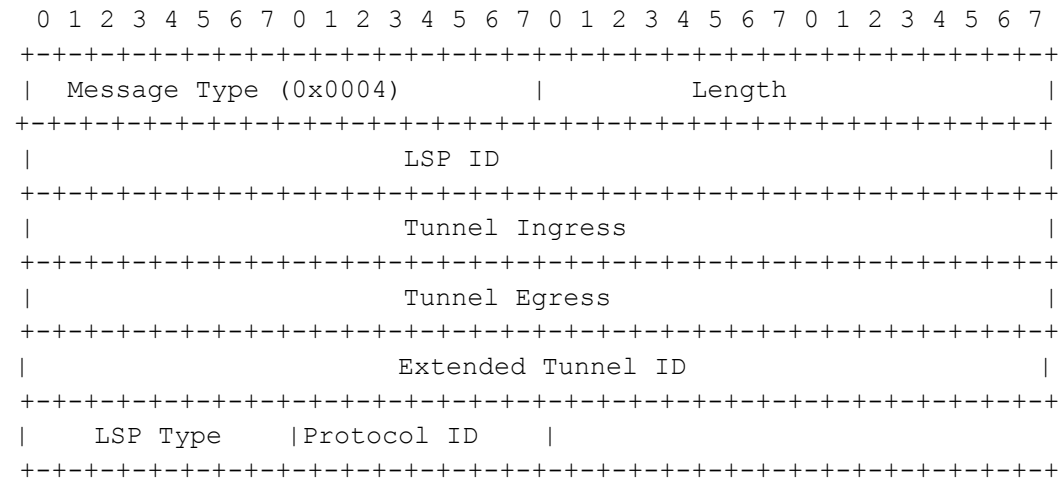
LSP Type - This octet specifies a primary or a backup LSP. The value 0 signifies primary LSP and the value 1 signifies backup LSP.

LSP Metric – This four-byte unsigned integer is used to specify a TE metric for this LSP. This metric may be different from OSPF cost metric.

Protocol ID – This identifies the signaling protocol (RSVP-TE or CR-LDP). Refer to Appendix A for protocol ID values.

LSP Delete Message

This message is sent to CSPF to delete a given LSP (identified by LSP ID).



Message Type – This two-byte unsigned integer identifies the type of message. Refer to Appendix A for message types.

Length – This two-byte unsigned integer specifies the length of data excluding the message type and length field. It takes into account all the required fields as well as the optional TLV objects.

LSPID – This four-byte unsigned integer identifies the LSP to be deleted.

Ingress – This four octet unsigned integer identifies the address of the local router that is used to create an LSP tunnel.

Egress – This four octet unsigned integer specifies the address of the tunnel endpoint. It could be set to the router ID of the egress router or one of the interface addresses.

Extended Tunnel ID - This four octet unsigned integer specifies the extended tunnel identifier used by signalling protocols.

LSP Type - This octet specifies a primary or a backup LSP. The value 0 signifies primary LSP and the value 1 signifies backup LSP.

Protocol ID – This identifies the signaling protocol (RSVP-TE or CR-LDP). Refer to Appendix A for protocol ID values.

Reserved – This three-byte field should be set to zero.

©2012 IP Infusion Inc. Proprietary

Notification Messages

These messages are exchanged between CSPF and signaling module to indicate error or other exceptional conditions. The general format for a notification message is given below.

```
 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Message Type (0x0005)        |         Length                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Status TLV                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Message Type – This two-byte unsigned integer identifies the type of message. Refer to Appendix A for message types.

Length – This two-byte unsigned integer specifies the length of data excluding the message type and length field. It takes into account all the required fields as well as the optional TLV objects.

Status TLV – This TLV describes the type of notification and associated data. See section “TLVs” for encoding.

TLVs

The following TLVs can be used in the messages exchanged between CSPF and signaling modules.

Re-Optimize TLV

This TLV specifies whether re-optimization feature is enabled for a given LSP.

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  TLV Type (0x0100)            |         Length                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Re-Optimization Timer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

TLV Type - This two octet unsigned integer identifies the type of TLV. Refer to Appendix A for TLV types.

Length - This two octet unsigned integer specifies the length of data excluding the TLV type and length field.

Re-optimization Timer - The two octet unsigned integer specifies the time interval (in seconds) between successive CSPF route computation for LSP route optimization.

Retry TLV

This TLV specifies whether CSPF should perform route computation again in case of computation failures and associated retry parameters detailed below.

```
 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  TLV Type (0x0101)            |         Length                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Retry Interval               |         Retry Limit           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

TLV Type – This two-byte unsigned integer identifies the type of TLV. Refer to Appendix A for TLV types.

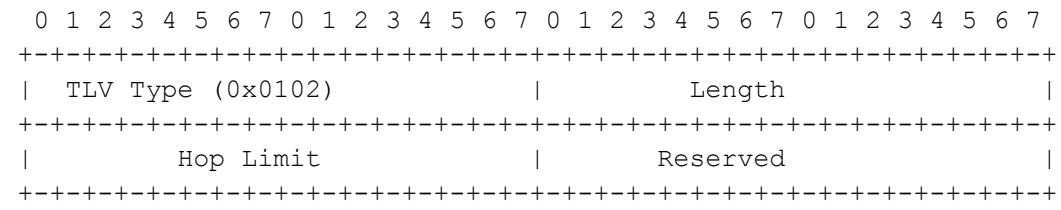
Length – This two-byte unsigned integer specifies the length of data excluding the TLV type and length field.

Retry Interval – This two-byte unsigned integer specifies the time interval (in seconds) between successive route computations in case of computation failure.

Retry Limit - This two-byte unsigned integer specifies the number of times route computations should be done in case of computation failure.

Hop Limit TLV

This TLV specifies the constraint on the number of maximum hops that a computed route can have. This hop limit excludes the ingress router.



TLV Type – This two-byte unsigned integer identifies the type of TLV. Refer to Appendix A for TLV types.

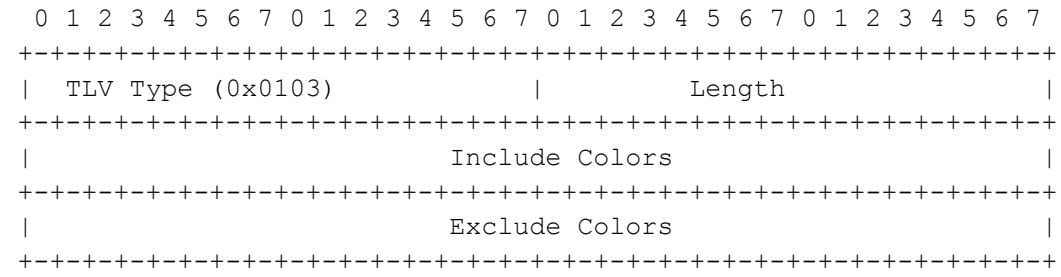
Length – This two-byte unsigned integer specifies the length of data excluding the TLV type and length field.

Hop Limit – This two-byte unsigned integer specifies the constraint on the number of maximum hops that a computed route can have. This hop limit excludes the ingress router.

Reserved – This two-byte field should be set to zero.

Admission Control TLV

This TLV includes admission control information which specifies the set of link colors which need to be included and excluded from route computation.



TLV Type – This two-byte unsigned integer identifies the type of TLV. Refer to Appendix A for TLV types.

Length – This two-byte unsigned integer specifies the length of data excluding the TLV type and length field.

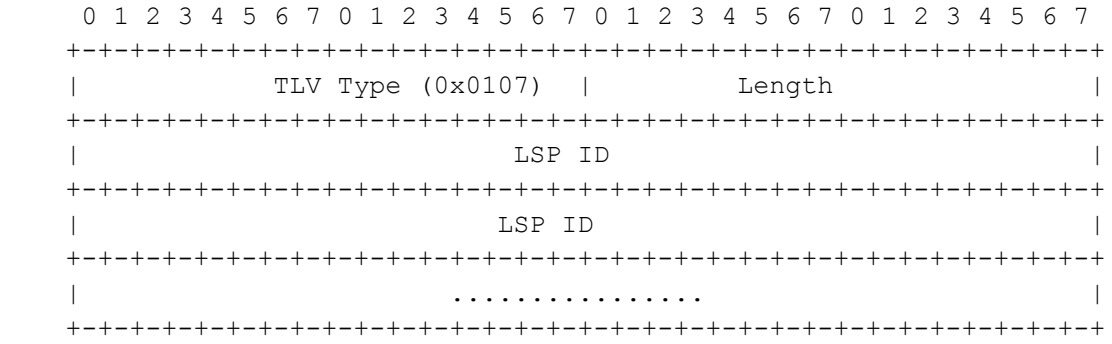
Include Colors – This two-byte unsigned integer specifies the class (color) of resources that must be included in cspf route computation. Each color is represented by a bit position in the 32-bit mask.

Exclude Colors – This two-byte unsigned integer specifies the class (color) of resources which must be excluded in cspf route computation. Each color is represented by a bit position in the 32-bit mask.

©2012 IP Infusion Inc. Proprietary

LSP TLV

This TLV contains list of LSPs. It can be used to convey a list of LSPs which need to be pre-empted.



`TLV Type` – This two-byte unsigned integer identifies the type of TLV. Refer to Appendix A for TLV types.

`Length` – This two-byte unsigned integer specifies the length of data excluding the TLV type and length field.

