

INTERNET-DRAFT
Intended Status: Informational draft
Expires: April 5, 2015

Arunkumar Arumuga Nainar
Tata Communications Ltd
October 2, 2014

Layer 4 Path preference negotiation for DPS
draft-arumuganainar-rtgwg-dps-l4-ppn-01

Abstract

This document is a supporting draft to draft-arumuganainar-rtgwg-DPS-Requirements-01. The DPS draft talks about high level architecture to implement dynamic path selection based on application. The DPS draft suggests the implementation to be done in three steps:

- 1) DPS Signaling: Here participating routers communicate with each other to exchange application related information
- 2) Profile Based Filter: This section describes how packets can be classified and filtered
- 3) DPS Routing Frame Work: This ensures that separated traffic remains separated through out the network

While overall architecture is still valid, this draft suggests an enhancement to the DPS Signaling component. The currently implemented technique uses BGP for the signaling requirements. Whilst this is good for certain cases, applications that can be off loaded to the secondary link are pre-decided. It restricts behavior from responding to dynamic network conditions. For example, a network administrator would want to off load some of the non-critical applications over the secondary link, however when there is acute congestion within the network, they might want the router to behave aggressively by off loading more applications to the secondary circuit. Yet while doing so there should not be any asymmetric routing on the network.

Since BGP is essentially a control plane protocol, it is not aware of what is happening on the network in the forwarding plane, hence there is a need to do the signaling in the forwarding plane. This drafts suggest one such mechanism. Here the idea is to exchange the Path Preference information at layer 4 level. Such signaling could happen during TCP connection establishment phase. When done this way, a decision can be taken for each of the session and hence making it more dynamic than the one that can achieved through BGP.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the

provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>

Copyright and License Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1	Introduction	4
1.1	Terminology	4
2.	DPS Layer 4 Signaling Overview.	4
3.	Application Classification :-	5
4.	Application List And Grey Scales	6
5.	Layer 7 Classification Issue:-	7
6.	Customization Of Path Selection	8
8.	Implementation Details.	9
7.	Summary	9
8	Security Considerations	11

9	IANA Considerations	11
10	References	11
10.1	Normative References	11
10.2	Informative References	11
	Authors' Addresses	11

1 Introduction

BGP is used for signaling in the current implementation of DPS . BGP provides a consistent mechanism to exchange the Path Preference information between the two routers/sites. However, since BGP is a control plane protocol, we will not be able to exchange the dynamically changing network conditions. Here there is need to do the signaling in the forwarding plane.

This draft proposes to do the signaling during the TCP connection establishment phase. When done this way the signaling is done for each session. This enables the router to be aware of network conditions dynamically and take the most appropriate path.

1.1 Terminology

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

2. DPS Layer 4 Signaling Overview.

The TCP connection establishment follows the three way process. This can be summarised in 3 steps:

- 1) Client M/C first sends a SYN packet
- 2) Sever M/C accepts the SYN and responds back with a SYN/ACK
- 3) Client Acknowledges the SYN/ACK with an ACK

During the whole process, many parameters will be exchanged and agreed upon. Some router implementations intercept these session initiation packets and influence the exchanged information. For example, routers could intercept the SYN packet and force the end hosts to negotiate a reduced MSS size.

Similar legal intercept technique could be used to exchange Path Preference information.

Client -----Router1 ====(Network)=====Router2-----Server

Here the interception works in the below fashion:

- 1) Client sends a packet to server

2) Router 1 intercepts the SYN packet and adds TCP options. This option could be loaded with Path Preference information. For example, let us assume there is no congestion on the client site's tail circuit. Hence, router 1 wants to send traffic that belongs to this session over the primary circuit.

3) When router 2 receives the SYN packet with that known TCP option. It comes to know that the SYN is coming from a DPS capable site. It then caches the TCP state, strips the option and sends it to the server.

4) When the server responds back with a SYN/ACK, router 2 intercepts the SYN/ACK and adds its Path Preference information in to the SYN/ACK packet using the same TCP option. Let's assume there is a congestion on the network at router 2, it replies back saying my side is congested so let us off load this session to the secondary link.

5) Router 1, when it receives back the SYN/ACK it knows it is coming from a DPS capable site and the remote site wants the application to be off loaded to the secondary link.

6) When the client sends back ACK, router 1 intercepts it and adds an option confirming that it agrees to the remote site choice to offload that session

Subsequently when the data packets are sent over a path that is negotiated and agreed. Here the preference of a particular path can be taken based on dynamic local network conditions. Since the path is negotiated and agreed across both ends, the network will respond to both local and remote network conditions.

3.Application Classification :-

The key thing here is how does a router decide which application needs to be off loaded. This draft makes the following recommendations.

To begin with, the administrator of the network should prepare 3 lists: a) white list b) grey list c) black list

White List:- Application listed in the white list always goes over the primary link. The behavior does not change even if there is any congestion on the link.

Black List:- Application listed in the black list always goes over secondary link. Even when there is no congestion on the primary link.

Grey List:- Application listed in the grey list behaves like a white list application when the primary link is not congested and behaves like a black list application when there is congestion.

It should be noted that the grey list behavior can also be a result of remote side congestion. For example, when given Site prefers the primary link for the grey list application but the remote site prefers the back up link, then both sites will agree on back path for the session.

Also the definition of application could also include a layer 7 signature. For example:

- 1) HTTP (TCP Port: 80) made to URL www.youtube.com. Could be put on a black list
- 2) HTTP (TCP Port: 80) made to URL www.gmail.com. Could be put on a grey list

Hence it is mandatory that router implementations should support deep packet inspection for the purpose of classification (based on layer 7 signature)

4. Application List And Grey Scales

While the 3 list hierarchy does seems to work. Under practical circumstances this is not sufficient. Network administrators will want to off load applications less aggressively when the utilization levels are very low. However, they may want to progressively off load more applications or TCP sessions. To illustrate this, the following example is given:

Let us say following applications are listed as grey applications:

Grey Shade 1:- Email, Intranet

Grey Shade 2:- Video streaming application such as You Tube

Grey shade 3:- All peer to peer applications.

1) When link utilization is ≤ 60 all grey applications will go over the primary link

2) When link utilisation is > 60 but ≤ 70 , shade 3 applications(peer-to-peer) will go over the secondary link

3) When link link utilisation is >70 but <=90, shade 3 and shade 2 applications (peer-to-peer & video streaming) will go over the secondary

4) When link utilisation is >90, all grey applications will be off loaded

5. Layer 7 Classification Issue:-

In order to make the off loading very effective, we should be able to do two things.

1) Classification and grouping of protocol to be done based on L7 signatures

2) Path Preferences should be negotiated per session. This actually provides the ability to react to local or remote network conditions

Whilst negotiating the Path Preferences during the connection establishment, there will be problem reading the layer 7 Signatures during that phase. Layer 7 signature will be only know after the connection is established. Hence if the grey list is based on layer 7 signatures, then direct negotiation of Path Preferences will not be possible. Hence an alternate approach needs to be worked out.

To overcome the above problem, the draft suggests, rather than negotiating Path Preferences directly, we could get pre-authorization for the use of specific path. This will work based on the following rules.

1) When the SYN packet hits router, it intercepts that packet and adds a TCP option. This option will communicate what is the maximum grey shade that router can send via the primary link. For example, if you have 6 shades of grey defined and if the TCP option indicates option 4, then it indicates router is seeking pre-authorization for offloading applications that belongs to shade 5 or 6.

2) However it could happen that in the remote site primary link is heavily congested and hence it would want to offload all applications that belong to shades 3, 4, 5 & 6. Hence it's router responds back with TCP option set to 2.

3) Since 2 is less than 4, Originating site-router accepts Remote site router's choice and confirms it agreement by sending back the acceptance in the ACK message.

When the connection is established routers in both site would have formally agreed what path to choose when they determine the layer 7 signatures. Because of this pre-authorization, there is no chance of asymmetric routing. In this scheme of things following rules apply:

- 1) Administrator should clearly indicate what is LAN facing interface and what is WAN facing interface on the router.
- 2) Any SYN packet with no Path Preference information set would indicate the packet is coming from a site that does not have DPS enabled. In such a case normal routing will be followed to forward the packet.
- 3) If the two routers specify different Path Preference information, the lowest number always wins.

Additionally it is based on the assumption that classification of application and grey scale definition will have to be done globally. This is because only grey scale levels are negotiated. The premise of the negotiation is based on the fact that once the grey scale is negotiated, the path of application could be determined with certainty. This is only possible when the definitions are made globally. Hence the draft recommends any implementations should support central definition of an application list. And any individual routers/devices participating in DPS should be able to pull the definition from central location so as to avoid inconsistencies in path selection.

UDP and other non-TCP based applications may not be able to participate in the negotiation. Hence they could be either placed in a white list or a black list.

6. Customization Of Path Selection

Though the definition of the white list, grey list and black list is universal across the network, customization is still possible. For example a smaller site with a smaller primary circuit can be configured to support only two shades of grey. While a larger site that has large bandwidths can be configured to support 5 shades of grey. Also grey shade selection could also be individually configured. For illustration following example is provided.

A given network 3 type of site.

Site type A: Primary circuit : 100 MBPS , Secondary : 100 MBPS

Site Type B: Primary circuit : 10 MBPS , Secondary : 10 MBPS

Site Type C: Primary circuit : 2 MBPS , Secondary : 2 MBPS

Site type A and B can support all the 5 shades of grey. And site type C can support only 2 shades of grey. Further site, type A starts off loading traffic when the utilization level goes above 80% while site type B does it when it crosses 60%.

Even if we tweak this per site, the chance of asymmetric routing does not arise because two sites that are communicating will always negotiate the same grey levels to off load.

8. Implementation Details.

Currently many non-routing devices also perform deep packet inspection. For example, WAN optimization appliances. The signaling portion can also be off loaded to devices behind routers. In such a case DPS will be implemented on the router based on the traditional model. However the router will be configured to trust the marking done on the sites that hosts these WAN optimization devices. i.e Coloring will be done on an external device that is capable of doing deep packet inspection.

7. Summary

Currently in the DPS implementation, signaling is done in the signaling plane via BGP. Because the signaling happens in the control plane, we are limited to define applications statically. It does not provide room for off loading applications based on dynamically changing network condition.

By moving the signaling to forwarding plane, the draw back of the previous method of signaling can be done away with.

This draft suggests doing the signaling when the TCP connection is established. This draft also acknowledges difficulty to integrate layer 7 signatures based classification. To overcome this, the implementation should support global definition application lists. The two sites, whilst negotiating the Path Preferences, only negotiates the highest shade of grey that can sent over the primary circuit.

In summary, it is possible to do the signaling of path preference information at layer 4. This would enable DPS implementation to alter the DPS behavior based on network level changes that are local or remote to the given site.

```
Definitions and code {  
    line 1  
    line 2  
}
```

Special characters examples:

The characters , , ,

However, the characters \0, \&, \%, \" are displayed.

.ti 0 is displayed in text instead of used as a directive.

.\" is displayed in document instead of being treated as a comment

C:\dir\subdir\file.ext Shows inclusion of backslash \".

8 Security Considerations

TBD

9 IANA Considerations

TBD

10 References

10.1 Normative References

10.2 Informative References

11 Acknowledgements

The authors would like to thank Hesham Moussa for his review and comments.

Authors' Addresses

Arunkumar Arumuga Nainar
Tata Communications (UK)
1st Floor
20 Old Bailey
London EC4M 7AN
United Kingdom

Email: arun.arumuganainar@tatacommunications.com

INTERNET-DRAFT
Intended Status: Informational draft
Expires: April 5, 2014

Arunkumar Arumuga Nainar
Tata Communications Ltd
October 2, 2014

Dynamic Path Selection Requirements
draft-arumuganainar-rtgwg-dps-requirements-01

Abstract

The high availability puzzle can be resolved by building in resiliency to network designs. Whilst active/backup routing schemes are sufficient to create redundancy with low convergence times the following deficiencies and customer demands are not addressed comprehensively.

1. IP routing is essentially best path based. This will lead to underutilized or over utilized links.
2. WAN application performance could be adversely impacted due to congestion whilst the backup link remains idle. Techniques such as DiffServ QoS do address the problem effectively, but those approaches address only the symptoms and not the root cause.
3. Half of the network resources that the end customer has paid for, always remains unused. This is a matter of huge concern for small and mid-size customers as WAN circuit costs are very high and recurring.

Hence there is genuine requirement to offload some of the traffic to an alternate path while the primary path is still active and running. This document defines this requirement in detail.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at

<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

Copyright and License Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1	Introduction	3
1.1	Terminology	3
2.	Basic assumption	3
3.	Problem statement :-	5
4.	DPS Routing Requirements	5
5.	Dynamic Path selection Framework	7
6.	Packet flow in DPS Environment	10
8.	Implementation Details and Gap Analysis.	10
8.1	DPS Routing domain:-	10
8.2	DPS Signaling:-	12
8.3	Profile based Filter	13
7.	Summary	13
8	Security Considerations	15
9	IANA Considerations	15
10	References	15
10.1	Normative References	15
10.2	Informative References	15
	Authors' Addresses	15

1 Introduction

In a large enterprise environment, network locations are often distributed over large geographical areas (often it spans different countries and continents). In such a case enterprise network managers often buy virtual private networks from service providers. Such a network is built over either MPLS or Internet/IPSEC extension to MPLS networks.

Generally the enterprise end user location connects in to the service provider using last mile connections. These last mile circuits comes in various technology options and bandwidth capacities. While the service provider core is very resilient and virtually has infinite capacity in terms of bandwidth, the weak link in the enterprise network is indeed the last mile.

Last miles are the costliest component for the enterprise and most of the times it contributes to 50-60% of total network costs. Yet these are the weakest link in the network. They constitute single point of failures and do fail frequently. Hence in order to increase the network availability the network managers buy redundant links.

With redundant links in place, the availability puzzle is fully resolved, however, it does pose a significant question. Half of the costliest resource that they have purchased will remain idle throughout the life of this network. This happens while network managers work over time to resolve the issues that arise out of congestion on the primary circuit.

There is a genuine need to identify few select application and off load them on to a path that is considered to be the best path by the routing protocols. This draft defines and documents this requirement. This draft also describes the high level framework based on which such application off loading could be accomplished. The scope of draft also includes the evaluation of the existing techniques and their gap analysis.

1.1 Terminology

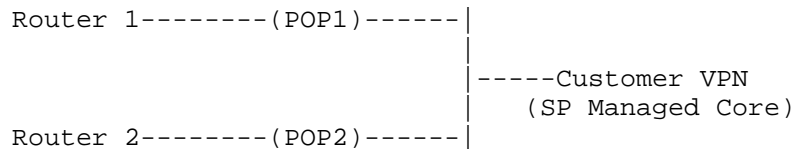
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Basic assumption

This draft addresses a problem faced in an enterprise network. Typical enterprises comprise of collection of sites/Local Area

Networks (LANs) that are spread across a large geographical area (across cities, countries and continents). These sites are interconnected via a virtual private network of some form. Such enterprises typically go to single or even multiple service providers to establish this connectivity.

In such a case, the service provider will interconnect the site at their POP location via a last mile circuit. The choice of last mile technology could be diverse such as Ethernet, Frame-Relay, ATM, DSL, T1/E1/T3/E3, etc. Bandwidth capacity also varies from location to location (as low as 512 Kbps DSL to as high as 10 Gbps Ethernet). In cases where VPN connectivity is not possible, the VPN network can be extended in to a remote location via Internet/IPSEC technologies as well. Due to the diverse nature of the connectivity option, any framework should be agonistic to technologies and choice of providers.



This draft assumes availability is very important for the target enterprises, hence there will be at least two last mile circuits that link them to the service provider core. Again, technology and the last mile provider can be chosen independently. Often enterprises selects a good or premium primary circuit and cheaper option for the secondary connectivity. For example

- 1) Primary: 10 Mbps Ethernet to MPLS; Secondary:- 10 Mbps Ethernet to MPLS
- 2) Primary: 10 Mbps Ethernet to MPLS; secondary:- 2 MBPS E1 circuit to MPLS
- 3) Primary: 1.5 Mbps T1; Secondary:- ADSL 2+ Internet/IPSEC

The above samples are for illustration taken from typical examples found in the field.

Routing schemes usually designate the premium circuit option as primary. Routing protocol metrics will hence be better for the link and all the traffic goes over it as long as they are active. Traffic shift to the secondary if the primary circuit goes down. This kind of routing scheme is called classical routing or Active-passive routing scheme.

3. Problem statement :-

In the classical routing scheme, all traffic goes over the primary circuit. Sizing of the circuit will depend on the bandwidth requirements of all the applications running on the LAN side which need remote connectivity. Last mile circuits generally constitute a large bulk of the wan network cost. In some networks that are geographically well spread this could constitute 50-60% of total WAN network costs, hence over sizing the network is not an option.