`LSPID` – LSPID of an LSP

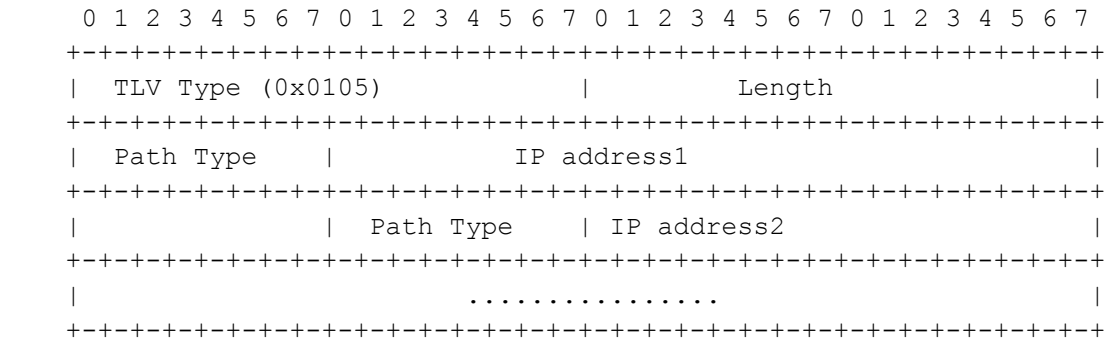
The remaining fields are variable in length.

Shared LSP TLV

This TLV specifies the set of LSPs sharing network resources with the LSP for which a route computation is desired. Refer to the section on LSP TLV for packet details.

Path TLV

This TLV specifies a list of IP addresses which should be included in the route computation in the specified order. Each address can be specified as strict or loose.



TLV Type – This two-byte unsigned integer identifies the type of TLV. Refer to Appendix A for TLV types.

Length – This two-byte unsigned integer specifies the number of IP addresses in the following data. The actual size of TLV data can be calculated by the formula

size = length * 5

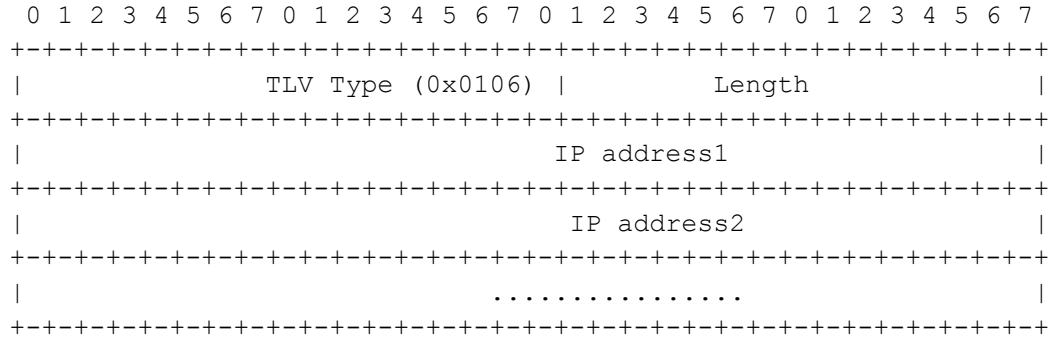
Path Type – Strict or loose

IP address – IP address of node to be considered.

The remaining fields are variable in length.

ERO TLV

This TLV contains a list of IP addresses which form an explicit route from ingress to egress (including the egress router IP address).



TLV Type – This two-byte unsigned integer identifies the type of TLV. Refer to Appendix A for TLV types.

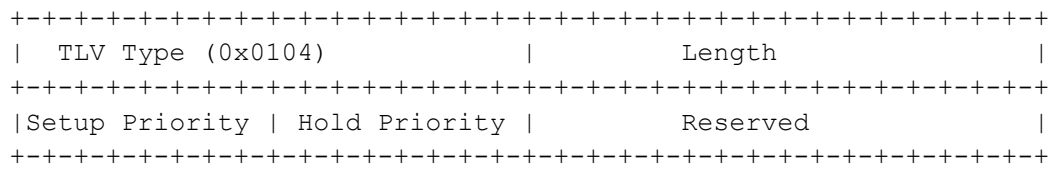
Length – This two-byte unsigned integer specifies the length of data excluding the TLV type and length field.

IP address – IP address of next router.

The remaining fields are variable in length.

Priority TLV

This TLV specifies the setup and hold priorities of the LSP. Each priority can take a value from 0 to 7.



TLV Type – This two-byte unsigned integer identifies the type of TLV. Refer to Appendix A for TLV types.

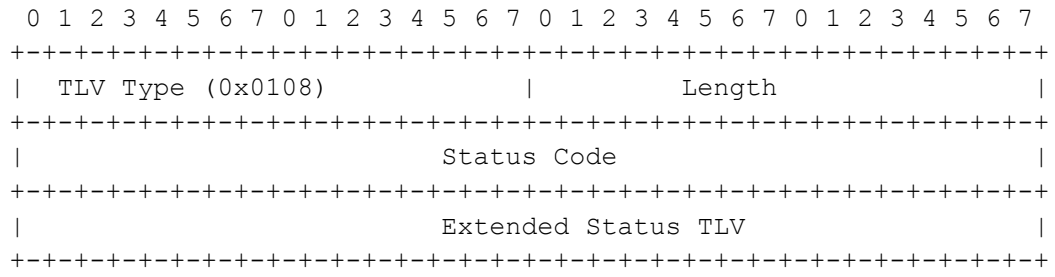
Length – This two-byte unsigned integer specifies the length of data excluding the TLV type and length field.

Setup Priority – Setup priority of LSP.

Hold Priority – Hold Priority of LSP.

Status TLV

This TLV carries status information for error or exceptional conditions.



`TLV Type` – This two-byte unsigned integer identifies the type of TLV. Refer to Appendix A for TLV types.

`Length` – This two-byte unsigned integer specifies the length of data excluding the TLV type and length field.

 ©2012 IP Infusion Inc. Proprietary

`Status Code` – This four-byte unsigned integer specifies the status code for this TLV. The following status codes are defined:-

- SHUTDOWN - This is used to signal to the receiver that the sender is terminating the tcp link.
- ROUTE NOT FOUND - This is used by CSPF to tell the signaling module that the CSPF route computation failed for a given LSP.
- ROUTE FAILURE - This is used by CSPF to notify the signaling module about failure of a route for a given LSP.
- PACKET DECODE FAILURE - This can be sent by either signaling module or CSPF module on receipt of an invalid cspf message (through a route request etc.).
- REQUEST MSG PROCESSING FAILURE - This is sent by CSPF to signaling module if there is an error encountered in processing route request message.
- DELETE MSG PROCESSING FAILURE - This is sent by CSPF to signaling module if there is an error encountered in processing LSP delete message.
- ESTABLISHED MSG PROCESSING FAILURE - This is sent by CSPF to signaling module if there is an error encountered in processing LSP established message.

Shared LSP TLV

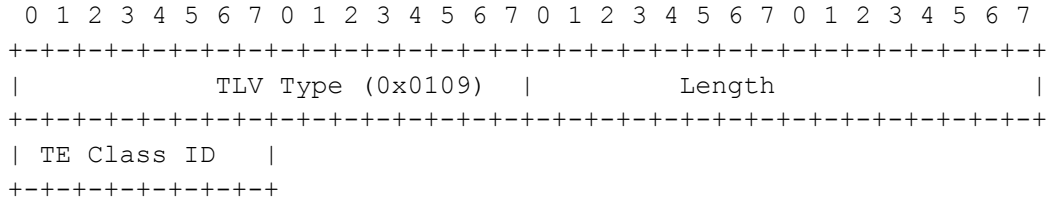
This TLV specifies the set of LSPs sharing network resources with the LSP for which a route computation is desired.

Please refer to the section on LSP TLV for packet details.

TE Class TLV

This TLV is available only if DS-TE feature is enabled.

It specifies the TE class identifier of a given LSP.



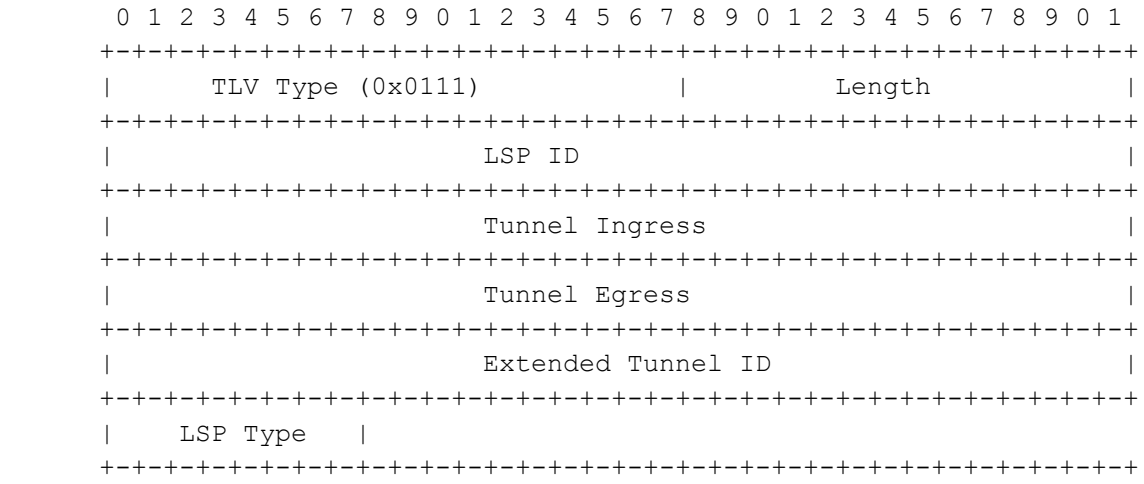
`TLV Type` – This two-byte unsigned integer identifies the type of TLV. Refer to Appendix A for TLV types.