As an alternative, the Network Administrator could resort to Quality of Service. This will ensure applications that are critical to the business get enough bandwidth, but as collateral damage, applications that are not so critical could be starved of the bandwidth. This starvation could happen even while secondary circuit is active and contains lots of unused bandwidth.

Key challenge here is to find a way to select certain applications that are considered not-so critical for the business and off load them on to secondary circuit. This would ensure applications that would otherwise be starved of bandwidth in the primary circuit will have lots of room in the secondary circuit. This kind of active-active routing scheme will hence forth be called as DPS (Dynamic Path Selection) routing. This when achieved, will improve the overall user experience for the not-so-critical applications without compromising the critical applications. In majority of the situations this will result in a slightly better experience for the critical applications.

It should be noted that during the event of a primary circuit failure, both critical and non-critical applications will go over the secondary circuit. Hence the QoS mechanism will have to be applied even on the secondary circuit so as to punish the not-so-critical applications. Though this is a real problem, the occurrence of such an event is very rare. SLA for a premium circuit such as Carrier or Metro Ethernet could be as high as 99.5% . With this SLA in place network administrators can guarantee acceptable level of performance for all applications 99.5% of the time whilst for critical applications the experience can be guaranteed 99.999% of times.

If this DPS routing can be achieved, then it will directly translate in to greater user experience for users with out compromising the cost that the Network managers will have to spend on the WAN networks.

4. DPS Routing Requirements

The DPS requirements can be summarized in 5 simple steps.

1) Any two sites participating in the DPS routing, will have to agree on off loading patterns. This pattern is referred to as DPS Profiles. Example of a profile is illustrated below.

Profile 1: Critical Application: SAP, Citrix, VOIP ,Telnet; Non-Critical: remaining applications

Profile 2: Critical Application: SAP, Citrix, VOIP, Telnet, SMTP, HTTP; Non-Critical: remaining applications

Profile 3: Critical Application: All non-Internet traffic; Non-Critical: Internet-traffic

If three sites wanted to participate in DPS routing, they need to exchange the profile information and agree on a common profile to off load. It should be possible that the first site will negotiate to use profile 1 for the second site, and profile 2 with the third. The underling requirement is if two sites communicate they should be able to agree on a common profile.

2) Traffic hitting the input interface of router will have to be first classified based on application. Here the application can be defined as follows:

- a) A specific known TCP port number
- b) Combination of TCP Port and Server's IP Address
- c) A known Layer 7 Signature

For example:

- a) Traffic sent to TCP Port (80) could be classified as HTTP application.
- b) Traffic sent to TCP Port 80 and Server IP (ex: proxy server) can be classified as Internet
- c) Traffic sent to TCP Port 80 and Server IP (ex. proxy server) sent to the URL www.example.com can be classified as "EXAMPLE.COM APP". Here the classification was based on the Layer 7 Signature.

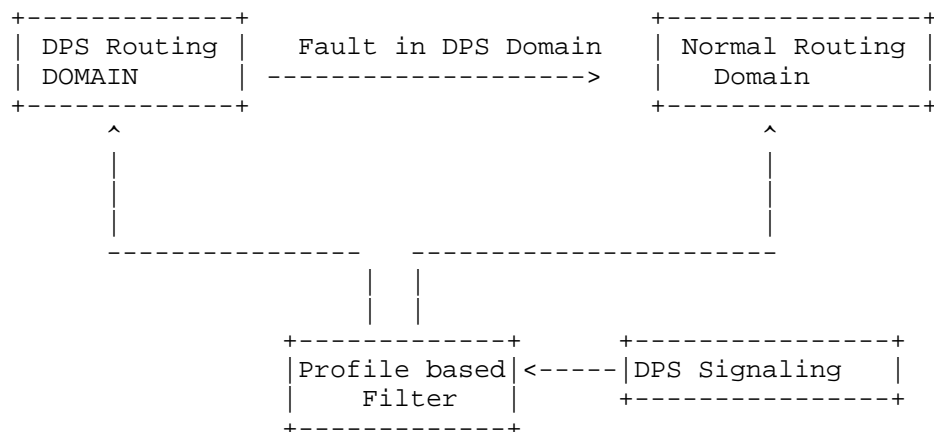
3) If the profile is agreed and classification is done, then the participating router should be able to filter the traffic and send it

across available paths.

4) During step 2, if the traffic is pushed to the secondary circuit (not the best path as per the routing protocol) it should stick with the circuit end to end. For example an application that has been off loaded will have to enter the remote site via the secondary circuit. Return path for that flow will have to take the secondary path at both locations so as to prevent asymmetric routing.

As the quality of the link does vary largely at any given site asymmetric routing will invariably create a application performance issues and hence should be avoided at all times. Hence, path symmetry should be honored even during positive and negative events on the network. For example if the secondary circuit fails, either locally or at the remote site, it is treated as a negative event and during this time path symmetry should not be compromised.

5. Dynamic Path selection Framework



The objective of DPS is to achieve end-to-end application separation with out introducing asymmetric routing within the network. In order to ensure the above objectives, we should have a comprehensive mechanism to achieve the following tasks.

Task 1: Any two sites participating in DPS will have to agree on a common set of applications that will be off loaded to secondary path (also referred to as DPS path on this draft).

Task 2: At the time of forwarding the packets, packet should be filtered based on agreed application set. Packets should then be pushed in to appropriate paths.

Task 3: If the packet is pushed on to a DPS path, it should always use the secondary link end to end. .

Task 4: A comprehensive fault detection mechanism should be put in place to detect the faults in the DPS domain. In such a case, the DPS traffic should be re-routed via the normal routing domain.

To achieve the above four tasks, this draft recommends four functional blocks as described in the above diagram. These functional blocks are individually designed and fine tuned to achieve various level of sophistication. Details of the building blocks are described in the below section.

Normal Routing Domain:

This is the Active-Passive or classical routing as described in section 2. Any site that is not running DPS (DPS unaware), will have only this module. Here, the routing protocol will be configured in a traditional fashion. The primary circuit will have a best metric and will be preferred and the secondary will be acting as backup.

DPS Routing domain:

This will be overlay routing domain that will be built over the existing Normal Routing domain. However some of the paths will not be considered while building the domain. For example Primary Circuits can be ignored while building the overlay domain. Alternatively overlay domain can be built in such a way that it prefers secondary circuit as a path with better metric and hence primarily route over secondary path.

It is mandatory that all the LAN routes that are available on the normal routing domain is fully injected in to the overlay DPS Routing domain. However, in order to separate the traffic, the overlay DPS Routing domain will be put into a separate VRF. Once a packet is pushed in to the overlay DPS Routing domain, the Normal Routing domain is never consulted, until the time the fault tolerance mechanism pushes traffic out of the overlay DPS Routing domain.

DPS Signaling

One of the core objective of DPS is to avoid asymmetry while routing. Hence, if a particular application is off loaded, then the remote network should also off load the same application. The signaling

module will provide a mechanism for participating sites to communicate with each other and agree about the applications that need to be off loaded.

During the signaling, sites will exchange the Application Profiles as described in section 4. Sites could be able to support more than one profile, however when they communicate a common profile must be chosen. For example, profiles can be defined as follows:

Profile 1: Critical Application: SAP, Citrix, VoIP, Telnet; Non-Critical: remaining applications

Profile 2: Critical Application: SAP, Citrix, VoIP, Telnet, SMTP, HTTP; Non-Critical: remaining applications

Site A & B can support both profile 1 and profile 2 Site C can support profile 1 Site D can support Profile 2

When Site A talks to Site C, applications will be off loaded as per definition of Profile 1

When Site A talks to Site C, applications will be off loaded as per definition of Profile 2

When Site A talks with Site B, off loading can be done either by choosing one profile or by choosing a combination of both profiles. One of the profiles can be dynamically chosen based on the network conditions prevailing at the time.

For example Site A could signal to Site B its preference for profile 1 when the link utilization level is very high. At all other times it can prefer profile 2 when dealing with Site B. Such a constraint based profile selection should be explicitly acknowledged and agreed by site B. If agreement could not be reached then it will default to normal routing.

It should be noted that the profile definition as such is static. All of the application sets will have to be individually defined by the network administrator. But selection of profiles as such can be dynamic and it could depend on a) local or remote network conditions and b) capability of the site of which the target IP address belongs to.

Profile Based Filtering Module

Once off loaded applications are defined, the framework should support a classification and filtering engine. Classification could be done via variety of application signatures. For example an

application signature can be defined as:

- 1) specific TCP port or an Server IP address
- 2) A combination of TCP port and IP Address
- 3) A Layer 7 Signature (eg: specific URL like http://www.example.com)

The profile based filter should be able to classify the packets and then push the off loaded application traffic in to the DPS Routing domain. Other traffic should be routed normally.

6. Packet flow in DPS Environment

If a site is deployed conforming to the above specifications, the following things will happen when the packet hits router.

- 1) When the packet hits the input interface, it gets classified based on the Application Signature.
- 2) If signaling happens in the forwarding plane, then the Signaling Module will communicate with the remote site and agree on the profile that will be used. If the signaling is implemented in the control plane, then the profile negotiation would have already taken place. In both the scenarios, the profile information are fed in to the profile filter.
- 3) Once the profile and application signature is determined, the profile based filter will push the packet in to the appropriate routing domain.
- 4) In the case of the packet being pushed in to the DPS Routing domain, the forwarding happens based on information available in the DPS Routing domain.
- 5) Failures might happen in the network and this might result in loss of routing information in the DPS routing domain. In such a case the DPS routing domain will push the packet out of the domain and in to the Normal Routing domain. Under no circumstances should the packet be dropped due to non-availability of routing information in the DPS routing domain.

8. Implementation Details and Gap Analysis.

8.1 DPS Routing domain:-

The main constraint is that provider devices should be transparent to DPS Domain routing. This adds more complexity to the design. The following techniques were considered for implementation:

a) Multi-Topology routing:

Multi-Topology Routing was considered but found unsuitable for DPS routing. This is because it requires dependency on providers supporting the DPS Routing scheme. Also, it forces packets to be marked only with two DSCP values (1 for normal routing and another other for DPS routing). Some implementations mandate usage of the same or similar DSCP/IP Precedence values for traffic traveling through both routing domains.

And hence it was found that MTR as not suitable for this purpose.

b) A separate VPN for DPS:

This does work fine for the DPS Routing domain. Here we extend a second sub-interface to the same POP via the secondary last mile circuit. The second sub-interface could be put on a separate VPN.

Technically this is feasible. There few implementations in the field of this type. However this is not a preferred option. Any new VPN on a MPLS network will come with extra cost and most of the time, the costs are prohibitively high.

c) Overlay VPN Using Dynamic Mesh VPN tunnels:

Here the overlay network is built using DMVPN (Dynamic Mesh VPN). Details of DMVPN is described in detail in the draft-detienne-dmvpn. There is a vendor implementation that exists today that supports a multi-point overlay tunnel as described in the draft. This could be used to create an overlay DPS Routing domain. A standard routing protocol can be run over this overlay network.

This technique is the widely deployed technique used in field deployments today. In the production environment, several implementations were done with largest one consisting of 300 sites & 2000 prefixes.

It should be noted that the proposed draft draft-detienne-dmvpn suggests that binding IPSEC encryption on to the DMVPN tunnel is mandatory. DPS Routing does not require IPSEC. Current implementations were all done without encryption. This draft recommends the DMVPN draft to be amended so that IPSEC binding requirement is de-linked from the DMVPN standard.

8.2 DPS Signaling:-

DPS signaling can be implemented in both forwarding plane and also in the control plane.

Control plane implementations are done via BGP. Here, BGP will be used as a routing protocol for the Normal Routing domain. Profile information can be exchanged via standard community. For example:

1) Profile 1 can be represented by community 100:1

2) Profile 2 can be represented by community 100:2

If a given site supports only profile 1, it will advertise site prefixes with a community 100:1. If the same site wanted to support both the profiles then the prefix would be sent with both communities, 100:1 and 100:2. At the remote end, the site will understand the capabilities based on the communities associated with the prefix.

For example Site 1 supports both profile 1 and profile 2 and hence it advertise the prefix with community 100:1 and 100:2. But site 2 supports only profile 1 (100:1). The common profile between both the sites is 100:1 and hence profile 1 will be used. It should be noted that both site 1 and site 2 with full certainty will use profile 1. Under such a scenario asymmetric routing will not happen.

Currently this scheme has been implemented using one vendor specific implementation. In the current implementation, the router will colour the packet with IP Precedence values. There will be one to one correspondence between the profile chosen and the IP Precedence values used for colouring. Once the packet is coloured, the profile based filter can be applied. Filtering will involve inspecting both the colouring done by the Signaling Module and the application to which the packet belongs to.

While this works perfectly well on large scale network, a couple of shortcomings remain.

1) The trust model could not be supported when DPS is enabled. This is because the DPS Signaling Module will remark the TOS bits during the colouring operation. Hence an alternate mechanism needs to be developed to color the packet without modifying the TOS bits.

2) Because profiles are exchanged via BGP, the router negotiates the off loaded applications list at the time of exchanging the BGP routing information. This would mean routers will not be able to react to local or remote network condition. For example, the choice

of profiles can not be changes if the link utilization is very high. The draft draft-arumuganainar-rtgwg-DPS-L4-path-preference-negotiation-00 suggests an alternate mechanism that will allow profile negotiation to happen when the TCP connection gets established. This will enable link preference to be negotiated for each TCP connection and hence such a mechanism will be more dynamic then the BGP one

8.3 Profile based Filter

Each profile is associated with a pre-determined list of applications. These applications are determined by the network administrator as an application that can be offloaded.

The administrator can define multiple profiles and associate this with one or more sites. When two sites communicate, they signal each other and agree on a single common profile. The Signaling Module will then colour the packet indicating which profile to be used while deciding the application to be off loaded.

Once the steps are complete, the profile based filter can be applied. The filter will look for the colour and then matches the protocol with corresponding off loaded application. If a match is found the packet will be pushed in to the DPS Routing domain, otherwise the packet will be normally routed.

Current implementations are done using Policy Based Routing (PBR). Here, filtering is done using a special PBR policy. This will check the colour of the packet and the pre-determined list of applications. It could then push the packet in to the appropriate routing domain.

While in theory PBR works perfectly fine, PBR is a processor intensive mechanism and hence when it is applied, the throughput of the router is adversely impacted. Hence, router scalability should be kept in mind while sizing the router for any particular site. In the future a light weight mechanism optimised for profile based filtering could be developed as well.

7. Summary

To summarise, by adopting the DPS frame work, true end to end application based routing scheme could be achieved. The DPS framework has the following advantages:

1. Give lots of room for Network Manager to determine which path should be used for which application.
2. DPS also provides a scalable frame work for achieving the above

objective.

3. By modularizing the DPS components, advanced development of individual components becomes possible. Troubleshooting will also get greatly simplified.

4. DPS capable sites can co-exist with non-DPS sites and this capability provides enough room for a phased migration. Thus DPS technology adoption is easy and simple.

5. It should be noted that DPS frame work and signaling, needs to be understood only by edge devices and any other devices in the path such as provider routers need not be aware of DPS.

```
Definitions and code {  
    line 1  
    line 2  
}
```

Special characters examples:

The characters , , ,

However, the characters \0, \&, \%, \" are displayed.

.ti 0 is displayed in text instead of used as a directive.

.\" is displayed in document instead of being treated as a comment

C:\dir\subdir\file.ext Shows inclusion of backslash \".

8 Security Considerations

TBD

9 IANA Considerations

TBD

10 References

10.1 Normative References

10.2 Informative References

11 Acknowledgements

The authors would like to thank Hesham Moussa for his review and comments.

Authors' Addresses

Arunkumar Arumuga Nainar
Tata Communications (UK) Limited
1st Floor
20 Old Bailey
London EC4M 7AN
United Kingdom

Email: arun.arumuganainar@tatacommunications.com

INTERNET-DRAFT
Intended Status: Informational draft
Expires: April 6, 2015

Arunkumar Arumuga Nainar
Tata Communications Ltd
October 3, 2014

Dynamic Path Selection use-cases
draft-arumuganainar-rtgwg-dps-use-cases-01

Abstract

Dynamic Path Selection is framework for offloading certain applications traffic that are considered not-so-critical by the network administrator on to a secondary paths (The paths that are not considered best by routing protocols). The frame work for implementing such a offloading mechanism is clearly defined in the draft draft-arumuganainar-rtgwg-DPS-requirements-01.

This draft describes the use cases for DPS in brief.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

Copyright and License Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1	Introduction	3
1.1	Terminology	3
2.	Basic assumption	3
3.	Use Case 1:- Classical DPS	5
4.	Use Case 2 :- Adaptive DPS	6
5.	Use Case 3 : Hierarchical DPS	6
5.	Use Case 4 : One Way DPS	7
5.	Use Case 5 : IP Based DPS	8
7.	Summary	8
8	Security Considerations	10
9	IANA Considerations	10
10	References	10
10.1	Normative References	10
10.2	Informative References	10
	Authors' Addresses	10

1 Introduction

In a large enterprise environment, networks location is often distributed over large geographical area (often it spreads across countries and continents). In such a case enterprises network managers often buy virtual private network from service providers. Such network is built over either MPLS or Internet/IPSEC extension to MPLS networks.

Generally the enterprise end location connects in to Service provider using last mile connections. These last mile circuits comes in various technology options and bandwidth capacities. While service provider core is very resilient and virtually having infinite capacity in terms of bandwidth, the weak link in the enterprise network is indeed the last miles.

Last miles are the costliest component for the enterprise and most of the times it contribute to 50-60% of total network costs. Yet these are the weakest link in the network. They constitutes single point of failure and does fail frequently. Hence in order to increase the network availability the network managers buy redundant links

With redundant links in place, availability puzzle is fully resolved. However, it does pose a significant question. Half of the costliest resource that he has purchased will remain idle through out the life of this network. This happen while network managers work over time to resolves the issues that arise out of congestion in the primary circuit.

There is a genuine need to identify few select application and offload them on to path that are considered to be best by routing protocols. The draft-arumuganainar-rtgwg-DPS-Requirements-00 defines requirements and the framework. This draft is a companion draft and describes various use cases for DPS.

1.1 Terminology

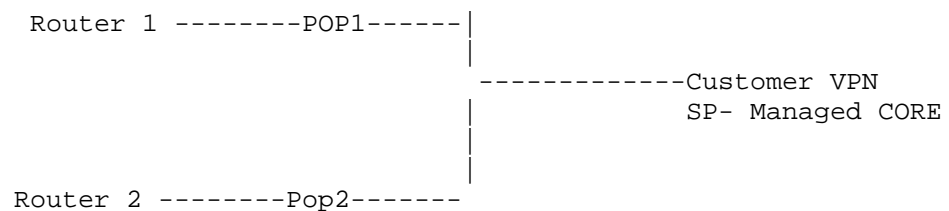
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Basic assumption

Draft address the problem faced in a typical enterprise network. Typical enterprises comprise of collection of sites / Local area network that is spread across large geographical area (across Cities ,Countries and continents). These sites are interconnected via

virtual private network. Any such enterprises typically can go to single or multiple service providers to establish this connectivity.

In such a case service provider will interconnect the site at their POP location via Last mile circuit. The choice last mile technology could be diverse such as ethernet, Frame relay , ATM , DSL, T1/E1/T3/E3 etc. Bandwidth capacity also varies from location to location (as low as 512KBPS DSL to as high as 10 GIG Ethernet). In cases when VPN connectivity is not possible ,VPN network can be extended in to a remote location via INTERNET/IPSEC technologies as well. Due to diverse nature of the connectivity option, any frame work should be agonistic to technologies and choice of providers.



Draft assumes availability is very important for the target enterprises. Hence there will be at least two last miles circuit that link them to service provider core. Again technology and last mile provider can chosen independently. Often enterprises selects a good or premium primary circuit and cheaper option for the secondary connectivity. For example

- 1) Primary : 10 MBPS Ethernet to MPLS ; Secondary :- 10 MBPS Ethernet to MPLS
- 2) Primary : 10 MBPS Ethernet to MPLS ; secondary :- 2MBPS E1 circuit to MPLS
- 3) Primary : 1.5MBPS T1 ; Secondary : ADSL 2+ Internet/IPSEC

The above samples are for illustration taken from typical examples found on the field.

Routing schemes usually designates Premium circuit option as primary. Routing protocol metrics will hence be better for the link and all the traffic goes over them as long as they are active. Traffic shift

to secondary if the primary circuit goes down . This kind of routing scheme is called Active-Passive routing.

However DPS framework as described in the draft draft-arumuganainar-rtgwg-DPS-use-cases-00 allows offloading of some application on to secondary. Network administrators can determine the offloading pattern to suit his/her needs. This draft documents several ways the offloading can be done/implemented.

3. Use Case 1:- Classical DPS

This is more common in the real life deployments today. Here Network Administrator classifies the application in two categories

1) Critical - These are application that are very critical to business and should always travel in the Links that have the best metric

2) Non-Critical - These are application that are not very critical to business. There is no significant penalty that the business would incur if the applications are not available for relatively short period of time . However , better experience of these applications are key to over all use experience level. For example , if access to Internet/ Central proxy is not available for 6 Hrs , there is no significant penalty for the business. But over all better quality Internet access is going to improve the user experiences in the network.

Such a type of classification can be done in two way.

Method 1 :- Define certain known application as critical and rest of the other application should be non critical.

Eg: Critical = { Telnet , RDP , Citrix and SAP } Non Critical = { Every thing else }

Method 2 : Non-Critical = { Traffic to Proxy server } ; Critical = { Everything Else }

It must be noted that all application should be assigned to either Critical set or non-critical ones. Un assigned applications are invalid under DPS framework. Once this is defined , DPS will kick in and start routing over various last mile links on the network

4. Use Case 2 :- Adaptive DPS

This is a slight modification to Classical DPS. Here in addition to critical and non-critical application sets , network administrator can also define not-so-critical application. The behavior of not so critical application will depend on local or remote network conditions.

To illustrate following example has be quoted. Network administrator , can define SMTP as not so critical application . He can also define network condition that will influence the path. For example if the Link utilization goes above 70% SMTP should be offloaded . Until that time SMTP will go over the primary path. Besides Link utilization many other parameters are possible such as Jitter , packet loss etc.

Again Network administrator can define multiple levels in not-so-critical application. For example SMTP can be offloaded with Link utilization is above 70% where CIFS offloading will begin when the utilization is goes above 50%

So in this scheme Network administrator will define multiple sets of application.

- 1) Critical Set : Always goes over primary path
- 2) Non Critical Set :- Always goes over secondary path
- 3) Not-So-Critical Sets :- It can either go over primary or over secondary.

It should be noted that not-so-critical application will be resolved in to critical or non-critical at the time of entry in to Edge router. Resolution in to critical and non-critical can either depend on local condition or the remote conditions. Hence Signaling should be done in the forwarding plane for adaptive DPS.

5. Use Case 3 : Hierarchical DPS

To illustrate this following example is provided.

Consider a small network consisting of 20 sites. The sites' profiles are categorized in to 3 types with the below configuration:

- * Type 1: Primary: 10 Mbps; Secondary: 2 Mbps
- * Type 2: Primary: 2 Mbps; Secondary: 8 Mbps/800 Kbps DSL

Common applications used on the network are Citrix, SAP, SMTP, FTP & HTTP. Among which Citrix and SAP are very critical to the business and needs to be protected.

The Network Manager wants to restrict Citrix and SAP to the primary link and the rest to the secondary link(Classical DPS). This works well on Type 2 sites. These are small sites predominantly consisting of thin client. However on Type 1 sites are large sites with thick client. Users utilize applications such as SMTP and Lotus notes more than SAP and Citrix. In such a scenario site will end up in a scenario where there is high congestion on the 2MBPS circuit while utilization level on 10 MBPS primary circuit is very low.

Hence Network Administrator wanted to follow a following heirarchy

When Type 1 site talks to Type 2 sites , Critical application = { SAP and Citrix }

When Type 1 circuit talks with another Type 1 site then critical application = { SAP , Citrix , SMTP and FTP }

It should be noted that Type 2 sites whether it communicates with type 1 or type 2 Critical application will always be SAP and Citrix.

The above example suggests two levels of hierarchy. In General 7 types of hierarchy is possible. 7 levels actually stems from the fact that in the current implementation IP precedence is used for coloring. IP Precedence restricts number of hierarchy to 7 . This restriction can be done away with if in the future , if alternate coloring mechanism is supported.

5. Use Case 4 : One Way DPS

With Cloud services becoming very popular , Servers are being consolidated in to a remote Data Centers. These Data centers today comes with large Bandwidth pipes . Typically DR DCs are placed in alternate location. Hence it is very common to find DCs connected to the network with Single last mile.

Withe Single last mile applications can not be offloaded . This will force remote sites to default to normal routing in order to meeting Symmetric routing requirement. This will eventually lead to polarization of links in the Remote sites.

In order to overcome this , certain sites could be made DPS capable even through they have only one leg. In such as case Overlay DPS routing domain will be created over the single last mile . This will create a illusion that the Single legged DC will be able to

participate in DPS . Impact could not be noticed on DC. But we could avoid polarization of links in the Remote sites.

One way DPS is fully compatible with Classical DPS , Hierarchical DPS and Adaptive DPS methods .

5. Use Case 5 : IP Based DPS

This is an extension of one Way DPS. But very popular in production environment. It is applied in DC-Remote site communications.

Here Network administrators wanted to offload all applications to DC but only for sub set of IP address. For example traffic to the DCs from IP address which has got odd number in last octet in it IP address be offloaded . While even addresses will be going via Primary path.

This is special case and can be implemented only between Remote sites and DCs. If the communications is between DCs-DCs or Remote sites and Remote sites is an invalid scenario. However IP Based DPS can co-exists with other forms of DPS.

For example ,

- 1) Remote site when talks with DC uses the IP Based Schema.
- 2) Remote Site when talks with another Remote sites it can use Classical or Adaptive schema

7. Summary

DPS framework provides extreme flexibility to Network Administrators. Use cases documented above are the one that implemented in production environment.

Draft draft-arumuganainar-rtgwg-DPS-requirements-00 suggests flexible frame work for implementing these use cases. There are several enhancements possible and suggested in the draft such as follows.

- 1) implementation of Signaling in the forwarding plane
- 2) Alternate mechanisms to color the packet than using IP Prec.
- 2) Supporting Layer 7 signatures in the profile based filters
- 3) Support for new and improved overlay routing mechanisms.

All these when implemented can great increase flexibility

```
Definitions and code {  
    line 1  
    line 2  
}
```

Special characters examples:

The characters , , ,

However, the characters \0, \&, \%, \" are displayed.

.ti 0 is displayed in text instead of used as a directive.

.\" is displayed in document instead of being treated as a comment

C:\dir\subdir\file.ext Shows inclusion of backslash \".

8 Security Considerations

TBD

9 IANA Considerations

TBD

10 References

10.1 Normative References

10.2 Informative References

11 Acknowledgements

The authors would like to thank Hesham Moussa for his review and comments.

Authors' Addresses

Arunkumar Arumuga Nainar
Tata Communications (UK)
1st Floor
20 Old Bailey
London EC4M 7AN
United Kingdom

Email: arun.arumuganainar@tatacommunications.com

MPLS Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 18, 2015

A. Atlas
K. Tiruveedhula
C. Bowers
Juniper Networks
J. Tantsura
Ericsson
IJ. Wijnands
Cisco Systems, Inc.
December 15, 2014

LDP Extensions to Support Maximally Redundant Trees
draft-atlas-mpls-ldp-mrt-03

Abstract

This document specifies extensions to the Label Distribution Protocol (LDP) to support the creation of label-switched paths for Maximally Redundant Trees (MRT). A prime use of MRTs is for unicast and multicast IP/LDP Fast-Reroute, which we will refer to as MRT-FRR.

The sole protocol extension to LDP is simply the ability to advertise an MRT Capability. This document describes that extension and the associated behavior expected for LSRs and LERs advertising the MRT Capability.

MRT-FRR uses LDP multi-topology extensions and requires three different multi-topology IDs to be allocated from the LDP MT-ID space.

Status of This Memo

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

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

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

This Internet-Draft will expire on June 18, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
2. Requirements Language	4
3. Terminology	4
4. Overview of LDP Signaling Extensions for MRT	5
4.1. MRT Capability Advertisement	5
4.1.1. Interaction of MRT Capability and MT Capability	6
4.1.2. Interaction of LDP MRT Capability with IPv4 and IPv6	6
4.2. Use of the Rainbow MRT MT-ID	7
4.3. MRT-Blue and MRT-Red FECs	7
5. LDP MRT FEC Advertisements	7
5.1. MRT-specific behavior	8
5.1.1. ABR behavior and use of the Rainbow FEC	8
5.1.2. Proxy-node attachment router behavior	9
5.2. LDP protocol procedures in the context of MRT label distribution	10
5.2.1. LDP peer in RFC5036	10
5.2.2. Next hop in RFC5036	10
5.2.3. Egress LSR in RFC5036	11
5.2.4. Use of Rainbow FEC to satisfy label mapping existence requirements in RFC5036	12
5.2.5. Validating FECs in routing table	13
5.2.6. Recognizing new FECs	13
5.2.7. Not propagating Rainbow FEC label mappings	13
6. Security Considerations	13
7. IANA Considerations	14
8. Acknowledgements	14
9. References	14
9.1. Normative References	14
9.2. Informative References	15
Authors' Addresses	16

1. Introduction

This document describes the LDP signaling extension and associated behavior necessary to support the architecture that defines how IP/LDP Fast-Reroute can use MRTs [I-D.ietf-rtgwg-mrt-frr-architecture]. It is necessary to be familiar with the architecture in [I-D.ietf-rtgwg-mrt-frr-architecture] to understand how and why the LDP extensions for behavior are needed.

At least one common standardized algorithm (e.g. the MRT Lowpoint algorithm explained and fully documented in [I-D.ietf-rtgwg-mrt-frr-algorithm]) is required so that the routers supporting MRT computation consistently compute the same MRTs. LDP depends on an IGP for computation of MRTs and alternates. Extensions to OSPF are defined in [I-D.atlas-ospf-mrt]. Extension to IS-IS are defined in [I-D.li-isis-mrt].

MRT can also be used to protect multicast traffic (signalled via PIM or mLDP) using either global protection or local protection [I-D.atlas-rtgwg-mrt-mc-arch]. An MRT path can be used to provide node-protection for mLDP traffic via the mechanisms described in [I-D.wijnands-mpis-mlbp-node-protection]; an MRT path can also be used to provide link protection for mLDP traffic.

For each destination, IP/LDP Fast-Reroute with MRT (MRT-FRR) creates two alternate destination-based trees separate from the shortest path forwarding used during stable operation. LDP uses the multi-topology extensions [RFC7307] to signal Forwarding Equivalency Classes (FECs) for these two sets of forwarding trees, MRT-Blue and MRT-Red.

In order to create MRT paths and support IP/LDP Fast-Reroute, a new capability extension is needed for LDP. An LDP implementation supporting MRT MUST also follow the rules described here for originating and managing FECs related to MRT, as indicated by their multi-topology ID. Network reconvergence is described in [I-D.ietf-rtgwg-mrt-frr-architecture] and the worst-case network convergence time can be flooded via the extension in Section 7 of [I-D.atlas-ospf-mrt].

IP/LDP Fast-Reroute using MRTs can provide 100% coverage for link and node failures in an arbitrary network topology where the failure doesn't partition the network. It can also be deployed incrementally; an MRT Island is formed of connected supporting routers and the MRTs are computed inside that island.

2. Requirements Language

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

3. Terminology

For ease of reading, some of the terminology defined in [I-D.ietf-rtgwg-mrt-frr-architecture] is repeated here.

Redundant Trees (RT): A pair of trees where the path from any node X to the root R along the first tree is node-disjoint with the path from the same node X to the root along the second tree. These can be computed in 2-connected graphs.

Maximally Redundant Trees (MRT): A pair of trees where the path from any node X to the root R along the first tree and the path from the same node X to the root along the second tree share the minimum number of nodes and the minimum number of links. Each such shared node is a cut-vertex. Any shared links are cut-links. Any RT is an MRT but many MRTs are not RTs. The two MRTs are referred to as MRT-Blue and MRT-Red.

MRT-Red: MRT-Red is used to describe one of the two MRTs; it is used to describe the associated forwarding topology and MT-ID. Specifically, MRT-Red is the decreasing MRT where links in the GADAG are taken in the direction from a higher topologically ordered node to a lower one.

MRT-Blue: MRT-Blue is used to describe one of the two MRTs; it is used to describe the associated forwarding topology and MT-ID. Specifically, MRT-Blue is the increasing MRT where links in the GADAG are taken in the direction from a lower topologically ordered node to a higher one.

Rainbow MRT MT-ID: It is useful to have an MT-ID that refers to the multiple MRT topologies and to the default topology. This is referred to as the Rainbow MRT MT-ID and is used by LDP to reduce signaling and permit the same label to always be advertised to all peers for the same (MT-ID, Prefix).

MRT Island: From the computing router, the set of routers that support a particular MRT profile and are connected via MRT-eligible links.

Island Border Router (IBR): A router in the MRT Island that is connected to a router not in the MRT Island and both routers are in a common area or level.