`Length` – This two-byte unsigned integer specifies the length of data excluding the TLV type and length field.

`TE Class ID` – TE class identifier for an LSP.

Extended Status TLV

This TLV carries additional status information for error or exceptional conditions. This is used to augment the information carried by the Status TLV.



Message Type – This two-byte unsigned integer identifies the type of message. Refer to Appendix A for message types.

Length – This two-byte unsigned integer specifies the length of data excluding the message type and length field. It takes into account all the required fields as well as the optional TLV objects.

LSPID – This four-byte unsigned integer identifies the LSP.

Ingress – This four octet unsigned integer identifies the address of the local router that is used to create an LSP tunnel.

Egress – This four octet unsigned integer specifies the address of the tunnel endpoint. It could be set to the router ID of the egress router or one of the interface addresses.

Extended Tunnel ID - This four octet unsigned integer specifies the extended tunnel identifier used by signalling protocols.

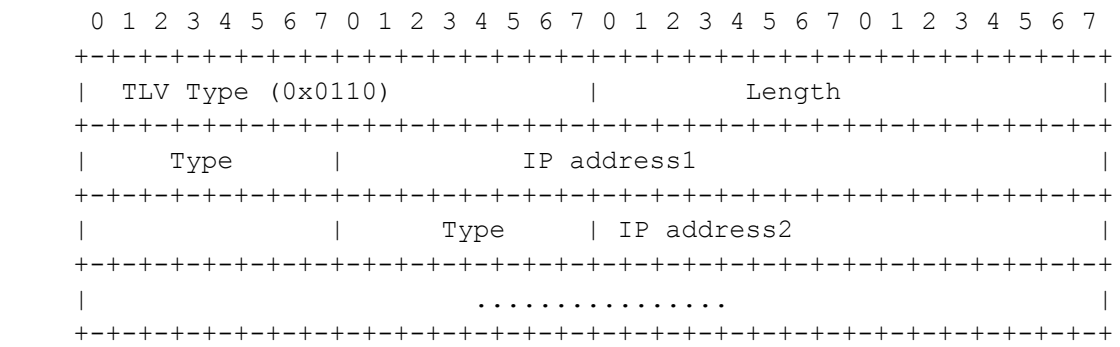
LSP Type - This octet specifies a primary or a backup LSP. The value 0 signifies primary LSP and the value 1 signifies backup LSP.

©2012 IP Infusion Inc. Proprietary

CSPF Modifications for RSVP-TE Fast Reroute

For the RSVP-TE Fast Reroute feature, CSPF has been extended to support another path constraint, the exclude route. A new TLV (Exclude Path) has been added to route request message. This TLV contains a set of IPV4 or IPv6 addresses which should be avoided by CSPF computation engine while computing a constrained route. Each address also contains a flag specifying whether a link on a node or the node itself is to be excluded.

IPv4 Exclude Path TLV



TLV Type – This two-byte unsigned integer identifies the type of TLV. Refer to Appendix A for TLV types.

Length – This two-byte unsigned integer specifies the number of IP addresses in the following data. The actual size of TLV data can be calculated by the formula

size = length * 5

Path Type – Exclude Route Link and Exclude Route Node

- Exclude Route Link - The presence of this path type indicates that the link identified by the specified IP address should be avoided.
- Exclude Route Node - The presence of this path type indicates that the node, to which the specified IP address belongs, should be avoided by the route computation engine. This the default value.

IP address – IP address of node to be considered.

The remaining fields are variable in length.

OSPFv3 CSPF

ZebOS supports traffic engineering extensions to OSPF Version 3 (OSPFv3), and TLVs and sub TLVs for Intra-Area-TE Link State Advertisement (LSA) used in constrained-based SPF calculation.

TLVs and sub TLVs extend TE capabilities to IPv6 networks. The functions defined in RFC 4203 and RFC 3270 are extended to OSPFv3 by using the TLVs and sub-TLVs described in the IETF draft, draft-ietf-ospf-ospfv3-traffic-09.txt.

Intra-Area TE LSA

The Intra-Area-TE LSA payload consists of many TLVs and sub TLVs that can be configured to propagate resource availability information in link-state routing updates. This information is used to perform a constraint-based SPF (CSPF) calculation.

To accommodate OSPFv3, a link-state (LS) type is defined for the Intra-Area TE LSA, with function code 10. The LSA payload consists of one or more nested TLV triplets. There are two applicable top-level TLVs:

- 2 - Link TLV
- 3 - Router IPv6 Address TLV

The Router IPv6 Address TLV advertises a reachable IPv6 address. This stable IPv6 address is always reachable if there is connectivity to the OSPFv3 router.

The Link TLV describes a single link and consists of a set of sub TLVs. All of the sub TLVs in RFC 3630, other than the Link ID sub TLV, are applicable to OSPFv3. The Link ID sub TLV is not used in OSPFv3, due to the protocol differences between OSPFv2 and OSPFv3. Thus, ZebOS uses three sub TLVs for the Link TLV:

- 18 - Neighbor ID (8 octets)
- 19 - Local Interface IPv6 Address (16N octets, where N is the number of IPv6 addresses)
- 20 - Remote Interface IPv6 Address (16N octets, where N is the number of IPv6 addresses)

GMPLS and Differentiated Services

The sub TLVs defined in RFC 4203 and RFC 3270 are extended to OSPFv3. The following sub TLVs are added to the Link TLV:

Sub TLV Type	Length	Name
11	8	Link Local/Remote Identifiers
14	4	Link Protection Type
15	Variable	Interface Switching Capability Descriptor
16	Variable	Shared Risk Link Group

Traffic Engineering Database

The Traffic Engineering Database (TED) stores the traffic engineering and resource availability information. The algorithm to calculate the CSPF requires router, network, and traffic engineering data for each link. For OSPFv3, Type-1 (Router-LSA), Type-2 (Network-LSA) or Type-10 (Inter-Area TE LSA), fill these roles, respectively.

 ©2012 IP Infusion Inc. Proprietary

MPLS LSP Destination Route Deletion

RSVP-TE uses the CSPF module to compute the Explicit Route Object (ERO) for its LSPs. CSPF uses the TE Link State Database (LSDB) populated by the OSPF module in calculating a constraint-based path from the ingress to the egress. Previously, RSVP-TE used its refresh timer mechanism to be notified of failures in reachability to the LSP egress for CSPF based LSPs.

The MPLS LSP Destination Route Deletion feature defines a mechanism for the CSPF server to detect the loss of egress reachability and asynchronously communicates this to the RSVP-TE module. This eventually speeds up the LSP tear-down operation, instead of requiring long intervals for refresh timer expiration.

This section describes the procedures involved in detecting route deletions and the notification process between CSPF and RSVP-TE. It also describes the changes in the code in the OSPF-TE, CSPF, and RSVP-TE modules to accommodate this feature.

Design Overview

The OSPF TE database is used to detect LSP path deletion in two scenarios:

- Handling the CSPF established message from the RSVP-TE to CSPF server
- Handling TE link TLV deletion (triggered by OSPF updates) from the TE LSDB

The above scenarios are explained in detail in the subsections which follow.

The CSPF server stores an address to LSP mapping information in its main structure (`cspf`). Upon TE LSA deletion, OSPF-TE intimates this information to the CSPF server.

The CSPF server notifies the CSPF client, and subsequently the RSVP-TE module, about route changes in the LSP path.

Multiple LSPs may be dependent on the TE link to be deleted. In this case, a single notification is generated with its status TLV containing all the encoded LSP keys. The CSPF_CODE_ROUTE_FAIL status code is used for this notification.

RSVP-TE handles this notification message similarly to the handling of Route Delete messages received from NSM in non-CSPF based LSPs cases. It tears down the LSP by propagating Path Tear messages along the LSP path and sends an LSP Delete message back to the CSPF to update its LSP Table.

LSP Established Messages from RSVP-TE

RSVP-TE sends a Route Request message to the CSPF server to compute an LSP path. CSPF replies to this with a Route message containing the constraint-based hop-by-hop path. The RSVP-TE ingress propagates the PATH message towards the egress, which returns its confirmation with a RESV message. In turn, the LSP is established, and RSVP-TE communicates this establishment to CSPF via an LSP ESTABLISHED message. CSPF receives this message and updates its LSP status.

CSPF creates or updates a binding data structure containing a reference count (`refCount`) that indicates the number of dependent LSPs on this address.

A Patricia tree is keyed on the set of all IP addresses that appear in the LSP path for each LSP. Each node in the tree corresponds to a local address in the `cspf_nexthop_data` structure. The local address is the key because the corresponding local address is always impacted and gets deleted upon link failure or routing changes across the LSP path. Each node stores the `refCount` element that indicates the number of LSPs dependent on the address.

This binding is created only when a link is used for the first time by an LSP. When subsequent LSP(s) use the same link, a search is made for the presence of a node with this address in the Patricia tree. If present, the reference count is incremented.

The following illustration depicts the steps for handling the receipt of an LSP-established message by the CSPF server.