Island Neighbor (IN): A router that is not in the MRT Island but is adjacent to an IBR and in the same area/level as the IBR..

4. Overview of LDP Signaling Extensions for MRT

Routers need to know which of their LDP neighbors support MRT. This is communicated using the MRT Capability Advertisement. Supporting MRT indicates several different aspects of behavior, as listed below.

1. Sending and receiving multi-topology FEC elements, as defined in [RFC7307].
2. Understanding the Rainbow MRT MT-ID and applying the associated labels to all relevant MT-IDs.
3. Advertising the Rainbow MRT FEC to the appropriate neighbors for the appropriate prefix.
4. If acting as LDP egress for a prefix in the default topology, also acting as egress for the same prefix in MRT-Red and MRT-Blue.
5. For a FEC learned from a neighbor that does not support MRT, originating FECs for MRT-Red and MRT-Blue with the same prefix. This MRT Island egress behavior is to support an MRT Island that does not include all routers in the area/level.

4.1. MRT Capability Advertisement

A new MRT Capability Parameter TLV is defined in accordance with LDP Capability definition guidelines[RFC5561].

The LDP MRT capability can be advertised during LDP session initialization or after the LDP session is established. Advertisement of the MRT capability indicates support of the procedures for establishing the MRT-Blue and MRT-Red LSP paths detailed in this document. If the peer has not advertised the MRT capability, then it indicates that LSR does not support MRT procedures.

If a router advertises the LDP MRT capability to its peer, but the peer has not advertised the MRT capability, then the router MUST NOT advertise MRT-related FEC-label bindings to that peer.

for the same address families for which it advertises shortest-path FEC-label bindings. Therefore, an LSR advertising MRT LDP capability and shortest path FEC-label bindings for IPv4 only (or IPv6 only) would be expected to advertise MRT-related FEC-label binding for IPv4 only (or IPv6 only). An LSR advertising the MRT LDP capability and shortest-path FEC label bindings for BOTH IPv4 and IPv6 is expected to advertise MRT-related FEC-label bindings for BOTH IPv4 and IPv6. In this scenario, advertising MRT-related FEC-label bindings only for IPv4 only (or only for IPv6) is not supported.

4.2. Use of the Rainbow MRT MT-ID

Section 10.1 of [I-D.ietf-rtgwg-mrt-frr-architecture] describes the need for an area border router (ABR) to have different neighbors use different MPLS labels when sending traffic to the ABR for the same FEC. More detailed discussion of the Rainbow MRT MT-ID is provided in Section 5.1.1.

Another use for the Rainbow MRT MT-ID is for an LSR to send the Rainbow MRT MT-ID with an IMPLICIT_NULL label to indicate penultimate-hop-popping for all three types of FECs (shortest path, red, and blue). The EXPLICIT_NULL label advertised using the Rainbow MRT MT-ID similarly applies to all the types of FECs. Note that the only scenario in which it is generally useful to advertise the implicit or explicit null label for all three FEC types is when the FEC refers to the LSR itself. See Section 5.2.3 for more details.

The value of the Rainbow MRT MT-ID (TBA-MRT-LDP-2) will be assigned by IANA from the LDP MT-ID space. Prototype experiments have used the value 3999.

4.3. MRT-Blue and MRT-Red FECs

To provide MRT support in LDP, the MT Prefix FEC is used. [I-D.ietf-rtgwg-mrt-frr-architecture] contains the IANA request for the MRT-Red and MRT-Blue MT-IDs associated with the Default MRT Profile.

The MT Prefix FEC encoding is defined in [RFC7307] and is used without alteration for advertising label mappings for MRT-Blue, MRT-Red and Rainbow MRT FECs.

5. LDP MRT FEC Advertisements

This sections describes how and when labels for MRT-Red and MRT-Blue FECs are advertised. The associated LSPs must be created before a failure occurs, in order to provide protection paths which are

immediately usable by the point of local repair in the event of a failure.

In this section, we will use the term "shortest path FEC" to refer to the usual FEC associated with the shortest path destination-based forwarding tree for a given prefix as determined by the IGP. We will use the terms "red FEC" and "blue FEC" to refer to FECs associated with the MRT-Red and MRT-Blue destination-based forwarding trees for a given prefix as determined by a particular MRT algorithm.

We first describe label distribution behavior specific to MRT. Then we provide the correct interpretation of several important concepts in [RFC5036] in the context of MRT FEC label distribution.

5.1. MRT-specific behavior

5.1.1. ABR behavior and use of the Rainbow FEC

Section 10.1 of [I-D.ietf-rtgwg-mrt-frr-architecture] describes the need for an area border router (ABR) to have different neighbors use different MPLS labels when sending traffic to the ABR for the same FEC. The method to accomplish this using the Rainbow MRT MT-ID is described in detail in [I-D.ietf-rtgwg-mrt-frr-architecture]. Here we provide a brief summary. To those LDP peers in the same area as the best route to the destination, the ABR advertises two different labels corresponding to the MRT-Red and MRT-Blue forwarding trees for the destination. An LDP peer receiving these advertisements forwards MRT traffic to the ABR using these two different labels, depending on the FEC of the traffic. We refer to this as best-area advertising and forwarding behavior, which is identical to normal MRT behavior.

For all other LDP peers supporting MRT, the ABR advertises a FEC-label binding for the Rainbow MRT MT-ID scoped FEC with the label corresponding to the default forwarding tree for the destination. An LDP peer receiving this advertisement forwards MRT traffic to the ABR using this label, for both MRT Red and MRT Blue traffic. We refer to this as non-best-area advertising and forwarding behavior.

The use of the Rainbow-FEC by the ABR for non-best-area advertisements is RECOMMENDED. An ABR MAY advertise the label for the default topology in separate MRT-Blue and MRT-Red advertisements. An LSR advertising the MRT capability MUST recognize the Rainbow MRT MT-ID and associate the advertised label with the specific prefix with the MRT-Red and MRT-Blue MT-IDs associated with all MRT Profiles that advertise LDP as the forwarding mechanism.

Due to changes in topology or configuration, an ABR and a given LDP peer may need to transition from best-area advertising and forwarding

behavior to non-best-area behavior for a given destination, and vice versa. When the ABR requires best-area behavior for a red(blue) FEC, it MUST withdraw any existing label mappings advertisements for the corresponding rainbow FEC and advertise label mappings for the red(blue) FEC. When the ABR requires non-best-area behavior for a red(blue) FEC, it MUST withdraw any existing label mappings for both red and blue FECs and advertise label mappings for the corresponding Rainbow FEC label binding.

If an LSR receives a label mapping advertisement for a rainbow FEC from an MRT LDP peer while it still retains a label mapping for the corresponding red or blue FEC, the LSR MUST continue to use the label mapping for the red or blue FEC, and it MUST send a Label Release Message corresponding to the rainbow FEC label advertisement. If an LSR receives a label mapping advertisement for red or blue FEC while it still retains a label mapping for the corresponding rainbow FEC, the LSR MUST continue to use the label mapping for the rainbow FEC, and it MUST send a Label Release Message corresponding to the red or blue FEC label advertisement.

5.1.2. Proxy-node attachment router behavior

Section 11.2 of [I-D.ietf-rtgwg-mrt-frr-architecture] describes how MRT provides FRR protection for multi-homed prefixes using calculations involving a named proxy-node. This covers the scenario where a prefix is originated by a router in the same area as the MRT Island, but outside of the MRT Island. It also covers the scenario of a prefix being advertised by a multiple routers in the MRT Island.

In the named proxy-node calculation, each multi-homed prefix is represented by a conceptual proxy-node which is attached to two real proxy-node attachment routers. (A single proxy-node attachment router is allowed in the case of a prefix advertised by a same area router outside of the MRT Island which is singly connected to the MRT Island.) All routers in the MRT Island perform the same calculations to determine the same two proxy-node attachment routers for each multi-homed prefix. The resulting graph in the computation consists of the MRT Island with the proxy-node representing the multi-homed prefix directly attached to the two proxy-node attachment routers. Conceptually, one then runs the MRT algorithm on this simplified graph to determine the MRT-red and blue next-hops to reach the proxy-node, which gives the next-hops to reach the prefix. In this manner, one can see that one of the two proxy-node attachment routers will always have a MRT-red next-hop to the proxy-node while the other will always have the MRT-blue next-hop to the proxy-node. We will refer to these as the red and blue proxy-node attachment routers respectively. (In practice, the MRT-red and blue next-hops to reach the proxy-node can then be determined in a more computationally

efficient manner based on the MRT-red and blue next-hops to reach the proxy-node attachment routers, as described in [I-D.ietf-rtgwg-mrt-frr-algorithm].)

In terms of LDP behavior, a red proxy-node attachment router for a given prefix MUST originate a label mapping for the red FEC for that prefix, while the a blue proxy-node attachment router for a given prefix MUST originate a label mapping for the blue FEC for that prefix. If the red(blue) proxy-node attachment router is an Island Border Router (IBR), then when it receives a packet with the label corresponding to the red(blue) FEC for a prefix, it MUST forward the packet to the Island Neighbor (IN) whose cost was used in the selection of the IBR as a proxy-node attachment router. The IBR MUST swap the incoming label for the outgoing label corresponding to the shortest path FEC for the prefix advertised by the IN. In the case where the IN does not support LDP, the IBR MUST pop the incoming label and forward the packet to the IN.

If the proxy-node attachment router is not an IBR, then the packet MUST be removed from the MRT forwarding topology and sent along the interface(s) that caused the router to advertise the prefix. This interface might be out of the area/level/AS.

5.2. LDP protocol procedures in the context of MRT label distribution

[RFC5036] specifies the LDP label distribution procedures for shortest path FECs. In general, the same procedures can be applied to the distribution of label mappings for red and blue FECs, provided that the procedures are interpreted in the context of MRT FEC label distribution. The correct interpretation of several important concepts in [RFC5036] in the context of MRT FEC label distribution is provided below.

5.2.1. LDP peer in RFC5036

In the context of distributing label mappings for red and blue FECs, we restrict LDP peer in [RFC5036] to mean LDP peers for which the LDP MRT capability has been negotiated. In order to make this distinction clear, in this document we will use the term "MRT LDP peer" to refer to an LDP peer for which the LDP MRT capability has been negotiated.

5.2.2. Next hop in RFC5036

Several procedures in [RFC5036] use the next hop of a (shortest path) FEC to determine behavior. The next hop of the shortest path FEC is based on the shortest path forwarding tree to the prefix associated with the FEC. When the procedures of [RFC5036] are used to

distribute label mapping for red and blue FECs, the next hop for the red/blue FEC is based on the MRT-Red/Blue forwarding tree to the prefix associated with the FEC.

For example, Appendix A.1.7. of [RFC5036] specifies the response by an LSR to a change in the next hop for a FEC. For a shortest path FEC, the next hop may change as the result of the LSR running a shortest path computation on a modified IGP topology database. For the red and blue FECs, the red and blue next hops may change as the result of the LSR running a particular MRT algorithm on a modified IGP topology database.

As another example, Section 2.6.1.2 of [RFC5036] specifies how that when an LSR is using LSP Ordered Control, it may initiate the transmission of a label mapping only for a (shortest path) FEC for which it has a label mapping for the FEC next hop, or for which the LSR is the egress. The FEC next hop for a shortest path FEC is based on the shortest path forwarding tree to the prefix associated with the FEC. In the context of distributing MRT LDP labels, this procedure is understood to mean the following. When an LSR is using LSP Ordered Control, it may initiate the transmission of a label mapping only for a red(blue) FEC for which it has a label mapping for the red(blue) FEC next hop, or for which the LSR is the egress. The red or blue FEC next hop is based on the MRT-Red or Blue forwarding tree to the prefix associated with the FEC.

5.2.3. Egress LSR in RFC5036

Procedures in [RFC5036] related to Ordered Control label distribution mode rely on whether or not an LSR may act as an egress LSR for a particular FEC in order to determine whether or not the LSR may originate a label mapping for that FEC. The status of being an egress LSR for a particular FEC is also used in loop detection procedures in [RFC5036]. Section 2.6.1.2 of [RFC5036] specifies the conditions under which an LSR may act as an egress LSR with respect to a particular (shortest path) FEC.

1. The (shortest path) FEC refers to the LSR itself (including one of its directly attached interfaces).
2. The next hop router for the (shortest path) FEC is outside of the Label Switching Network.
3. (Shortest path) FEC elements are reachable by crossing a routing domain boundary.

The conditions for determining an egress LSR with respect to a red or blue FEC need to be modified. An LSR may act as an egress LSR with

respect to a particular red(blue) FEC under any of the following conditions:

1. The prefix associated with the red(blue) FEC refers to the LSR itself (including one of its directly attached interfaces).
2. The LSR is the red(blue) proxy-node attachment router with respect to the multi-homed prefix associated with the red(blue) FEC. This includes the degenerate case of a single red and blue proxy-node attachment router for a single-homed prefix.
3. The LSR is an area border router (ABR) AND the MRT LDP peer requires non-best-area advertising and forwarding behavior for the prefix associated with the FEC.

Note that condition(3) scopes an LSR's status as an egress LSR with respect to a particular FEC to a particular MRT LDP peer. Therefore, the condition "Is LSR egress for FEC?" that occurs in several procedures in [RFC5036] needs to be interpreted as "Is LSR egress for FEC with respect to Peer?"

Also note that there is no explicit condition that allows an LSR to be classified as an egress LSR with respect a red or blue FEC based only on the primary next-hop for the shortest path FEC not supporting LDP, or not supporting LDP MRT capability. These situations are covered by the proxy-node attachment router and ABR conditions (conditions 2 and 3). In particular, an Island Border Router is not the egress LSR for a red(blue) FEC unless it is also the red(blue) proxy-node attachment router for that FEC.

Also note that in general a proxy-node attachment router for a given prefix should not advertise an implicit or explicit null label for the corresponding red or blue FEC, even though it may be an egress LSR for the shortest path FEC. In general, the proxy-node attachment router needs to forward red or blue traffic for that prefix to a particular loop free island neighbor, which may be different from the shortest path next-hop. The proxy-node attachment router needs to receive the red or blue traffic with a non-null label to correctly forward it.

5.2.4. Use of Rainbow FEC to satisfy label mapping existence requirements in RFC5036

Several procedures in [RFC5036] require the LSR to determine if it has previously received and retained a label mapping for a FEC from the next hop. In the case of an LSR that has received and retained a label mapping for a Rainbow FEC from an ABR, the label mapping for the Rainbow FEC satisfies the label mapping existence requirement for

the corresponding red and blue FECs. Label mapping existence requirements in the context of MRT LDP label distribution are modified as: "Has LSR previously received and retained a label mapping for the red(blue) FEC (or the corresponding Rainbow FEC) from the red(blue) next hop?"

As an example, this behavior allows an LSR which has received and retained a label mapping for the Rainbow FEC to advertise label mappings for the corresponding red and blue FECs when operating in Ordered Control label distribution mode.

5.2.5. Validating FECs in routing table

In [RFC5036] an LSR uses its routing table to validate prefixes associated with shortest path FECs. For example, section 3.5.7.1 of [RFC5036] specifies that "an LSR receiving a Label Mapping message from a downstream LSR for a Prefix SHOULD NOT use the label for forwarding unless its routing table contains an entry that exactly matches the FEC Element." In the context of MRT FECs, a red or blue FEC element matches a routing table entry if the corresponding shortest path FEC element matches a routing table entry.

5.2.6. Recognizing new FECs

Section A.1.6 of [RFC5036] describes the response of an LSR to the "Recognize New FEC" event, which occurs when an LSR learns a new (shortest path) FEC via the routing table. In the context of MRT FECs, when MRT LDP capability has been enabled, when an LSR learns a new shortest path FEC, it should generate "Recognize New FEC" events for the corresponding red and blue FECs, in addition to the "Recognize New FEC" event for the shortest path FEC.

5.2.7. Not propagating Rainbow FEC label mappings

A label mapping for the Rainbow FEC should only be originated by an ABR under the conditions described in Section 5.1.1. A neighbor of the ABR that receives a label mapping for the Rainbow FEC MUST NOT propagate a label mapping for that Rainbow FEC.

6. Security Considerations

The labels distributed by the extensions in this document create additional forwarding paths that do not follow shortest path routes. The transit label swapping operations defining these alternative forwarding paths are created during normal operations (before a failure occurs). Therefore, a malicious packet with an appropriate label injected into the network from a compromised location would be forwarded to a destination along a non-shortest

path. When this technology is deployed, a network security design should not rely on assumptions about potentially malicious traffic only following shortest paths.

It should be noted that the creation of non-shortest forwarding paths is not unique to MRT.

7. IANA Considerations

IANA is requested to allocate a value for the new LDP Capability TLV (the first free value in the range 0x0500 to 0x05FF) from the LDP registry "TLV Type Name Space": MRT Capability TLV (TBA-MRT-LDP-1).

Value	Description	Reference	Notes / Reg. Date
TBA-MRT-LDP-1	MRT Capability TLV	[This draft]	

IANA is requested to allocate a value for the new LDP Status Code (the first free value in the range 0x00000032-0x00000036) from the LDP registry "Status Code Name Space": "MRT Capability negotiated without MT Capability" (TBA-MRT-LDP-3).

Value	E	Description	Reference	Notes / Reg. Date
TBA-MRT-LDP-3	0	MRT Capability negotiated without MT Capability	[This draft]	

IANA is requested to allocate a value from the MPLS Multi-Topology Identifiers Name Space [RFC7307]: Rainbow MRT MT-ID (TBA-MRT-LDP-2).

Value	Purpose	Reference
TBA-MRT-LDP-2	Rainbow MRT MT-ID	[This draft]

8. Acknowledgements

The authors would like to thank Ross Callon, Loa Andersson, Stewart Bryant, Mach Chen, and Greg Mirsky for their suggestions.

9. References

9.1. Normative References

[I-D.ietf-rtgwg-mrt-frr-algorithm]
 Enyedi, G., Csaszar, A., Atlas, A., Bowers, C., and A. Gopalan, "Algorithms for computing Maximally Redundant Trees for IP/LDP Fast-Reroute", draft-rtgwg-mrt-frr-algorithm-01 (work in progress), July 2014.

- [I-D.ietf-rtgwg-mrt-frr-architecture]
Atlas, A., Kebler, R., Bowers, C., Enyedi, G., Csaszar, A., Tantsura, J., Konstantynowicz, M., and R. White, "An Architecture for IP/LDP Fast-Reroute Using Maximally Redundant Trees", draft-rtgwg-mrt-frr-architecture-04 (work in progress), July 2014.
- [RFC5036] Andersson, L., Minei, I., and B. Thomas, "LDP Specification", RFC 5036, October 2007.
- [RFC5561] Thomas, B., Raza, K., Aggarwal, S., Aggarwal, R., and JL. Le Roux, "LDP Capabilities", RFC 5561, July 2009.
- [RFC7307] Zhao, Q., Raza, K., Zhou, C., Fang, L., Li, L., and D. King, "LDP Extensions for Multi-Topology", RFC 7307, July 2014.

9.2. Informative References

- [I-D.atlas-ospf-mrt]
Atlas, A., Hegde, S., Bowers, C., and J. Tantsura, "OSPF Extensions to Support Maximally Redundant Trees", draft-atlas-ospf-mrt-02 (work in progress), July 2014.
- [I-D.atlas-rtgwg-mrt-mc-arch]
Atlas, A., Kebler, R., Wijnands, I., Csaszar, A., and G. Enyedi, "An Architecture for Multicast Protection Using Maximally Redundant Trees", draft-atlas-rtgwg-mrt-mc-arch-02 (work in progress), July 2013.
- [I-D.li-isis-mrt]
Li, Z., Wu, N., Zhao, Q., Atlas, A., Bowers, C., and J. Tantsura, "Intermediate System to Intermediate System (IS-IS) Extensions for Maximally Redundant Trees(MRT)", draft-li-isis-mrt-01 (work in progress), July 2014.
- [I-D.wijnands-mpls-mldp-node-protection]
Wijnands, I., Rosen, E., Raza, K., Tantsura, J., Atlas, A., and Q. Zhao, "mLDP Node Protection", draft-wijnands-mpls-mldp-node-protection-04 (work in progress), June 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

Authors' Addresses

Alia Atlas
Juniper Networks
10 Technology Park Drive
Westford, MA 01886
USA

Email: akatlas@juniper.net

Kishore Tiruveedhula
Juniper Networks
10 Technology Park Drive
Westford, MA 01886
USA

Email: kishoret@juniper.net

Chris Bowers
Juniper Networks
1194 N. Mathilda Ave.
Sunnyvale, CA 94089
USA

Email: cbowers@juniper.net

Jeff Tantsura
Ericsson
300 Holger Way
San Jose, CA 95134
USA

Email: jeff.tantsura@ericsson.com

IJsbrand Wijnands
Cisco Systems, Inc.

Email: ice@cisco.com

OSPF Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 22, 2015

A. Atlas
S. Hegde
C. Bowers
Juniper Networks
J. Tantsura
Ericsson
Z. Li
Huawei Technologies
July 21, 2014

OSPF Extensions to Support Maximally Redundant Trees
draft-atlas-ospf-mrt-03

Abstract

This document specifies extensions to OSPF to support the distributed computation of Maximally Redundant Trees (MRT). Some example uses of the MRTs include IP/LDP Fast-Reroute and global protection or live-live for multicast traffic. The extensions indicate what MRT profile(s) each router supports. Different MRT profiles can be defined to support different uses and to allow transitioning of capabilities. An extension is introduced to flood MRT-Ineligible links, due to administrative policy.

The need for a mechanism to allow routers to advertise a worst-case FIB compute/install time is well understood for controlling convergence. This specification introduces the Controlled Convergence TLV to be carried in the Router Information LSA.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 22, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Requirements Language	3
3. Terminology	3
4. Overview of OSPF Extensions for MRT	4
4.1. Supporting MRT Profiles	4
4.2. GADAG Root Selection	5
4.3. Triggering an MRT Computation	5
5. MRT Capability Advertisement	6
5.1. Advertising MRT Capability in OSPFv2	6
5.2. Advertising MRT Capability in OSPFv3	7
5.3. MRT Profile TLV in Router Information LSA	8
6. Advertising MRT-ineligible links for MRT	9
6.1. MRT-Ineligible Link sub-TLV	10
7. Worst-Case Network Convergence Time	10
8. Backwards Compatibility	11
8.1. Handling MRT Capability Changes	12
9. Implementation Status	12
10. Security Considerations	12
11. IANA Considerations	12
12. References	13
12.1. Normative References	13
12.2. Informative References	13
Authors' Addresses	14

1. Introduction

This document describes the OSPF extensions necessary to support the architecture that defines how IP/LDP Fast-Reroute can use MRTs [I-D.ietf-rtgwg-mrt-frr-architecture]. At least one common standardized algorithm (such as the lowpoint algorithm explained and fully documented in [I-D.ietf-rtgwg-mrt-frr-algorithm]) is required

so that the routers supporting MRT computation consistently compute the same MRTs. MRT can also be used to protect multicast traffic via either global protection or local protection.[I-D.atlas-rtgwg-mrt-mc-arch]

IP/LDP Fast-Reroute using MRTs can provide 100% coverage for link and node failures in an arbitrary network topology where the failure doesn't split the network. It can also be deployed incrementally inside an OSPF area; an MRT Island is formed of connected supporting routers and the MRTs are computed inside that island.

In the default MRT profile, a supporting router both computes the MRTs and creates the necessary transit forwarding state necessary to provide the two additional forwarding topologies, known as MRT-Blue and MRT-Red.

2. Requirements Language

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

3. Terminology

For ease of reading, some of the terminology defined in [I-D.ietf-rtgwg-mrt-frr-architecture] is repeated here.

network graph: A graph that reflects the network topology where all links connect exactly two nodes and broadcast links have been transformed into the standard pseudo-node representation.

Redundant Trees (RT): A pair of trees where the path from any node X to the root R along the first tree is node-disjoint with the path from the same node X to the root along the second tree. These can be computed in 2-connected graphs.

Maximally Redundant Trees (MRT): A pair of trees where the path from any node X to the root R along the first tree and the path from the same node X to the root along the second tree share the minimum number of nodes and the minimum number of links. Each such shared node is a cut-vertex. Any shared links are cut-links. Any RT is an MRT but many MRTs are not RTs.

MRT Island: From the computing router, the set of routers that support a particular MRT profile and are connected via MRT-eligible links.

GADAG: Generalized Almost Directed Acyclic Graph - a graph that is the combination of the ADAGs of all blocks. Transforming a network graph into a GADAG is part of the MRT algorithm.

MRT-Red: MRT-Red is used to describe one of the two MRTs; it is used to describe the associated forwarding topology and MT-ID. Specifically, MRT-Red is the decreasing MRT where links in the GADAG are taken in the direction from a higher topologically ordered node to a lower one.

MRT-Blue: MRT-Blue is used to describe one of the two MRTs; it is used to describe the associated forwarding topology and MT-ID. Specifically, MRT-Blue is the increasing MRT where links in the GADAG are taken in the direction from a lower topologically ordered node to a higher one.

4. Overview of OSPF Extensions for MRT

There are two separate aspects that need to be advertised in OSPF. Both derive from the need for all routers supporting an MRT profile to compute the same pair of MRTs to each destination. By executing the same algorithm on the same network graph, distributed routers will compute the same MRTs. Convergence considerations are discussed in [I-D.ietf-rtgwg-mrt-frr-architecture].

The first aspect that must be advertised is which MRT profile(s) are supported and the associated GADAG Root Selection Priority. The second aspect that must be advertised is any links that are not eligible, due to administrative policy, to be part of the MRTs. This must be advertised consistently across the area so that all routers in the MRT Island use the same network graph.

4.1. Supporting MRT Profiles

An MRT Profile defines the exact MRT Algorithm, the MRT-Red LDP MT-ID, the MRT-Blue LDP MT-ID, and the forwarding mechanisms supported for the transit MRT-Red and MRT-Blue forwarding topologies. Finally, the MRT Profile defines exact behavioral rules such as:

- o how reconvergence is handled,
- o inter-area forwarding behavior,

A router that advertises support for an MRT Profile MUST provide the specified forwarding mechanism for its MRT-Red and MRT-Blue forwarding topologies. A router that advertises support for an MRT Profile MUST implement an algorithm that produces the same set of MRT-Red and MRT-Blue next-hops for its MRT-Red and MRT-Blue

topologies as is provided by the algorithm specified in the MRT Profile.

A router MAY indicate support for multiple MRT Profiles. A router computes its local MRT Island for each separate MRT Profile that the router supports. Supporting multiple MRT Profiles also provides a mechanism for transitioning from one profile to another. Different uses of MRT forwarding topologies may behave better on different MRT profiles.

The default MRT Profile is defined in [I-D.ietf-rtgwg-mrt-frr-architecture]. Its behavior is intended to support IP/LDP unicast and multicast fast-reroute.

4.2. GADAG Root Selection

One aspect of the MRT algorithms is that the selection of the GADAG root can affect the alternates and the traffic through that GADAG root. Therefore, it is important to provide an operator with control over which router will play the role of GADAG root. A measure of the centrality of a node may help determine how good a choice a particular node is.

The GADAG Root Selection Policy (defined as part of an MRT profile) may make use of the GADAG Root Selection Priority value advertised in the MRT Profile TLV of the Router Information LSA. For example, the GADAG Root Selection Policy for the default MRT profile is the following: Among the routers in the MRT Island and with the highest priority advertised, an implementation MUST pick the router with the highest Router ID to be the GADAG root.

4.3. Triggering an MRT Computation

When an MRT Computation is triggered, it is triggered for a given MRT Profile in a given area. First, the associated MRT Island is determined. Then, the GADAG Root is selected. Finally, the actual MRT algorithm is run to compute the transit MRT-Red and MRT-Blue topologies. Additionally, the router MAY choose to compute MRT-FRR alternates or make other use of the MRT computation results.

Prefixes can be attached and detached and have their associated MRT-Red and MRT-Blue next-hops computed without requiring a new MRT computation.

5. MRT Capability Advertisement

A router that supports MRT indicates this by setting a newly defined M bit in the Router LSA. If the router provides no other information via a separate MRT Profile TLV, then the router supports the default MRT Profile with a GADAG Root Selection Priority of 128.

In addition, a router can advertise a newly-defined MRT Profile TLV within the scope of the OSPF router information LSA [RFC4970]. This TLV also includes the GADAG Root Selection Priority.

5.1. Advertising MRT Capability in OSPFv2

A new M-bit is defined in the Router-LSA (defined in [RFC2328] and updated in [RFC4915]), as pictured below.

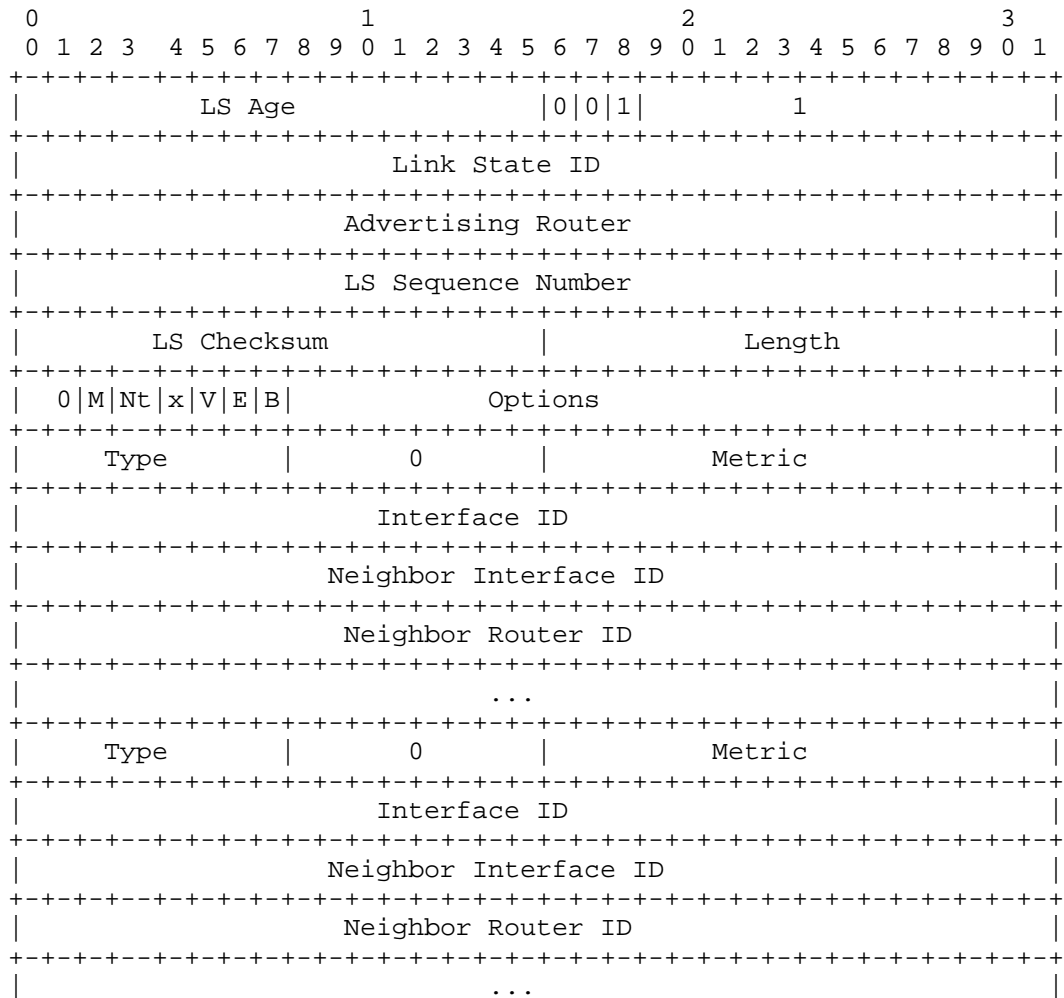
0										1										2										3																																																																																															
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1																																																																																														
										LS age																				Options																				1																																																																											
																																										Link State ID																																																																																			
																																										Advertising Router																																																																																			
																																										LS sequence number																																																																																			
																																										LS checksum																																										length																																									
* * M N W V E B										0																				# links																																																																																															
																																										Link ID																																																																																			
																																										Link Data																																																																																			
										Type																				# MT-ID																				metric																																																																											
										MT-ID																				0																				MT-ID										metric																																																																	
																																										...																																																																																			
										MT-ID																				0																				MT-ID										metric																																																																	
																																										Link ID																																																																																			
																																										Link Data																																																																																			
																																										...																																																																																			

M-bit in OSPFv2 Router LSA

M bit: When set, the router supports MRT. If no separate MRT Profile TLV is advertised in the associated Router Information LSA, then the router supports the default MRT Profile and has a GADAG Root Selection Priority of 128.

5.2. Advertising MRT Capability in OSPFv3

Similarly, the M-bit is defined in the OSPFv3 Router LSA as shown below. Since there can be multiple router LSAs, the M-bit needs to be set on all of them.



M-bit in OSPFv3 Router LSA

M bit: When set, the router supports MRT. If no separate MRT Profile TLV is advertised in the associated Router Information LSA, then the router supports the default MRT Profile and has a GADAG Root Selection Priority of 128.