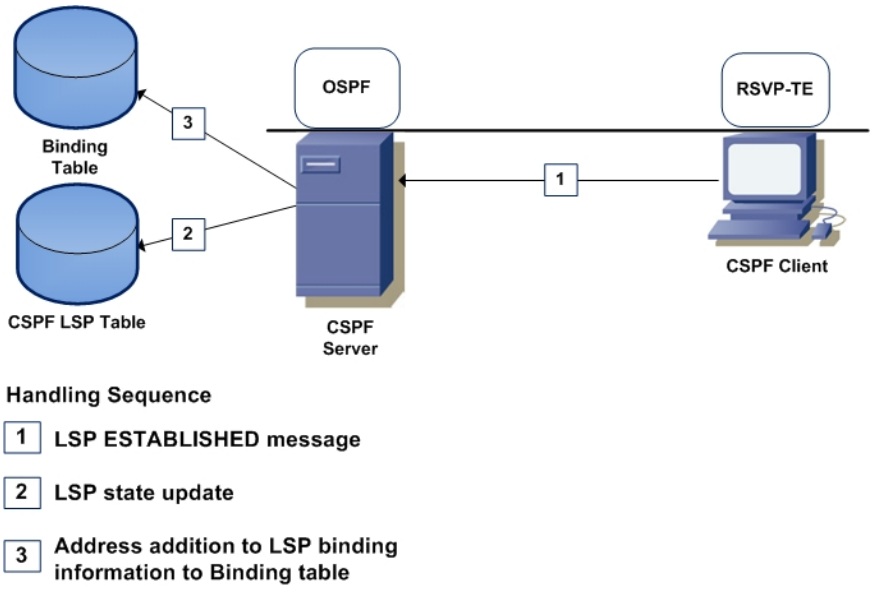


Figure 14: Establish LSP with CSPF

Step 3 demonstrates a significant performance optimization. Without this step, it would be required to compare each prefix in the ERO of each LSP in the LSP table against the local interface in the TE Link TLV being deleted. This would be unnecessary overhead, because not every Link TLV deletion may impact the set of established LSPs in CSPF at any point of time.

TE Link TLV Deletion

TE-Link TLV deletion occurs in response to routing changes detected by OSPF. This deletion may be a consequence of a corresponding interface going down or removal of a network from the OSPF domain. It may be attributed to deletion of the entire TE LSA or only the Link TLV in that LSA.

Receipt of TE Link TLV Deletion is detected by the OSPF module section handling TE operation. The OSPF TE module sends the local address in the link TLV to the CSPF server, before deleting it from the TE LSDB. CSPF then checks for the presence of this information and ascertains whether the deletion would impact active LSP(s): It does this by searching its Binding table, based on the address sent and verifies if a matching entry exists. The reference count is extracted from this node and stored along with the key.

The CSPF server scans each node in the LSP table, starting from the top, inspecting LSP(s) in each node for any matching local address in their route list. On each successful match, the reference count is decremented by 1, and the `cspf_lsp_key` of the matching LSP is added to a list. This operation continues until the reference count becomes 0.

When the reference count becomes 0, all of the affected LSPs, whose LSP keys are available as a list, are identified. CSPF sends out a notification message with the status code set to CSPF_CODE_ROUTE_FAIL, and the status TLV containing the LSP keys encoded as per the list. This indicates that one of the hops along the path is now unreachable, and as a result, signals an asynchronous LSP failure for each of those LSPs to RSVP-TE.

RSVP-TE handles this notification type by performing operations to tear down this session, including propagation of the PATH Tear message towards the egress. This propagation can be as far as the last reachable node in the outcome of the routing change that occurred.

The LSP tear-down mechanism includes dispatch of the LSP Delete message back to the CSPF module to update its local LSP table. The following illustration depicts the sequence of events in handling TE LSDB based route deletions.

©2012 IP Infusion Inc. Proprietary

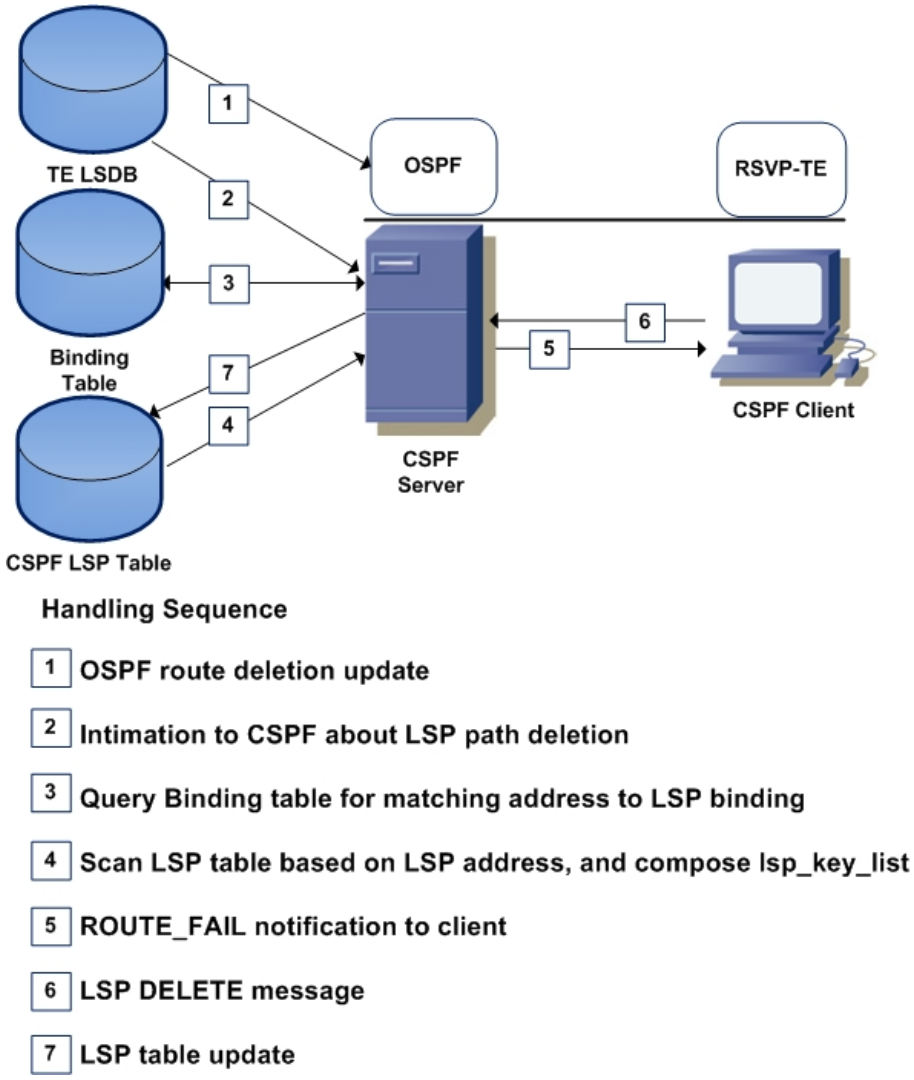


Figure 15: Route Delete Processing

Steps 2 through 5 correspond to the MPLS LSP Destination Route Deletion feature.

Files

The following files were modified to accommodate the MPLS LSP Destination Route Deletion feature:

OSPF-TE

```
ospfd/ospf_te.c
```

CSPF

```
lib/cspf/cspf_common.h
lib/cspf/cspf_common.c
lib/cspf/cspf_server.c
```

RSVP-TE

```
rsvpd/rsvp_cspf.h
rsvpd/rsvp_cspf.c
```

Data Structures

The following sections describe new or modified structures to accommodate MPLS LSP Destination Route Deletion.

Modified Data Structures

cspf

This data structure, referred to as the cspf server data structure, is modified to include the following list element:

```
struct ptree *binding;
```

The `binding` element contains a tree of entries of type `cspf_lsp_addr_binding`. This new data structure, `cspf_lsp_addr_binding`, is detailed in the *New Data Structures* section which follows.

cspf_api_tlv_ext_status

The `lsp_key` member is modified as a pointer to the following structure:

```
struct cspf_lsp_key *lsp_key;
```

This accommodates encoding multiple LSP keys as part of the Status TLV of a notification message, to notify multiple dependent LSPs of a link failure with a single Notification message.

New Data Structures

The `cspf_lsp_addr_binding` data structure is added to the `cspf_server` `cspf_common.h`. This structure is defined as follows:

```
struct cspf_lsp_addr_binding
{
  int refCount;
}
```

The reference count (`refCount`) indicates the number of dependent LSP(s) on a given address, as specified by the key of this node.

©2012 IP Infusion Inc. Proprietary

CSPF API

The CSPF API abstracts the client-server messaging from client signaling protocols. These APIs have been implemented as a linkable library module. The following APIs can be used by a signaling protocol to communicate with CSPF module.

cspf_api_client_new

This routine creates and initializes a new CSPF client instance and initiates a tcp connection to the CSPF module. When the tcp connection is successfully established, it initiates a session connection (by sending Session Connect Message) and notifies the signaling module.

Files

lib/cspf_api.h

Arguments

1. Client ID (unsigned char) - Type of Client (CR-LDP, RSVP-TE or MANAGEMENT)
2. Callbacks (struct cspf_api_callback *) - This structure contains all the callback routines that a signaling module needs to register to get event notifications. Following callbacks are supported :-
    1. Connection reset - This handler is called when the tcp connection to the CSPF module is terminated either locally or by the server.
    2. Connection established - This handler is called when the connection with the server is established.
    3. Route Message Received - This handler is called when a route message is received from the server. The client library decodes the received data and passed the decoded message to the signaling module as an argument to the callback routine. The signaling module should make a local copy of this data
3. Zglobal data (struct zglobals *) - A pointer to global data structure maintained by signaling modules.