5.3. MRT Profile TLV in Router Information LSA

A router may advertise an MRT Profile TLV to indicate support for multiple MRT Profiles, for a non-default MRT Profile, and/or to indicate a non-default GADAG Root Selection Priority. The MRT Profile TLV is advertised within the OSPF router information LSA

which is defined for both OSPFv2 and OSPFv3 in [RFC4970]. The RI LSA MUST have area scope.

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
Type										Length																													
Profile ID(1)										GADAG Priority										Reserved																			
.....																																							
Profile ID(n)										GADAG Priority										Reserved																			

TYPE: TBA-MRT-OSPF-1 (To Be Allocated by IANA)

LENGTH: 4 * (number of Profiles)

Profile ID : 1 byte

GADAG Priority: 1 byte

MRT Profile TLV in Router Information LSA

Each Profile ID listed indicates support for a given MRT Profile, as defined in [I-D.ietf-rtgwg-mrt-frr-architecture]. A Profile ID value of 0 corresponds to the default MRT profile.

The GADAG Priority is the GADAG Root Selection Priority associated with the advertising router in the MRT Island for the associated MRT Profile, as indicated by the Profile ID. If multiple MRT Profiles are supported, then the length of this TLV varies. The ordering of the profiles inside the TLV is not significant. Multiple appearances of this TLV is not an error.

Lack of support for the default MRT profile is indicated by the presence of an MRT Profile TLV with a non-zero Profile ID value, and the absence of an MRT Profile TLV with a zero Profile ID value.

6. Advertising MRT-ineligible links for MRT

Due to administrative policy, some otherwise eligible links in the network topology may need to be excluded from the network graph upon which the MRT algorithm is run. Since the same network graph must be used across the area, it is critical for OSPF to flood which links to exclude from the MRT calculation. This is done by introducing a new MRT-Ineligible Link sub-TLV. For OSPFv2, this sub-TLV is carried in

the Extended Link TLV defined in [I-D.ietf-ospf-segment-routing-extensions]. For OSPFv3, this sub-TLV is carried in the Router-Link TLV defined in [I-D.ietf-ospf-ospfv3-lsa-extend].

If a link is marked by administrative policy as MRT-Ineligible, then a router MUST flood the OSPFv2 Extended Link TLV (or OSPFv3 Router-Link TLV) for that link, including the MRT-Ineligible Link sub-TLV. The OSPFv2 Extended Link TLV and OSPFv3 Router-Link TLV have area wide scope.

6.1. MRT-Ineligible Link sub-TLV

0										1										2										3																			
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1																		
Type																																	Length																

TYPE: TBA-MRT-OSPF-2 in OSPFv2 Extended Link TLV
 TBA-MRT-OSPF-3 in OSPFv3 Router-Link TLV
 (To Be Allocated by IANA)
 LENGTH: 0

MRT-Ineligible Link sub-TLV

This zero-length sub-TLV can appear in the OSPFv2 Extended Link TLV or the OSPFv3 Router-Link TLV. Its presence indicates that the associated link MUST NOT be used in the MRT calculation for all profiles.

7. Worst-Case Network Convergence Time

As part of converging the network after a single failure, Section 12.2 of [I-D.ietf-rtgwg-mrt-frr-architecture] describes the need to wait for a configured or advertised period for all routers to be using their new SPTs. Similarly, any work on avoiding micro-forwarding loops during convergence[RFC5715] requires determining the maximum among all routers in the area of the worst-case route computation and FIB installation time. More details on the specific reasoning and need for flooding it are given in [I-D.atlas-bryant-shand-lf-timers].

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
Type										Length																													
Reserved										FIB compute/install time																													

TYPE: TBA-MRT-OSPF-4 (To Be Allocated by IANA)

LENGTH: 4

FIB compute/install time: This is the worst-case time the router may take to compute and install all OSPF routes in the area after a change to a stable network. The value is in milliseconds.

Controlled Convergence TLV in Router Information LSA

The Controlled Convergence TLV is carried in the Router Information LSA and flooded with area-wide scope. The FIB compute/install time value sent by a router SHOULD be an estimate taking into account network scale or real-time measurements, or both. Advertisements SHOULD be dampened to avoid frequent communication of small changes in the FIB compute/install time.

A router receiving the Controlled Convergence sub-TLV SHOULD estimate the network convergence time as the maximum of the FIB compute/install times advertised by the routers in an area, including itself. In order to account for routers that do not advertise the Controlled Convergence sub-TLV, a router MAY use a locally configured minimum network convergence time as a lower bound on the computed network convergence time. A router MAY use a locally configured maximum network convergence time as an upper bound on the computed network convergence time.

8. Backwards Compatibility

The MRT capability bit, the MRT Profile, the MRT-Ineligible Link, and the OSPFv3 MRT-Ineligible Link TLVs are defined in this document. They should not introduce any interoperability issues. Routers that do not support the MRT capability bit in the router LSA SHOULD silently ignore it. Routers that do not support the new MRT-related TLVs SHOULD silently ignore them.

8.1. Handling MRT Capability Changes

When a router changes from supporting MRT to not supporting MRT, it is possible that Router Information LSAs with MRT-related TLVs remain in the neighbors' database briefly. Such MRT-related TLVs SHOULD be ignored when the associated Router LSA from that router does not have the MRT capability set in its Router LSA.

When a router changes from not supporting MRT to supporting MRT, it will flood its Router LSA(s) with the M-bit set and may send an updated Router Information LSA. If a Router LSA is received with the M-bit newly set, an MRT computation SHOULD be scheduled but MAY be delayed up to 60 seconds to allow reception of updated related Router Information LSAs. In general, when changes in MRT-related information is received, an MRT computation SHOULD be triggered.

The rationale behind using the M bit in router LSA is to handle the MRT capability changes gracefully in case of version upgrade/downgrade. The M bit in router LSA ensures the latest "MRT capability" information is available for computation when there is a downgrade to the version that doesn't support MRT and RI LSA.

9. Implementation Status

[RFC Editor: please remove this section prior to publication.]

Please see [I-D.ietf-rtgwg-mrt-frr-architecture] for details on implementation status.

10. Security Considerations

This OSPF extension is not believed to introduce new security concerns. It relies upon the security architecture already provided for Router LSAs and Router Information LSAs.

11. IANA Considerations

IANA is requested to allocate values for the following OSPF Router Information TLV Types [RFC4970]: MRT Profile TLV (TBA-MRT-OSPF-1), and Controlled Convergence TLV (TBA-MRT-OSPF-4).

IANA is requested to allocate a value from the OSPF Extended Link LSA sub-TLV registry [I-D.ietf-ospf-segment-routing-extensions] for the MRT-Ineligible Link sub-TLV (TBA-MRT-OSPF-2).

IANA is requested to allocate a value from the OSPFv3 Extended-LSA sub-TLV registry [I-D.ietf-ospf-ospfv3-lsa-extend] for the MRT-Ineligible Link sub-TLV (TBA-MRT-OSPF-3).

12. References

12.1. Normative References

- [I-D.ietf-ospf-ospfv3-lsa-extend]
Lindem, A., Mirtorabi, S., Roy, A., and F. Baker, "OSPFv3 LSA Extendibility", draft-ietf-ospf-ospfv3-lsa-extend-03 (work in progress), May 2014.
- [I-D.ietf-ospf-segment-routing-extensions]
Psenak, P., Previdi, S., Filsfils, C., Gredler, H., Shakir, R., Henderickx, W., and J. Tantsura, "OSPF Extensions for Segment Routing", draft-ietf-ospf-segment-routing-extensions-01 (work in progress), July 2014.
- [I-D.ietf-rtgwg-mrt-frr-algorithm]
Enyedi, G., Csaszar, A., Atlas, A., Bowers, C., and A. Gopalan, "Algorithms for computing Maximally Redundant Trees for IP/LDP Fast-Reroute", draft-rtgwg-mrt-frr-algorithm-01 (work in progress), July 2014.
- [I-D.ietf-rtgwg-mrt-frr-architecture]
Atlas, A., Kebler, R., Bowers, C., Enyedi, G., Csaszar, A., Tantsura, J., Konstantynowicz, M., and R. White, "An Architecture for IP/LDP Fast-Reroute Using Maximally Redundant Trees", draft-rtgwg-mrt-frr-architecture-04 (work in progress), July 2014.
- [RFC2328] Moy, J., "OSPF Version 2", STD 54, RFC 2328, April 1998.
- [RFC4970] Lindem, A., Shen, N., Vasseur, JP., Aggarwal, R., and S. Shaffer, "Extensions to OSPF for Advertising Optional Router Capabilities", RFC 4970, July 2007.
- [RFC5340] Coltun, R., Ferguson, D., Moy, J., and A. Lindem, "OSPF for IPv6", RFC 5340, July 2008.

12.2. Informative References

- [I-D.atlas-bryant-shand-lf-timers]
K, A. and S. Bryant, "Synchronisation of Loop Free Timer Values", draft-atlas-bryant-shand-lf-timers-04 (work in progress), February 2008.

[I-D.atlas-rtgwg-mrt-mc-arch]

Atlas, A., Kebler, R., Wijnands, I., Csaszar, A., and G. Envedi, "An Architecture for Multicast Protection Using Maximally Redundant Trees", draft-atlas-rtgwg-mrt-mc-arch-02 (work in progress), July 2013.

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

[RFC3137] Retana, A., Nguyen, L., White, R., Zinin, A., and D. McPherson, "OSPF Stub Router Advertisement", RFC 3137, June 2001.

[RFC4915] Psenak, P., Mirtorabi, S., Roy, A., Nguyen, L., and P. Pillay-Esnault, "Multi-Topology (MT) Routing in OSPF", RFC 4915, June 2007.

[RFC5715] Shand, M. and S. Bryant, "A Framework for Loop-Free Convergence", RFC 5715, January 2010.

Authors' Addresses

Alia Atlas
Juniper Networks
10 Technology Park Drive
Westford, MA 01886
USA

Email: akatlas@juniper.net

Shraddha Hegde
Juniper Networks
Embassy Business Park
Bangalore, KA 560093
India

Email: shraddha@juniper.net

Chris Bowers
Juniper Networks
1194 N. Mathilda Ave.
Sunnyvale, CA 94089
USA

Email: cbowers@juniper.net

Jeff Tantsura
Ericsson
300 Holger Way
San Jose, CA 95134
USA

Email: jeff.tantsura@ericsson.com

Zhenbin Li
Huawei Technologies
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095
China

Email: lizhenbin@huawei.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2015

B. Decraene
Orange
March 9, 2015

Back-off SPF algorithm for link state IGP
draft-decraene-rtgwg-backoff-algo-01

Abstract

This document defines a standard algorithm to back-off link-state IGP SPF computations.

Having one standardized algorithm improves interoperability by reducing the probability and/or duration of transient forwarding loops during the IGP convergence in the area/level when the network reacts to multiple consecutive events.

Requirements Language

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

Status of This Memo

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

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

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

This Internet-Draft will expire on September 10, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

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

1. Introduction

Link state IGP, such as IS-IS [ISO10589-Second-Edition] and OSPF [RFC2328], performs distributed computation on all nodes of the area/level. In order to have consistent routing tables across the network, such distributed computation requires that all routers have the same vision of the network (Link State DataBase (LSDB)) and perform their computation at the same time.

In general, when the network is stable, there is a desire to compute the new SPF as soon as the failure is known, in order to quickly route around the failure. However, when the network is experiencing multiple consecutive failures over a short period of time, there is a desire to limit the frequency of SPF computations. Indeed, this allow reducing the control plane resources used by IGP and all protocols/sub system reacting on it such as LDP, RSVP-TE, BGP, Fast ReRoute computations, FIB updates..., reduce the churn on nodes and in the network, in particular reduce side effects such as micro-loops which may happen during each IGP convergence.

To allow for this, some back-off algorithm have been implemented. Different implementations choose different algorithms, hence in a multi-vendor network, it's not possible to enforce that all routers triggers their SPF computation after the same waiting delay. This situation increases the average differential delay between routers end of RIB computation. It also increases the probability that different routers compute their RIB based on a different LSDB. Both increases the probability and/or duration of micro-loops.

To allow for multi-vendors networks having all the routers delaying their SPF for the same duration, this document specifies a standardized algorithm. Implementations may offer alternative optional algorithms.

2. High level goals

The high level goals of this algorithm are the following:

- o Very fast convergence for single simple events (link failure).
- o Fast convergence in general while the IGP stability is considered under control.
- o A long delay when the IGP stability is considered out of control, in order to let all related process calm down.
- o At any time, try to avoid using different SPF_TIMERS values for nodes in the area/level. Even though not all nodes will receive IGP message at the same time (due to difference in distance from the source and due to different flooding implementations on the path from the source).

3. Definitions and parameters

IGP events: An LSDB change requiring a new RIB computation (topology change, prefix change, metric change). No distinction is done between the type of computation performed (e.g. full SPF, incremental SPF, PRC). The type of computation is a local consideration.

The SPF_DELAY timer can take the following values:

INITIAL_WAIT: a very small delay to quickly handle link failure. e.g. 0 millisecond.

FAST_WAIT: a small delay to have a fast convergence. e.g. 50-100 millisecond. Note: we want to be fast, but as this failure requires multiple IGP events, being too fast increase the probability to receive additional IGP events just after the RIB computation.

LONG_WAIT: a long delay as IGP is unstable. e.g. 2 seconds. Note: let's bring calm in the IGP.

The TIME_TO_CONVERGE timer is the time to learn all the IGP events related to a single failure (e.g. node failure, SRLG failure). e.g. 1 second. It's mostly dependent on variation of failure detection times between all nodes which are neighbour to the failure, and then may depend on different flooding algorithms of nodes in the network.

The HOLD_DOWN timer is the time needed with no IGP events received, before considering that the IGP is quiet again and we can set the SPF_DELAY back to INITIAL_WAIT. e.g. 5 seconds.

4. Principle of SPF delay algorithm

The first IGP event is handled very quickly (`INITIAL_WAIT`) in order to be very reactive for the first event if it only needs one IGP event (e.g. link failure, prefix change).

If more IGP events are received quickly after, we consider that they are related to the same single failure, and handle the IGP events relatively quickly (`FAST_WAIT`) during the time needed to receive all the IGP events related to the failure (`TIME_TO_CONVERGE`).

If IGP events are still received after this time, then the network is presumably experiencing multiple independent failures and the while waiting for its stability, the computations are delayed for a longer time (`LONG_WAIT`).

Note: previous SPF delay algorithms used to count the number of RIB computations. However, as all nodes may receive the LSP events in a different way we cannot assume that all nodes will perform the same number of SPF computations or that they will schedule them at the same time. For example, assuming that the SPF delay is 50 ms, node R1 may receive 3 IGP events (E1, E2, E3) in those 50 ms and hence will perform a single routing computation. While another node R2 may only receive 2 events (E1, E2) in those 50ms and hence will schedule another routing computation when further receiving E3. That's why this document prefers to define a time limit (`TIME_TO_CONVERGE`) since the first event, rather than a number of routing computations.

5. Specification of SPF delay algorithm

When the previous IGP events is more than `HOLD_DOWN` ago:

- o The IGP is set to the `QUIET` state.

When the IGP is in the `QUIET` state and an IGP event is received:

- o The time of this first IGP event is stored in `FIRST_EVENT_TIME`.
- o The next RIB computation time is set to LSP receive time + `INITIAL_WAIT`.
- o The IGP is set to the `FAST_WAIT` state.

When the IGP is in the `FAST_WAIT` state and an IGP event is received:

- o If more than `TIME_TO_CONVERGE` has passed since `FIRST_EVENT_TIME`, then the IGP is set to the `HOLD_DOWN` state.

- o If the next RIB_computation time is in the past, set the next RIB computation time to LSP receive time + FAST_WAIT.

When the IGP is in the HOLD_DOWN state and an IGP event is received:

- o If the next RIB_computation time is in the past, set the next RIB computation time to LSP receive time + LONG_WAIT.

6. Impact on micro-loops

Micro-loops during IGP convergence are due to a non synchronized or non ordered update of the forwarding information tables (FIB) [RFC5715] [RFC6976] [I-D.litkowski-rtgwg-spf-uloop-pb-statement]. FIB are installed after multiple steps such as SPF wait time, SPF computation, FIB distribution and FIB update. This document only address the first contribution. This standardized procedure reduces the probability and/or duration of micro-loops when the IGP experience multiple consecutive events. It does not remove all micro-loops. However, it is beneficial and its cost seems limited compared to full solutions such as [RFC5715] or [RFC6976].

7. IANA Considerations

No IANA actions required.

8. Security considerations

This document has no impact on the security of the IGP.

9. Acknowledgements

We would like to acknowledge Hannes Gredler, Les Ginsberg and Pierre Francois for the discussions related to this document.

10. References

10.1. Normative References

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

10.2. Informative References

- [I-D.litkowski-rtgwg-spf-uloop-pb-statement]
Litkowski, S., "Link State protocols SPF trigger and delay algorithm impact on IGP microloops", draft-litkowski-rtgwg-spf-uloop-pb-statement-02 (work in progress), March 2015.

[ISO10589-Second-Edition]

International Organization for Standardization,
"Intermediate system to Intermediate system intra-domain
routing information exchange protocol for use in
conjunction with the protocol for providing the
connectionless-mode Network Service (ISO 8473)", ISO/IEC
10589:2002, Second Edition, Nov 2002.

[RFC2328] Moy, J., "OSPF Version 2", STD 54, RFC 2328, April 1998.

[RFC5715] Shand, M. and S. Bryant, "A Framework for Loop-Free
Convergence", RFC 5715, January 2010.

[RFC6976] Shand, M., Bryant, S., Previdi, S., Filsfils, C.,
Francois, P., and O. Bonaventure, "Framework for Loop-Free
Convergence Using the Ordered Forwarding Information Base
(oFIB) Approach", RFC 6976, July 2013.

Author's Address

Bruno Decraene
Orange
38 rue du General Leclerc
Issy Moulineaux cedex 9 92794
France

Email: bruno.decraene@orange.com

Routing Area Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 12, 2016

A. Atlas
Juniper Networks
K. Koushik
Cisco Systems
S. Litkowski
Orange
February 9, 2016

IP MIB for IP Fast-Reroute
draft-ietf-rtgwg-ipfrr-ip-mib-08

Abstract

This draft defines a portion of the Management Information Base (MIB) for use with network management protocols in the Internet community. In particular, it describes managed objects relevant for IP routes using IP Fast-Reroute [RFC5714]

Requirements Language

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

Status of This Memo

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

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

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

This Internet-Draft will expire on August 12, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
1.1. The SNMP Management Framework	3
2. Brief description of MIB Objects	3
2.1. ipFrrProtectStats Group	3
2.2. ipFrrInstanceTable	3
2.3. ipFrrIfTable	4
2.4. ipFrrProtectStatsTable	4
2.5. ipFrrAltTable	4
2.6. ipFrrNoAltTable	4
3. IP Fast-Reroute MIB Module Definitions	4
4. Security Considerations	25
5. Acknowledgements	26
6. IANA Considerations	26
7. References	26
7.1. Normative References	26
7.2. Informative References	27
Authors' Addresses	27

1. Introduction

This document defines a portion of the Management Information Base (MIB) for use with network management protocols in the Internet community. In particular, it defines the managed objects used for IP routes and interfaces in relation to IP Fast-Reroute. This document uses terminology from [RFC5714] and [RFC5286].

Current work is underway to define mechanisms for determining alternate paths for traffic to use when the original path becomes unavailable due to a local failure. The alternate next-hops can be computed in the context of any IGP.

There are certain configuration attributes for IP Fast-Reroute that should be configured to enable IP Fast Reroute in the context of the IGP. These configuration attributes of IP Fast-Reroute are not covered by this MIB module. Examples include whether IP Fast-Reroute is enabled on a network region (i.e. an OSPF area or IS-IS level) and

the desired local hold-down timer [RFC5286], whose proper value is dependent upon the size of the network region.

It is possible for traffic other than IP to depend upon and use the alternate next-hops computed by IP Fast-Reroute. An example would be MPLS traffic whose path is configured via LDP [RFC5036]. The additional details (for example, outgoing MPLS label) pertaining to alternate next-hops that are required by such traffic are not covered by this MIB module.

An IP route may be reachable via multiple primary next-hops which provide equal-cost paths. Where IP Fast-Reroute is enabled, each primary next-hop will be protected by one or more alternate next-hops. Such an alternate next-hop may itself be a primary next-hop.

1.1. The SNMP Management Framework

For a detailed overview of the documents that describe the current Internet-Standard Management Framework, please refer to section 7 of [RFC3410].

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. MIB objects are generally accessed through the Simple Network Management Protocol (SNMP). Objects in the MIB are defined using the mechanisms defined in the Structure of Management Information (SMI). This memo specifies a MIB module that is compliant to the SMIv2, which is described in STD 58, [RFC2578], STD 58, [RFC2579] and STD 58, [RFC2580].

2. Brief description of MIB Objects

2.1. ipFrrProtectStats Group

The global objects in this group provide summary information related to protection for all IP routes. The information available includes counts of all routes, of all protected routes, of all unprotected routes, of all routes which are protected against a link failure, and of all routes which are protected against a node failure.

2.2. ipFrrInstanceTable

The ipFrrInstanceTable provides information about configuration of IP FRR instantiations on a node. A single node may have multiple instances of IP FRR using different algorithms or protocols. ipFrrInstances cannot be created through the MIB.

2.3. ipFrrIfTable

The ipFrrIfTable provides information about configuration of interfaces for IPFRR. Entries can be created to activate IPFRR on a particular interface or setting the candidate properties.

2.4. ipFrrProtectStatsTable

The ipFrrProtectStatsTable complements the ipFrrProtectStats group by providing statistics per IP FRR instance.

2.5. ipFrrAltTable

The ipFrrAltTable extends the inetCidrRouteTable [RFC4292] to provide information about each alternate next-hop associated with a primary next-hop used by a route.

2.6. ipFrrNoAltTable

The ipFrrNoAltTable extends the inetCidrRouteTable [RFC4292] to provide information about the routes which do not have an alternate next-hop associated with any of the route's primary next-hop. The entry provides an explanation for the lack of protection.

3. IP Fast-Reroute MIB Module Definitions

IPFRR-MIB DEFINITIONS ::= BEGIN

IMPORTS

```
MODULE-IDENTITY,
OBJECT-TYPE,
Gauge32,
Integer32                FROM SNMPv2-SMI                -- [RFC2578]

RowStatus                FROM SNMPv2-TC                -- [RFC2579]

MODULE-COMPLIANCE,
OBJECT-GROUP              FROM SNMPv2-CONF              -- [RFC2580]

InetAddressType,
InetAddress              FROM INET-ADDRESS-MIB          -- [RFC4001]

ifIndex, InterfaceIndex  FROM IF-MIB                    -- [RFC2863]

ip                        FROM IP-MIB                    -- [RFC4293]
```

```
inetCidrRouteDestType,  
inetCidrRouteDest,  
inetCidrRoutePfxLen,  
inetCidrRoutePolicy,  
inetCidrRouteNextHopType,  
inetCidrRouteNextHop FROM IP-FORWARD-MIB  
-- [RFC4292]
```

```
IANAipRouteProtocol FROM IANA-RTPROTO-MIB
```

;

```
ipFrrMIB MODULE-IDENTITY  
LAST-UPDATED "201508040000Z" -- Aug 04, 2015  
ORGANIZATION "draft-ietf-ipfrr-ip-mib-06.txt"  
CONTACT-INFO
```

"

```
    A S Kiran Koushik  
    Cisco Systems Inc.  
    EMail: kkoushik@cisco.com
```

```
    Alia Atlas  
    Juniper Networks  
    Email: akatlas@juniper.net
```

```
    Stephane Litkowski  
    Orange Business Service  
    Email: stephane.litkowski@orange.com
```

"

DESCRIPTION

```
"IP MIB module for management of IP Fast-Reroute.
```

```
Copyright (C) The Internet Society (date).  
This version of this MIB module is part of  
draft-ietf-rtgwg-ipfrr-ip-mib-07.txt"
```

```
    REVISION      "201508040000Z" -- Aug 04, 2015  
DESCRIPTION
```

```
    "Fixing some syntax issues  
        Moved ipFrrInstanceTable to readonly  
        Moved ipFrrAltTable to readonly  
        Modified Readonly conformance
```

```
Deleting ipFrrInstanceRowStatus
Deleting ipFrrAltStatus
Added notProtect to ipFrrIfProtectionType
"
```

```
REVISION      "201406141200Z" -- Jun 14, 2014
DESCRIPTION
  "draft-ietf-rtgwg-ipfrr-ip-mib-03.txt"
```

```
REVISION      "201406131200Z" -- Jun 13, 2014
DESCRIPTION
  "Add ipFrrTunnelType in ipFrrAltEntry
  Modify ipFrrAltType"
```

```
REVISION      "201405261200Z" -- May 26, 2014
DESCRIPTION
  "Add ipFrrInstanceTable.
  Add ipFrrIfTable.
  ipFrrProtectStatsTable complements ipFrrProtectStats
  to have statistics per instance.
  Add ipFrrAltMetric2, ipFrrAltMetric3, ipFrrAltBest,
  ipFrrAltNonBestReason to ipFrrAltEntry.
  Add integer values to ipFrrAltType.
  Add integer values to ipFrrAltProtectionAvailable.
  Changed attachment of ipFrrAltStatus in ipFrrAltEntr
y.
  Added IPv6 objects in ipFrrProtectStats."
```

```
REVISION      "201203131200Z" -- Mar 13, 2012
DESCRIPTION
  "Editorial changes. Added new type to ipFrrAltType."
```

```
REVISION      "200502181200Z" -- February 18, 2005
DESCRIPTION
  "Add Set operations on ipFrrAltTable"
```

```
REVISION      "200502131200Z" -- February 13, 2005
DESCRIPTION
  "Initial version."
  ::= { ip 50 } -- To be assigned by IANA
```

-- Top level components of this MIB module.

```
ipFrrMIBObjects OBJECT IDENTIFIER ::= { ipFrrMIB 1 }
```

```
ipFrrProtectStats OBJECT IDENTIFIER ::= { ipFrrMIBObjects 1 }
```

-- the IP FRR MIB-Group

-- A collection of objects providing summarized information
-- about the protection availability and type of alternate paths
-- provided by IP Fast-Reroute mechanisms.

ipFrrTotalRoutes OBJECT-TYPE
SYNTAX Gauge32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of IPv4 valid routes known by this entity."
::= { ipFrrProtectStats 1 }

ipFrrUnprotectedRoutes OBJECT-TYPE
SYNTAX Gauge32
MAX-ACCESS read-only

STATUS current
DESCRIPTION
"The number of IPv4 valid routes known by this entity
which do not have an alternate next-hop associated
with any primary next-hop."
::= { ipFrrProtectStats 2 }

ipFrrProtectedRoutes OBJECT-TYPE
SYNTAX Gauge32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of IPv4 routes known by this entity
which have at least one alternate next-hop."
::= { ipFrrProtectStats 3 }

ipFrrLinkProtectedRoutes OBJECT-TYPE
SYNTAX Gauge32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of IPv4 routes known by this entity
for which all alternate next-hops provide link
protection for their associated primary next-hops."
::= { ipFrrProtectStats 4 }

ipFrrNodeProtectedRoutes OBJECT-TYPE
SYNTAX Gauge32
MAX-ACCESS read-only
STATUS current
DESCRIPTION


```
        "The number of IPv4 routes known by this entity
        for which all alternate next-hops provide node
        protection for their associated primary next-hops."
 ::= { ipFrrProtectStats 5 }

ipv6FrrTotalRoutes      OBJECT-TYPE
SYNTAX      Gauge32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of IPv6 valid routes known by this entity."
 ::= { ipFrrProtectStats 6 }

ipv6FrrUnprotectedRoutes OBJECT-TYPE
SYNTAX      Gauge32
MAX-ACCESS  read-only
STATUS      current

DESCRIPTION
    "The number of IPv6 valid routes known by this entity
    which do not have an alternate next-hop associated
    with any primary next-hop."
 ::= { ipFrrProtectStats 7 }

ipv6FrrProtectedRoutes OBJECT-TYPE
SYNTAX      Gauge32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of IPv6 routes known by this entity
    which have at least one alternate next-hop."
 ::= { ipFrrProtectStats 8 }

ipv6FrrLinkProtectedRoutes OBJECT-TYPE
SYNTAX      Gauge32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of IPv6 routes known by this entity
    for which all alternate next-hops provide link
    protection for their associated primary next-hops."
 ::= { ipFrrProtectStats 9 }

ipv6FrrNodeProtectedRoutes OBJECT-TYPE
SYNTAX      Gauge32
MAX-ACCESS  read-only
STATUS      current
```

```

DESCRIPTION
    "The number of IPv6 routes known by this entity
    for which all alternate next-hops provide node
    protection for their associated primary next-hops."
 ::= { ipFrrProtectStats 10 }

-- the IP FRR instance MIB-group
--
-- The ipFrrInstanceTable provides detail on current IPFRR
-- instances activated on the node

ipFrrInstanceTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF IpFrrInstanceEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "This entity's IP Fast Reroute Instance table."
    ::= { ipFrrMIBObjects 4 }

ipFrrInstanceEntry OBJECT-TYPE
    SYNTAX      IpFrrInstanceEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An entry containing information on a particular
        IP FRR instance on the node."

    INDEX { ipFrrInstanceId
    }
    ::= { ipFrrInstanceTable 1 }

    IpFrrInstanceEntry ::= SEQUENCE {
        ipFrrInstanceId          INTEGER,
                                IANAipRoutePro
        ipFrrInstanceProtocol    Integer32,
        ipFrrInstanceAlgorithm   INTEGER
        ipFrrInstancePerPrefixComputation
    }

ipFrrInstanceId OBJECT-TYPE
    SYNTAX      Integer32 (1..255)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "This object specifies an identifier a of particular IPFRR instance
        . "
    ::= { ipFrrInstanceEntry 1 }

```

ipFrrInstanceProtocol OBJECT-TYPE

SYNTAX IANAipRouteProtocol

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"This object specifies the protocol used by the IPFRR instance."

::= { ipFrrInstanceEntry 2 }

ipFrrInstanceAlgorithm OBJECT-TYPE

SYNTAX INTEGER {
loopFree(1),
loopFreeRemote(2),
loopFreeTI(3),
mrt(4)
}

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"This object specifies the algorithm used by the IPFRR instance."

::= { ipFrrInstanceEntry 3 }

ipFrrInstancePerPrefixComputation OBJECT-TYPE

SYNTAX INTEGER {
false(0),
true(1)
}

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"This object specifies if per prefix computation is used."

::= { ipFrrInstanceEntry 4 }

-- the IP FRR Interface MIB-Group

--

-- ipFrrIfTable provides information on configuration

-- of interfaces for IPFRR

ipFrrIfTable OBJECT-TYPE

SYNTAX SEQUENCE OF IpFrrIfEntry

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"This entity's IP Fast Reroute Alternates Interface configuration table."

```

 ::= { ipFrrMIBObjects 5 }

ipFrrIfEntry OBJECT-TYPE
    SYNTAX      IpFrrIfEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An entry containing information on a particular instance of an IPFRR
        interface."

    INDEX { ipFrrInstanceId,
            ifIndex
          }
 ::= { ipFrrIfTable 1 }

IpFrrIfEntry ::= SEQUENCE {
    ipFrrIfProtectionType      BITS,
    ipFrrIfCandidate           INTEGER,
    ipFrrIfRowStatus           RowStatus
}

ipFrrIfProtectionType OBJECT-TYPE
    SYNTAX      BITS {
        nodeProtect(0),
        linkProtect(1),
        nodelinkProtect(2),
        notProtect(3)
    }
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "This object specifies the scope of protection requested for the protection of the destinations.
        nodeProtect means node protection only compared to no delinkProtect which means node protection
        if available and link protection if not available. "
    ::= { ipFrrIfEntry 1 }

ipFrrIfCandidate OBJECT-TYPE
    SYNTAX      INTEGER {
        false (0),
        true (1)
    }
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "This object specifies the scope of protection requested for the protection of the destinations.
        nodeProtect means node protection only compared to no delinkProtect which means node protection
        if available and link protection if not available. "
    DEFVAL {1}

```

```

 ::= { ipFrrIfEntry 2 }

ipFrrIfRowStatus OBJECT-TYPE
    SYNTAX      RowStatus
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        ".
    ::= { ipFrrIfEntry 3 }

-- the IP FRR Stats MIB-Group
--
-- ipFrrProtectStatsTable provides provides
-- protection availability and type of alternate paths
-- provided by IP Fast-Reroute mechanisms per IPFRR instance.

ipFrrProtectStatsTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF IpFrrProtectStatsEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "This entity's IP Fast Reroute Alternates statistics table."
    ::= { ipFrrMIBObjects 6 }

ipFrrProtectStatsEntry OBJECT-TYPE
    SYNTAX      IpFrrProtectStatsEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An entry containing information on a particular instance of IPFRR.
        ."

    INDEX { ipFrrInstanceId }
    ::= { ipFrrProtectStatsTable 1 }

IpFrrProtectStatsEntry ::= SEQUENCE {
    ipFrrStatsTotalRoutes          Gauge32,
    ipFrrStatsUnprotectedRoutes    Gauge32,
    ipFrrStatsProtectedRoutes      Gauge32,
    ipFrrStatsLinkProtectedRoutes  Gauge32,
    ipFrrStatsNodeProtectedRoutes  Gauge32,
    ipv6FrrStatsTotalRoutes        Gauge32,
    ipv6FrrStatsUnprotectedRoutes  Gauge32,

```

```
        ipv6FrrStatsProtectedRoutes      Gauge32,
        ipv6FrrStatsLinkProtectedRoutes  Gauge32,
        ipv6FrrStatsNodeProtectedRoutes  Gauge32
    }

ipFrrStatsTotalRoutes      OBJECT-TYPE
    SYNTAX      Gauge32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of valid routes known by this entity."
    ::= { ipFrrProtectStatsEntry 1 }

ipFrrStatsUnprotectedRoutes      OBJECT-TYPE
    SYNTAX      Gauge32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of valid routes known by this entity

        which do not have an alternate next-hop associated
        with any primary next-hop."
    ::= { ipFrrProtectStatsEntry 2 }

ipFrrStatsProtectedRoutes      OBJECT-TYPE
    SYNTAX      Gauge32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of routes known by this entity
        which have at least one alternate next-hop."
    ::= { ipFrrProtectStatsEntry 3 }

ipFrrStatsLinkProtectedRoutes OBJECT-TYPE
    SYNTAX      Gauge32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of routes known by this entity
        for which all alternate next-hops provide link
        protection for their associated primary next-hops."
    ::= { ipFrrProtectStatsEntry 4 }

ipFrrStatsNodeProtectedRoutes OBJECT-TYPE
    SYNTAX      Gauge32
    MAX-ACCESS  read-only
    STATUS      current
```

DESCRIPTION

"The number of routes known by this entity
for which all alternate next-hops provide node
protection for their associated primary next-hops."