Return Value

Returns a pointer to a new client instance

cspf_api_client_free

This function disconnects the session to the CSPF module and frees all associated memory. It should also be called when connection is reset by the server.

Files

lib/cspf_api.h

Arguments

Client (struct cspf_client *) - Pointer to cspf client instance

Return Value

void

cspf_api_msg_request_send

This function sends a route request message to the server. The signaling module is responsible for freeing any memory associated with the request structure.

Files

lib/cspf_api.h

Arguments

Request Message (struct cspf_api_message_route_request *) - Pointer to request message structure which needs to be sent to the server

CSPF Client (struct cspf_client *) - Pointer to CSPF client instance

Return Value

0 on success, -1 on failure

cspf_api_msg_established_send

This function sends a LSP established message to the server. This message should only be sent when an LSP is signalled by the signaling module and an explicit route was previously received for the corresponding LSP. The signaling module is responsible for freeing any memory associated with the message structure.

Files

lib/cspf_api.h

Arguments

LSP Established Message (struct cspf_api_message_lsp_established *) - Pointer to LSP established message structure which needs to be sent to the server

CSPF Client (struct cspf_client *) - Pointer to CSPF client instance

Return Value

0 on success, -1 on failure

cspf_api_msg_delete_send

This function sends a LSP delete message to the server. This message should be sent only for those LSPs for which an explicit route was received from CSPF. This message should be sent to CSPF whenever an LSP is being deleted by the signaling module. The signaling module is responsible for freeing any memory associated with the delete message structure.

Files

lib/cspf_api.h

Arguments

Delete Message (struct cspf_api_message_lsp_delete *) - Pointer to LSP delete message structure which needs to be sent to the server

CSPF Client (struct cspf_client *) - Pointer to CSPF client instance

©2012 IP Infusion Inc. Proprietary

Return Value

0 on success, -1 on failure

cspf_api_msg_notification_send

This message is sent by the signaling module to CSPF before terminating the session. CSPF also may send this message for exceptional conditions: session termination, route computation failure and so on.

Files

lib/cspf_api.h

Arguments

Notification Message (struct cspf_api_message_notification *) - Pointer to notification message structure which needs to be sent to the server

CSPF Client (struct cspf_client *) - Pointer to CSPF client instance

Return Value

0 on success, -1 on failure

CSPF Computation Algorithm

CSPF route computation algorithm is a modified form of SPF algorithm. The steps involved in the computation are :-

1. Verify that the egress address does not belong to ingress node and ingress address is a valid address on the ingress node.
2. Verify that egress and ingress belong to the same area.
3. Verify that the addresses specified in the path constraint belong to the same area as ingress and are reachable.
4. Verify that the hop limit constraint is not smaller than the number of addresses specified in the path constraint.
5. Perform SPF computation taking into account all the links which satisfy all the specified constraints.
6. If the computation fails, then recompute by adding back the bandwidth associated with LSPs with lower hold priority.
7. A tie between two equal cost paths is resolved in the following manner:
    1. Choose the path with lower hop count
    2. If hop count is same, then choose the path with according to the tie-break method specified. In case of "most fill", a path with lower minimum available bandwidth ratio is chosen. In case of "least fill", a path with higher minimum available bandwidth ratio is selected.
    3. If there is still a tie, then choose a path at random.

       Note: Available Bandwidth ratio = Available bandwidth/Max. Reservable bandwidth.

       Minimum Available Bandwidth Ratio = Least available bandwidth ratio of all the links forming the path.
8. In the case of path constraints, the computation is done in stages. In each stage, the start and end nodes for that stage are chosen from among the ingress, egress and path addresses. For example, in the first stage, ingress is

chosen as the start node and the first path address is chosen as the end node. In the second stage, the first path address is chosen as the start node and next path address is chosen as the end node. This continues until the actual egress is chosen as the end node or the intermediate computation fails.

©2012 IP Infusion Inc. Proprietary

CHAPTER 10 Graceful Restart

OSPFv2 Graceful Restart

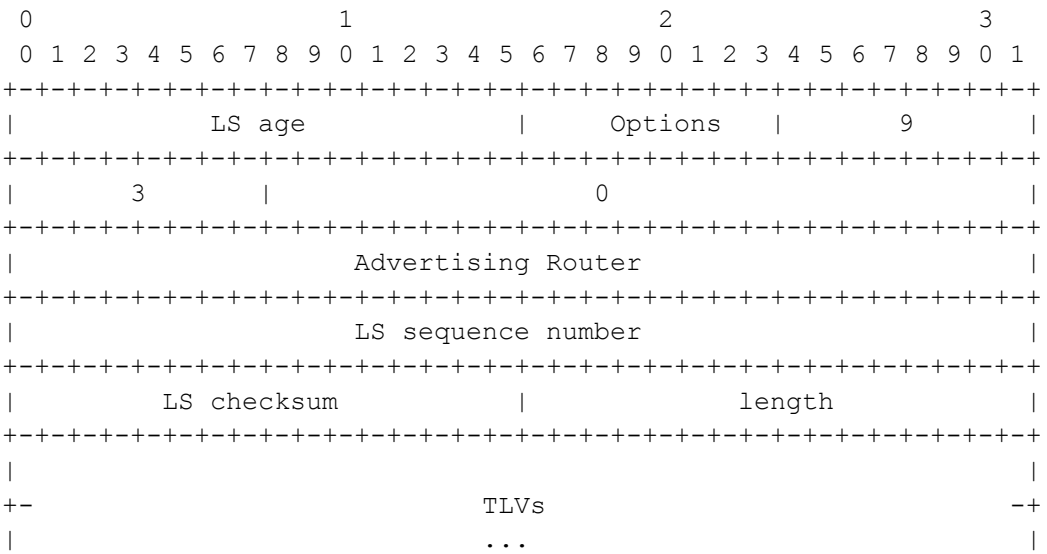
The ZebOS implementation of the OSPF graceful restart feature is based on `draft-ietf-ospf-hitless-restart-08`. Routers that separate control and management tasks from data-forwarding tasks are well-suited to the graceful restart feature. Network personnel initiate graceful restarts. Graceful restart is possible when the network topology is stable, and the restarting router retains its forwarding tables.

Normal Restart

Under normal conditions, OSPF routers automatically route around a restarting router. With graceful restart, a restarting router announces that it is entering a restart state with a Grace LSA. Neighboring routers continue to announce the restarting router as if it were still adjacent. When the router control is back online, the router is back online, with no time lost to reconstructing forwarding tables across the topology.

Graceful restarting is particularly suited to planned network outages. It might work in some unplanned outages, but generally the network has too little time to prepare and save the tables in a restartable state.

The layout of a Grace LSA contains several fields:



Restarting Mode

A router that sends out, at network personnel initiation, a Grace LSA is termed a restarting router. The interval of time from when the Grace LSA is sent until all the adjacencies are fully re-established is called the graceful restart. During graceful restart, the restarting router:

- does not originate any LSAs of type 1-5 or 7. The intent is for adjacent routers to calculate routes using the LSAs sent prior to the restart.
- does not install OSPF routes into the forwarding tables, relying on the forwarding entries extant prior to restart.
- retains its Designated Router (DR) status if it finds an entry in a Hello packet listing the restarting router as the DR.

Entering Restart Mode

A router enters restart mode when network personnel issue the `restart ospf graceful` command. The restarting router uses the `nsm_client_send_preserve_with_val` function to cause NSM to preserve the forwarding table. The restarting router also preserves, in non-volatile memory, the cryptographic sequence number using TLV fields: the `nsm_client_send_preserve_with_val` function accomplishes this.

If MD5 authentication is enabled, the cryptographic sequence number is included.

When the forwarding table and cryptographic data are preserved, Grace LSAs are sent, one for each of the OSPF interfaces. Graceful restart then proceeds with reloading/restarting the router.

Exiting Restart Mode

Graceful restart is exited when:

- All adjacencies are re-established.
- The restarting router receives an LSA that is not consistent with its pre-restart LSAs.
- The grace period expires.

At exit time, the router resumes normal routing duties, regardless of restart success. The `ospf_hitless_restart_exit` procedure accomplishes these actions:

- re-originates its LSAs for all attached areas.
- re-originates network LSAs for segments it is the DR
- recalculates the routes, installing results into the system forwarding table, originating summary LSAs, Type-7 LSAs and AS external LSAs, each as necessary.
- removes remnant entries from the system forwarding table
- flushes self-originated Grace LSAs

Helper Mode

A router that receives and accepts a Grace LSA is called a helper router. This router continues to advertise the restarting router. It sets the OSPF_NEIGHBOR_GRACEFUL flag in the `struct ospf.neighbor`. When this router exits from helper mode, it resets this flag.

Entering Helper Mode

Entrance into helper mode is contingent upon several conditions, not the least of which is local policy.

- Helper router has full adjacency with the restarting router.
- There are no changes to the link-state database since the initiation of the restart.
- The grace period (given in the Grace LSA) has not expired.
- Local policy is followed. (Examples of prohibitive policies are: never act as a helper; only at certain times of day; and only for certain routers.)

Note: Any router can act as a helper for multiple restarting routers. Grace periods can be updated if subsequent Grace LSAs are received.

©2012 IP Infusion Inc. Proprietary

Exiting Helper Mode

A helping router remains in helper mode until one of the following occurs:

- Restart is successful. When the Grace LSA is flushed, this is the signal that the restart was successful. The helper router calls `ospf_restart_helper_exit_by_lsa ()` (located in `ospf_restart.c`)
- The grace period expires. When the helping router receives a MaxAge Grace LSA, it terminates the helper mode calling `ospf_restart_helper_exit_timer ()` (located in `ospf_restart.c`).
- A change in the link-state database. When a helping router installs a new LSA in its database (Types 1-5, 7) with both:
  - the contents of the LSA changed (not a mere refresh)
  - the LSA would have been flooded to the restarting router if they would have been fully adjacent.

Note: If multiple helper routers are involved, any one of them can terminate the restart.

Source Code

The code for ZebOS OSPF graceful restart can be found in the `ospf_restart.c`, `ospf_restart.h`, and `ospf_nsm.c` files. The `ospf_nsm.c` file adds graceful-restart support to the NSM.

ospf_restart.h

This file contains the data definitions and structures for the graceful restart.

```
OSPF_GRACE_PERIOD_DEFAULT                    60
OSPF_GRACE_PERIOD_MIN                         1
OSPF_GRACE_PERIOD_MAX                      1800
OSPF_RESTART_STALE_TIMER_DELAY               40
OSPF_RESTART_STATE_TIMER_INTERVAL             3
OSPF_GRACE_LSA_TLV_DATA_SIZE_MAX             24
OSPF_GRACE_LSA_GRACE_PERIOD                   1
OSPF_GRACE_LSA_RESTART_REASON                 2
OSPF_GRACE_LSA_IP_INTERFACE_ADDRESS           3
OSPF_GRACE_LSA_GRACE_PERIOD_LEN               4
OSPF_GRACE_LSA_RESTART_REASON_LE              1
OSPF_GRACE_LSA_IP_INTERFACE_ADDRESS_LEN       4
```

ospf_restart.c

This file contains the procedural code for the graceful restart.

ospf_api.c

This file contains the functions that perform the CLI commands. See *API Calls* for details.

ospf_cli.c

Contains the `defun` statements for the CLI commands. See the *ZebOS OSPF Command Reference* for complete command details.

ospf_nsm.c

Contains support code for graceful restart bracketed by:

```
#ifdef HAVE_RESTART ... #endif HAVE_RESTART
```

ospf_nsm.h

This file contains function prototypes for the NSM support of graceful restart.

OSPFv3 Graceful Restart

OSPFv3 graceful restart lets a router gracefully restart the OSPFv3 control plane by maintaining the data-forwarding plane. The data-forwarding plane of a router can continue to process and forward packets, even if the control plane restarts.

After a router restarts/reloads, it changes its OSPFv3 processing, until it re-establishes full adjacencies with all of its former fully adjacent neighbors. Graceful restart is the time period between the restart/reload and re-establishment of adjacencies.

Features

ZebOS supports the OSPFv3 graceful restart features listed below.

- `restart ipv6 ospf graceful` CLI.
- CLI to enable helper policies on graceful restart helper.
- Preserving the restart TLVs like grace-period, restart reason, and link state IDs of router, network, external, inter-area, and link LSAs.
- Retrieving the restart TLVs from NSM.
- Framing and origination of the Grace LSA.
- Changes in OSPFv3 processing during the graceful restart period, as specified in RFC 3623 section 2.
- Actions by the restarting router before entering graceful restart, as described in RFC 3623 section 2.1.
- Exit conditions from graceful restart, as described in RFC 3623 section 2.2.
- Capability to revert back to normal OSPFv3 operation on exit from graceful restart, as described RFC 3623 section 2.3.
- Helper neighbor graceful restart.
- Actions by helper neighbor while entering and exiting the helper mode, as described in RFC 3623 section 3.

 ©2012 IP Infusion Inc. Proprietary

System Architecture

The following figure illustrates and describes graceful restart functionality between an OSPFv3 restarting router and NSM.

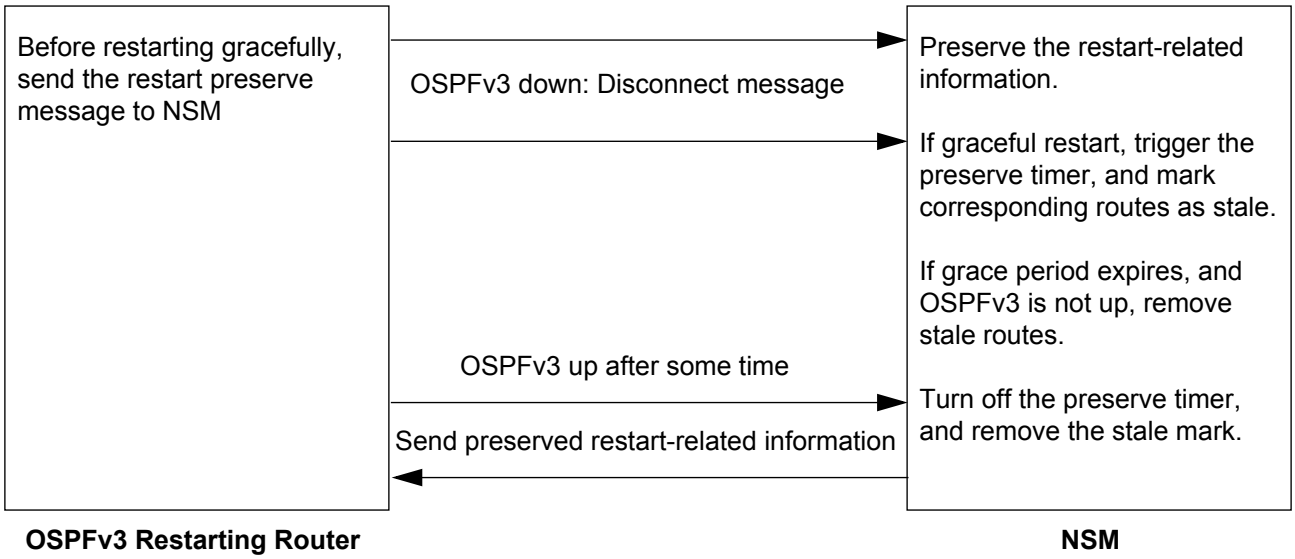


Figure 16: Graceful Restart Functionality

Whenever the `restart ipv6 ospf graceful` command is given, the restarting router ensures that its forwarding table(s) are updated before entering graceful restart mode. The ospf6d module shuts down after performing the following actions:

- Preserve a grace period and restart reason by sending a message to NSM.
- Notify NSM it is going to be restarted gracefully.
- Frame a Grace LSA by keeping the TLVs related to the grace period and restart reason. In this case, the Grace LSA is Type 0x000b.
- Wait for finite time (2 seconds) for acknowledgment of the Grace LSA. If not received, it retransmits the Grace LSA until acknowledged. The acknowledgement wait time is a static value (2 seconds).

When the daemon is up, it receives the preserved data from NSM, then schedules the restart timer for the remaining grace period.

While sending hellos, the graceful restarting state is monitored (completed or not): the adjacency state of the restarting router, with all of its neighbors, is checked.

- If the restarting router has reached full state adjacency with all of its neighbors, graceful restart mode is exited: successful restart.
- While in restarting mode, if an inconsistent router LSA has been received from any of its neighbors, graceful restart mode is immediately exited: unsuccessful restart.

If the grace period expires, restart mode is exited. Upon exiting,

- The router LSA for all areas is re-originated.
- If a DR, the network LSA is re-originated.
- The invalid self-originated Type-5 and Type-7 LSAs are flushed.
- The self-originated Type-5 and Type-7 LSAs are updated.
- The Grace LSA is flushed.
- The SPF calculation is re-run.

Helper Functionality

Whenever a restart-capable router receives a Grace LSA, helper policies are applied. The existing helper policies are:

- Never: Never act as restart helper
- Only-reload: Help only on software reloads
- Only-upgrade: Help only on software upgrades
- Max-grace-period: Help only if the received grace-period is less than this value

If the Grace LSA meets all specified helper policies, that router enters into helper mode. If the router is not restart capable, the received Unknown-type LSA is treated as a link-local-scoped LSA.

Exiting conditions from helper mode include:

- The Grace LSA originated by the restarting router on the segment is flushed.
- The grace period expired.
- A change in the link-state database contents indicates a network topology change, which forces graceful-restart termination.

Exiting actions include:

- Re-calculation of the DR for the segment.
- Re-origination of its router LSA for the segment's OSPF area.
- If the DR for the segment, it re-originates the network LSA for the segment.
- If the segment was a virtual link, its router LSA for the virtual link's transit area is re-originated.

©2012 IP Infusion Inc. Proprietary

Data Flow

The following illustrates the data flow of communication with NSM and neighbors.

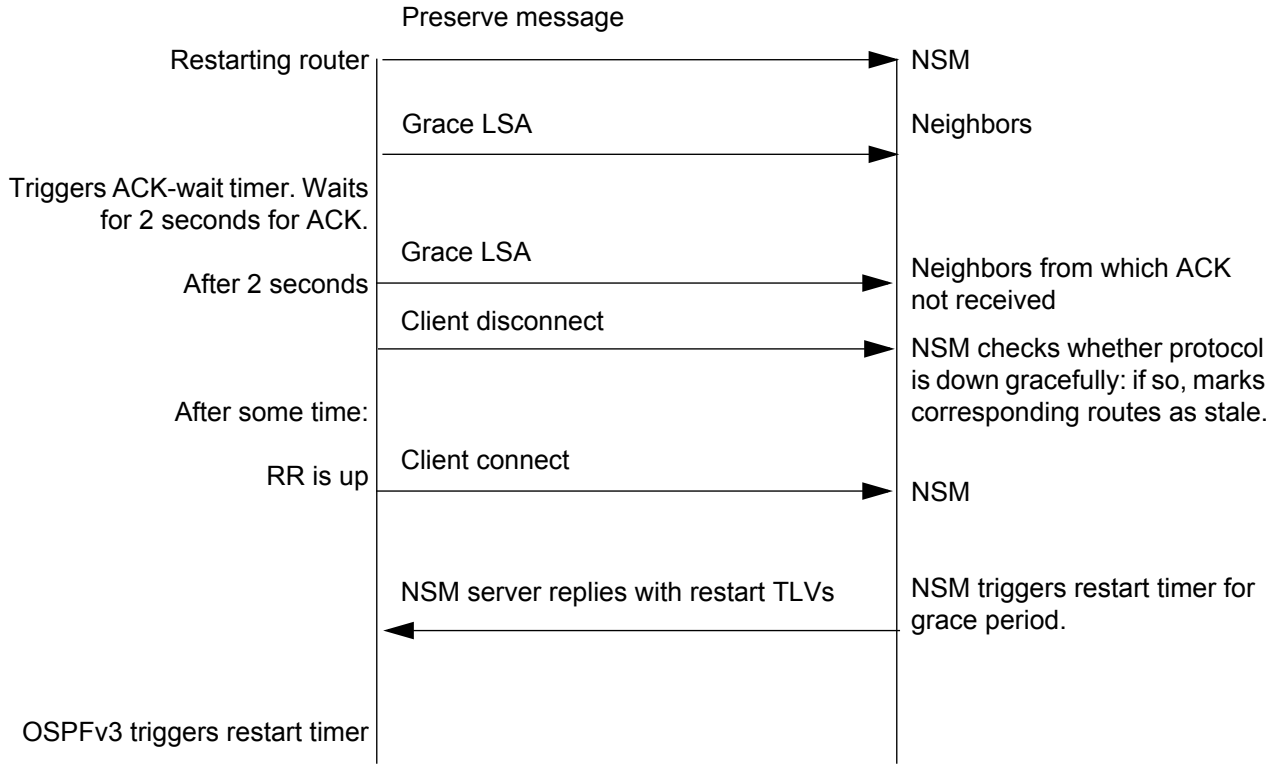


Figure 17: OSPF Graceful Restart Data Flow

Graceful Restart API Calls

The following lists graceful restart API calls. For information on these calls, refer to the *OSPF Developer Guide OSPFv2 Command Line Interface API Calls* and *OSPFv3 Command Line Interface API Calls* chapters.

OSPFv2

```
ospf_capability_restart_set
ospf_capability_restart_unset
ospf_graceful_restart_set
ospf_graceful_restart_unset
ospf_process_unset_hitless
ospf_restart_helper_grace_period_set
ospf_restart_helper_grace_period_unset
ospf_restart_helper_policy_set
ospf_restart_helper_policy_unset
```

OSPFv3

```
ospf6_capability_restart_set
ospf6_capability_restart_unset
ospf6_graceful_restart_set
ospf6_graceful_restart_unset
ospf6_restart_helper_grace_period_set
ospf6_restart_helper_grace_period_unset
ospf6_restart_helper_never_router_id_set
ospf6_restart_helper_never_router_id_unset
ospf6_restart_helper_never_router_unset_all
ospf6_restart_helper_policy_set
ospf6_restart_helper_policy_unset
ospf6_restart_helper_policy_unset_all
ospf6_router_unset_graceful
```


©2012 IP Infusion Inc. Proprietary

CHAPTER 11 Multi-Area Adjacency

Multi-area adjacency provides support for multiple OSPF areas on a single interface

Overview

This feature allows the link to be considered an intra-area link in multiple areas and be preferred over other higher-cost intra-area paths. This creates an intra-area path in each of the corresponding areas sharing the same link. For example, an interface can be configured to belong to multiple areas with a high-speed backbone link between two area border routers (ABRs) to allow creation of multi-area adjacencies that belong to different areas.

Features

ZebOS supports the multi-area adjacency features listed below.

- `multi-area-adjacency` CLI
- ABRs can establish multiple adjacencies belonging to different areas
- Multi-area-adjacency configuration and neighbor discovery as defined in section 2.1 of RFC 5185
- Packet transmission and reception as described in sections 2.2 and 2.3 of RFC 5185
- Configuration error if the link is configured as both virtual link and multi-area-adjacency
- Display of the multi-area-adjacency for multiple areas on the interface

System Overview

OSPF allows a single physical link to be shared by multiple areas to create an intra-area path in each of the corresponding areas sharing the same link.

The following provides a sample configuration and information on multi-area adjacency configuration.

Configuration Example

The following example uses a high-speed link between two ABRs in multiple areas.

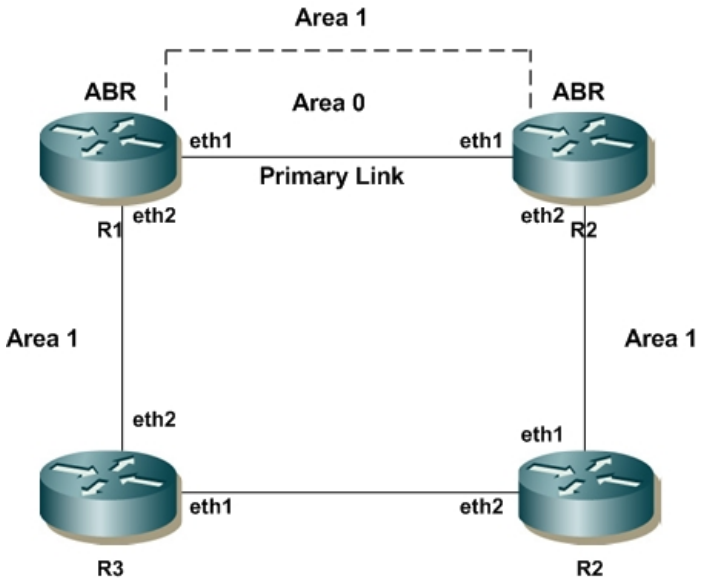


Figure 18: Multi-area Adjacency

The backbone-area link between R1 and R2 is a high-speed link. It is desirable to forward Area 1's traffic between R1 and R2 over the high-speed link. Previously, intra-area paths were preferred over inter-area paths. As a result, R1 would always route traffic to R4 through Area 1 over the lower-speed links. R1 would even use the intra-area Area 1 path though R3 to get to Area 1 networks connected to R2.

To reach from R1 to R2 on a direct link, the link is also configured in Area 1. With multi-area adjacency, the existing link is allowed for Area 0 and Area 1. This Area 1 link is configured as multi-area-adjacency and becomes an intra-area path for R1 to directly reach R2. In this way, R1 can directly route the Area 1 traffic from R1 to R2 on the Area 1 link.

Configuration Techniques

Multi-area adjacency establishes adjacency between the ABRs. A specific interface of the ABR is associated with multiple areas.

Each multi-area adjacency is announced as a point-to-point link in the configured area. The point-to-point link provides a topological path for that area. Because multi-area adjacency is based on the primary adjacency, the primary adjacency should be up to establish multi-area adjacency.

Multi-area adjacencies are configured between two routers with a common interface. The neighbor address of each multi-area adjacency must be configured.

There is no restriction on multi-area adjacency for the type of areas on which it can be configured. Multi-area adjacency can be configured in backbone and non-backbone areas. However, if a virtual link is configured between two routers

©2012 IP Infusion Inc. Proprietary

through a transit area, a multi-area-adjacency link cannot be configured in a backbone area for a link between these routers in a transit area. Similarly, if a multi-adjacency link is configured between a pair of routers in a backbone area using a primary link in a non-backbone area, a virtual link can not be established between these routers for a transit area the same as the area ID of the non-backbone area of the primary link used for the multi-adjacency link.

System Architecture

OSPF Interface Structure for Multi-Area Adjacency

Previously, ZebOS supported only one OSPF interface creation for each interface address.

Multi-area adjacency allows the OSPF interface creation for multiple areas for each interface address. The Interface State Machine (IFSM) is maintained on the respective OSPF interface.

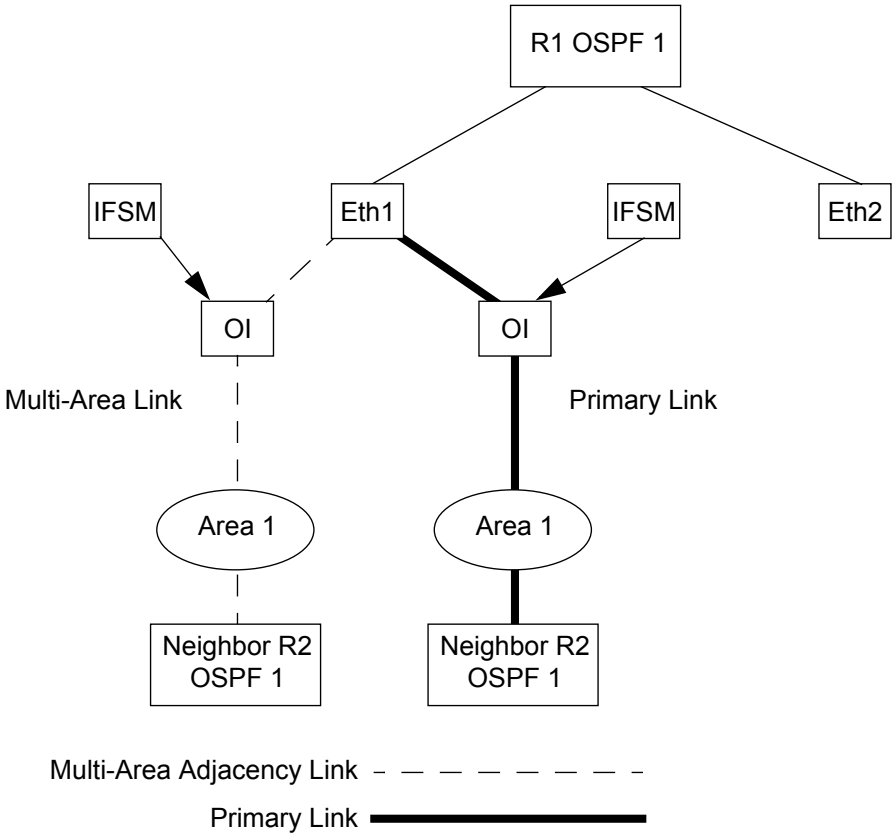


Figure 19: Multi-Area AdjacencyInterface Structure

In the diagram above, R1 and R2 are ABRs (border of Area 0 and Area 1) running an OSPF instance between them. There is a backbone area high-speed link between R1 and R2. This link is also configured as a multi-area link in Area 1. A separate OSPF interface data structure (OI) is maintained for both the primary and multi-area adjacent links. An IFSM is maintained separately for each OI.

Primary and Multi-Area Adjacent OSPF Interface

Packets associated with multi-area adjacency use the primary link's area as the transit area. Therefore, if the primary interface goes down, the multi-area adjacency corresponding interfaces using the primary interface also go down. Also, if any properties associated with the primary interface change, the same properties are applied to all multi-area adjacency interfaces associated with the primary interface.

If the primary adjacency is disabled (using the `no network` command), the multi-area adjacency is also deleted.

Functions

The following lists functions added or modified in the OSPFv2 module to accommodate multi-area adjacency.

New Functions

- `ospf_multi_area_link_entry_insert` in `ospf_multi_area_link.c` inserts the multi-area link in the multi-area interface table.
- `ospf_multi_area_link_entry_delete` in `ospf_multi_area_link.c` deletes the multi-area link from the multi-area interface table.
- `ospf_multi_area_link_entry_lookup` in `ospf_multi_area_link.c` lookup for the existing multi-area adjacency interface.
- `ospf_multi_area_link_if_new` in `ospf_multi_area_link.c` gets the interface for a multi-area adjacency.
- `ospf_multi_area_link_free` in `ospf_multi_area_link.c` frees the multi-area link.
- `ospf_multi_area_if_new` in `ospf_multi_area_link.c` creates a new interface structure for the multi-area link.
- `ospf_multi_area_link_get` in `ospf_multi_area_link.c` gets the multi-area link.
- `ospf_multi_area_if_entry_delete` in `ospf_multi_area_link.c` deletes the interface from the multi-area link.
- `ospf_multi_area_if_entries_del_corr_to_primary_adj` in `ospf_multi_area_link.c` deletes the multi-area adjacencies corresponding to the primary adjacency.
- `ospf_multi_area_link_delete` in `ospf_multi_area_link.c` frees the multi-area link.
- `ospf_multi_area_link_delete_all` in `ospf_multi_area_link.c` deletes the multi-area adjacencies to all neighbors on the interface with the given area ID.
- `ospf_multi_area_link_up_corr_to_primary` in `ospf_multi_area_link.c` multi-area link up corresponding to the primary link.
- `ospf_multi_area_if_up` in `ospf_multi_area_link.c` enables the OSPF multi-area link.
- `ospf_cli_show_multi_area_adjacencies` in `ospf_show.c` displays multi-area adjacency information.

©2012 IP Infusion Inc. Proprietary

Modified Functions

- `ospf_if_new` in `ospf_interface.c` adds a check for a multi-area adjacency. If the area is configured for multi-area adjacency, the interface is added to the multi-area interface table.
- `ospf_if_init` in `ospf_interface.c` adds a check for a multi-area adjacency, based on the area ID: If configured for multi-area adjacency, the interface type is configured as point-to-point, and the details of the primary interface type and primary interface network type are added for multi-area adjacency.
- `ospf_if_delete` in `ospf_interface.c` adds a check for an interface whether or not multi-area adjacency is enabled. If the interface is multi-area-adjacency enabled, it is removed from the multi-area interface table.
- `ospf_network_apply` in `ospfd.c` creates the interface for a multi-area.
- `ospf_packet_proc_clone` in `ospf_packet.c` gets the OSPF interface that corresponds to the multi-area link and checks if the received packet is for the multi-area adjacency.
- `ospf_packet_netmask_check` in `ospf_packet.c` adds a check for comparing the packet's source address with the interface IP address, if the packet is received for the multi-area.
- `ospf_hello_send` in `ospf_packet.c` sends packets to the neighbor address for the multi-area adjacency, if the common interface type or network type is other than point-to-point.
- `ospf_ls_req_read` in `ospf_packet.c` gets the destination address, based on the neighbor, if the interface is associated with multi-area.
- `ospf_hello` in `ospf_packet.c` cancels the thread on the static neighbor.
- `ospf_ls_ack_send_delayed` in `ospf_packet.c` sends the delayed LSAs, based on the neighbor, if the interface is associated with multi-area.
- `ospf_router_lsa_ptop_set` in `ospf_lsa.c` adds a check for multi-area adjacency: If multi-area adjacency, the Type-3 link is not advertised in the router LSA. Also, the neighbor address/interface index is added to the router LSA for multi-area adjacency networks, if the underlying interface is unnumbered.
- `ospf_network_withdraw` in `ospfd.c` removes entries from the multi-area interface table before removing the primary adjacency.
- `ospf_proc_init` in `ospfd.c` is the initialization for the multi-area interface table and multi-area networks.
- `ospf_proc_lsdb_clear_all` in `ospfd.c` removes all LS database entries corresponding to multi-area interface.
- `ospf_proc_down` in `ospfd.c` takes down all OSPF interfaces corresponding to multi-area.
- `ospf_proc_clear` in `ospfd.c` clears all networks and interface tables associated with multi-area.
- `ospf_process_unset_graceful` in `ospf_api.c` send grace LSAs for multi-area networks, also.
- `ospf_nbr_static_config_check` in `ospf_api.c` adds a check for the neighbor configuration state for multi-area networks.
- `ospf_ls_retransmit_delete_nbr_as` in `ospf_flood.c` removes all neighbor/interface's Link State Retransmit list for multi-area, also.
- `ospf_flood_self_lsa_check` in `ospf_flood.c` adds a check for self-generated LSAs for multi-area.
- `ospf_proc_lsdb_count` in `ospf_lsdb.c` counts the LS database entries in multi-area interfaces, also.
- `ospf_grace_lsa_recv_check` in `ospf_restart.c` keeps track of grace LSA ACKs for multi-area adjacencies.
- `ospf_grace_ack_wait_timer` in `ospf_restart.c` adds support for multi-area adjacency. If the grace LSA is not received till the expiration of the wait timer, the LSA is flooded again.
- `ospf_cli_show_neighbor` in `ospf_show.c` supports display of multi-area adjacency neighbors.
- `ospf_cli_show_neighbor_all` in `ospf_show.c` supports display of all multi-area adjacency neighbors.

- `ospf_cli_show_neighbor_by_nbr_id` in `ospf_show.c` supports display of multi-area adjacency neighbor by neighbor ID.
- `ospf_cli_show_neighbor_detail_all` in `ospf_show.c` supports display of details of all multi-area adjacency.
- `ospf_cli_show_neighbor_detail` in `ospf_show.c` supports display of details of multi-area neighbors.
- `ospf_cli_show_neighbor` in `ospf_show.c` adds a check to determine whether the multi-area-adjacency flag is set before showing the neighbor. If set, it shows whether or not it is a multi-area-adjacency neighbor.
- `ospf_cli_return` in `ospf_cli.c` defines two new macros: 1) For the `OSPF_API_SET_ERR_MULTI_AREA_LINK_CANT_GET` macro, the `unable to create the multi area link` string is assigned. 2) For the `OSPF_API_SET_ERR_MULTI_AREA_ADJ_NOT_SET` macro, the `Multi area adjacency is not assigned for the particular area on interface` string is assigned.
- `ospf_vlink_set` in `ospf_api.c` adds a check to determine whether the area is configured for multi-area adjacency: if configured, it returns error.

©2012 IP Infusion Inc. Proprietary

Appendix A CSPF Message Values

Message Types

Message Type	Value
Route Request Message	0x0001
Route Message	0x0002
LSP Established Message	0x0003
LSP Delete message	0x0004
Notification Message	0x0005

TLV Types

TLV Type	Value
Re-Optimize TLV	0x0100
Retry TLV	0x0101
Hop Limit TLV	0x0102
Adm. Control TLV	0x0103
Priority TLV	0x0104
Path TLV	0x0105
ERO TLV	0x0106
LSP TLV	0x0107
Status TLV	0x0108
TE Class TLV	0x0109
Exclude Path TLV	0x0110
Extended Status TLV	0x0111

Protocol Types

Protocol Type	Value
RSVP-TE	0x00
CR-LDP	0x01

Status Codes

Status Code	Value
CSPF_CODE_SHUTDOWN	0x4000

CSPF_CODE_ROUTE_NOT_FOUND	0x4001
CSPF_CODE_ROUTE_FAIL	0x4002

©2012 IP Infusion Inc. Proprietary

Index

©2012 IP Infusion Inc. Proprietary