::= { ipFrrProtectStatsEntry 5 }

ipv6FrrStatsTotalRoutes OBJECT-TYPE

SYNTAX Gauge32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The number of valid IPv6 routes known by this entity."

::= { ipFrrProtectStatsEntry 6 }

ipv6FrrStatsUnprotectedRoutes OBJECT-TYPE

SYNTAX Gauge32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The number of valid IPv6 routes known by this entity
which do not have an alternate next-hop associated

with any primary next-hop."

::= { ipFrrProtectStatsEntry 7 }

ipv6FrrStatsProtectedRoutes OBJECT-TYPE

SYNTAX Gauge32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The number of IPv6 routes known by this entity
which have at least one alternate next-hop."

::= { ipFrrProtectStatsEntry 8 }

ipv6FrrStatsLinkProtectedRoutes OBJECT-TYPE

SYNTAX Gauge32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The number of IPv6 routes known by this entity
for which all alternate next-hops provide link
protection for their associated primary next-hops."

::= { ipFrrProtectStatsEntry 9 }

ipv6FrrStatsNodeProtectedRoutes OBJECT-TYPE

SYNTAX Gauge32

MAX-ACCESS read-only

```
STATUS      current
DESCRIPTION
    "The number of IPv6 routes known by this entity
    for which all alternate next-hops provide node
    protection for their associated primary next-hops."
 ::= { ipFrrProtectStatsEntry 10 }

-- the IP FRR Alternate MIB-Group
--
-- The ipFrrAltTable extends the inetCidrRouteTable to indicate
-- the alternate next-hop(s) associated with each primary
-- next-hop. The additional indices (ipFrrAltNextHopType and
-- ipFrrAltNextHop ) allow for multiple alternate paths for a
-- given primary next-hop.

ipFrrAltTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF IpFrrAltEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "This entity's IP Fast Reroute Alternates table."
    ::= { ipFrrMIBObjects 2 }

ipFrrAltEntry OBJECT-TYPE
    SYNTAX      IpFrrAltEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An entry containing information on a particular route,
        one of its particular (primary) next-hops and one of
        the associated alternate next-hops.

        Implementers need to be aware that if the total
        number of elements (octets or sub-identifiers) in
        inetCidrRouteDest, inetCidrRoutePolicy,
        inetCidrRouteNextHop, and ipFrrAltNextHop exceeds 107
        then OIDs of column instances in this table will have
        more than 128 sub-identifiers and cannot be accessed
        using SNMPv1, SNMPv2c, or SNMPv3."

    INDEX { inetCidrRouteDestType,
            inetCidrRouteDest,
            inetCidrRoutePfxLen,
            inetCidrRoutePolicy,
            inetCidrRouteNextHopType,
            inetCidrRouteNextHop,
```



```

        ipFrrAltNextHopType,
        ipFrrAltNextHop
    }
    ::= { ipFrrAltTable 1 }

IpFrrAltEntry ::= SEQUENCE {
    ipFrrAltNextHopType      InetAddressType,
    ipFrrAltNextHop          InetAddress,
    ipFrrAltIfIndex          InterfaceIndex,
    ipFrrAltType              INTEGER,
    ipFrrTunnelType           INTEGER,
    ipFrrAltProtectionAvailable BITS,
    ipFrrAltMetric1           Integer32,
    ipFrrAltMetric2           Integer32,
    ipFrrAltMetric3           Integer32,
    ipFrrAltBest              INTEGER,
    ipFrrAltNonBestReason     OCTET STRING
}

ipFrrAltNextHopType OBJECT-TYPE
    SYNTAX      InetAddressType
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION

        "The type of the ipFrrNextHop address, as defined
        in the InetAddress MIB.

        Only those address types that may appear in an actual
        routing table are allowed as values of this object."
    REFERENCE "RFC 4001"
    ::= { ipFrrAltEntry 1 }

ipFrrAltNextHop OBJECT-TYPE
    SYNTAX      InetAddress
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The address of the next system along the alternate
        route.

        The type of this address is determined by the value
        of the ipFrrAltNextHopType."
    ::= { ipFrrAltEntry 2 }

ipFrrAltIfIndex OBJECT-TYPE
    SYNTAX      InterfaceIndex

```

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The ifIndex value which identifies the local interface through which the next hop of this alternate route should be reached."

::= { ipFrrAltEntry 3 }

ipFrrAltType OBJECT-TYPE

SYNTAX INTEGER {

other

(1), -- type not defined

equalCost

(2), -- primary path

loopFree

(3), -- loop free alternate

loopFreeRemote

(4), -- remote loop free alternate

loopFreeNH

(5), -- loop free alternate using a co

nfigured tunnel toward the nexthop

loopFreeNNH

(6), -- loop free alternate using a co

nfigured tunnel toward the nextnexthop

loopFreeTI

(7), -- loop free alternate using topo

logy independent algorithm

mrt

(8) -- Maximally Redundant Trees

}

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The type of alternate which is provided by the alternate next-hop. The supported types are as follows:

equalCost : The alternate next-hop is another primary next-hop.

loopFreeConnected : loop free alternate (LFA as described in RFC5286

)

loopFreeRemote : remote LFA (as described in draft-ietf-rtgwg-remote-lfa)

loopFreeNH : loop free alternate using a configured tunnel toward the nexthop (link protection only)

loopFreeNNH : loop free alternate using a configured tunnel toward the nextnexthop (node protection)

loopFreeTI : loop free alternate using topology independent algorithm

other : The mechanism by which the alternate next-hop can be used is not specified.

MRT : Maximally Redundant Trees, where each destination has two MRTs associated with it. These two trees are referred as blue and red MRTs.

See draft-ietf-rtgwg-mrt-frr-architecture-00.

```

"
 ::= { ipFrrAltEntry 4 }

ipFrrTunnelType OBJECT-TYPE
    SYNTAX      INTEGER {
        none          (1), -- No tunnel used
        other         (2), -- type not defined
        ldp           (3), -- LDP tunnel
        ip            (4), -- IP based tunnel (GRE, IPIP, L2TP .
        ..)
        srmppls       (5), -- SPRING tunnel using MPLS dataplane
        sripv6        (6), -- SPRING tunnel using IPv6 dataplane
        rsvpte        (7), -- RSVP-TE tunnel
        mtldp         (8)  -- LDP tunnel on another topology
    }
    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
        "The type of tunnel used to reach the alternate.
        The supported types are as follows:

        none : No tunnel used

        ldp : use LDP tunnel to reach the alternate (typicall
y the case of rLFA)

        ip : use IP based tunnel to reach the alternate

        srmppls or sripv6 : use SPRING based tunnel (typically
the case of TI-LFA)

        rsvpte : use a RSVP-TE LSP to reach the alternate

        mtldp : use an LDP tunnel based on another topology (
typically the case of MRT)

```

```

"
 ::= { ipFrrAltEntry 5 }

ipFrrAltProtectionAvailable OBJECT-TYPE
    SYNTAX      BITS {
        nodeProtect(0),
        linkProtect(1),
        srlgProtect(2),
        downstreamProtect(3),
        unknownProtection(4)
    }
    MAX-ACCESS read-only
    STATUS      current

```

DESCRIPTION

"This object specifies the scope of protection for which this alternate next-hop can provide failure protection. The alternate next-hop should provide one or more of node-protection and link-protection. If the protection provided by the alternate next-hop is unknown, then only unknownProtection should be specified. Specifying unknownProtection with any other type of protection is not supported. "

::= { ipFrrAltEntry 6 }

ipFrrAltMetric1 OBJECT-TYPE

SYNTAX Integer32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"This is the primary routing metric for this alternate path to the destination IP address. If the alternate path metric is unknown, the value should be set to -1."

::= { ipFrrAltEntry 7 }

ipFrrAltMetric2 OBJECT-TYPE

SYNTAX Integer32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"This is the primary routing metric for this alternate path from the PLR to the alternate. If the alternate path metric is unknown, the value

should be set to -1."

::= { ipFrrAltEntry 8 }

ipFrrAltMetric3 OBJECT-TYPE

SYNTAX Integer32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"This is the primary routing metric for this alternate path from the alternate to the destination. If the alternate path metric is unknown, the value should be set to -1."

::= { ipFrrAltEntry 9 }

ipFrrAltBest OBJECT-TYPE

SYNTAX INTEGER { false(0), true(1) }

```
MAX-ACCESS read-only
STATUS      current
DESCRIPTION
    "This object provides information if the alternate is the best one."
 ::= { ipFrrAltEntry 10 }

ipFrrAltNonBestReason OBJECT-TYPE
SYNTAX      OCTET STRING (SIZE (0..255))
MAX-ACCESS read-only
STATUS      current
DESCRIPTION
    "This object provides reason why an alternate is not the best one."
 ::= { ipFrrAltEntry 11 }

-- the IP FRR No Alternate MIB-Group
--
-- The ipFrrNoAltTable extends the inetCidrRouteTable
-- to indicate which routes are unprotected and the reason
-- why. The indices do not include the primary next-hop because
-- the lack of protection is for the route. This allows easy
-- access to the set of unprotected routes that would be
-- affected by a local failure of their primary next-hop.

ipFrrNoAltTable OBJECT-TYPE
SYNTAX      SEQUENCE OF IpFrrNoAltEntry
MAX-ACCESS not-accessible
STATUS      current
DESCRIPTION
    "This entity's IP Fast Reroute Unprotected Routes
    table."
 ::= { ipFrrMIBObjects 3 }

ipFrrNoAltEntry OBJECT-TYPE
SYNTAX      IpFrrNoAltEntry
MAX-ACCESS not-accessible
STATUS      current
DESCRIPTION
    "An entry containing the reason why a route does not
    have an alternate next-hop. The existence of an
    entry for a route indicates that there is no
    alternate next-hop."
INDEX { inetCidrRouteDestType,
        inetCidrRouteDest,
        inetCidrRoutePfxLen
```

```
    }
    ::= { ipFrrNoAltTable 1 }

IpFrrNoAltEntry ::= SEQUENCE {
    ipFrrNoAltCause      INTEGER
}

ipFrrNoAltCause OBJECT-TYPE
    SYNTAX      INTEGER {
        ipFrrUnavailable (1), -- No valid alternate(s)
        localAddress     (2), -- local/internal address
        ipFrrDisabled    (3), -- Protection not enabled
        other             (4)  -- unknown or other cause
    }
    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
        "For valid routes without an alternate next-hop, this
        object enumerates the reason why no protection is
        available.  The possibilities are as follows.

        ipFrrUnavailable : The supported IP Fast-Reroute
                           mechanisms could not find a safe
                           alternate next-hop.

        localAddress : The route represents a local address.
                       This system is the destination so no

                           alternate path is possible or necessary.

        ipFrrDisabled : Finding of alternate next-hops is
                       operationally disabled.

        other : The reason is unknown or different from those
                specifically enumerated possible causes."
    ::= { ipFrrNoAltEntry 1 }

-- conformance information

ipFrrMIBConformance
    OBJECT IDENTIFIER ::= { ipFrrMIB 2 }

ipFrrMIBCompliances
    OBJECT IDENTIFIER ::= { ipFrrMIBConformance 1 }

ipFrrMIBGroups
    OBJECT IDENTIFIER ::= { ipFrrMIBConformance 2 }
```

-- compliance statements

ipFrrMIBCompliance MODULE-COMPLIANCE

STATUS deprecated

DESCRIPTION

"Minimum requirements to state conformity
to this MIB. Supporting only IP v4 addresses
This is deprecated in favor of
ipFrrMIBInetCompliance

There are a number of INDEX objects that cannot be
represented in the form of OBJECT clauses in SMIV2,
but for which there are compliance requirements,
expressed in OBJECT clause form in this description:

OBJECT inetCidrRouteDestType

SYNTAX InetAddressType { ipv4(1), ipv4z(3) }

MIN-ACCESS read-only

DESCRIPTION

A (deprecated) complying implementation at this
level is required to support IPv4 addresses only.
This compliance level is defined so an
implementation only needs to support the addresses
it actually supports on the device.

OBJECT inetCidrRouteNextHopType

SYNTAX InetAddressType { ipv4(1), ipv4z(3) }

MIN-ACCESS read-only

DESCRIPTION

A (deprecated) complying implementation at this
level is required to support IPv4 addresses only.
This compliance level is defined so an
implementation only needs to support the addresses
it actually supports on the device.

OBJECT ipFrrAltNextHopType

SYNTAX InetAddressType { ipv4(1), ipv4z(3) }

MIN-ACCESS read-only

DESCRIPTION

A (deprecated) complying implementation at this
level is required to support IPv4 addresses only.
This compliance level is defined so an
implementation only needs to support the
addresses it actually supports on the device.

"

MODULE -- this module

```
MANDATORY-GROUPS { ipFrrBasicGroup }

 ::= { ipFrrMIBCompliances 1 }

ipFrrMIBInetCompliance MODULE-COMPLIANCE
    STATUS current
    DESCRIPTION
        "Full conformity to this MIB."
    MODULE -- this module
    MANDATORY-GROUPS { ipFrrBasicGroup }

    OBJECT ipFrrIfRowStatus
    SYNTAX INTEGER { active(1) }
    WRITE-SYNTAX INTEGER { createAndGo(4), destroy(6) }
    DESCRIPTION
        "Support for createAndWait and notInService is not
        required."

 ::= { ipFrrMIBCompliances 2 }

ipFrrReadOnlyCompliance MODULE-COMPLIANCE
    STATUS current
    DESCRIPTION

        "When this MIB is implemented without support for
        read-create (i.e. in read-only mode), then that
        implementation can claim read-only compliance. In that
        case, ipFrrAlt group can be monitored but cannot be
        configured with this MIB."

    MODULE
    MANDATORY-GROUPS { ipFrrBasicGroup }

    OBJECT ipFrrIfProtectionType
    MIN-ACCESS read-only
    DESCRIPTION
        "Write access is not required."

    OBJECT ipFrrIfCandidate
    MIN-ACCESS read-only
    DESCRIPTION
        "Write access is not required."

    OBJECT ipFrrIfRowStatus
    MIN-ACCESS read-only
    DESCRIPTION
```



```
        "Write access is not required."

 ::= { ipFrrMIBCompliances 3 }

-- units of conformance
ipFrrBasicGroup OBJECT-GROUP
    OBJECTS {ipFrrTotalRoutes,
              ipFrrUnprotectedRoutes,
              ipFrrProtectedRoutes,
              ipFrrLinkProtectedRoutes,
              ipFrrNodeProtectedRoutes,
              ipv6FrrTotalRoutes,
              ipv6FrrUnprotectedRoutes,
              ipv6FrrProtectedRoutes,
              ipv6FrrLinkProtectedRoutes,
              ipv6FrrNodeProtectedRoutes,
              ipFrrAltIfIndex,
              ipFrrAltType,
              ipFrrTunnelType,
              ipFrrAltProtectionAvailable,
              ipFrrAltMetric1,
              ipFrrAltMetric2,
              ipFrrAltMetric3,
              ipFrrAltNonBestReason,
              ipFrrAltBest,
              ipFrrNoAltCause,
              ipFrrInstanceAlgorithm,
              ipFrrInstanceProtocol,
              ipFrrInstancePerPrefixComputation,
              ipFrrIfCandidate,
              ipFrrIfProtectionType,
              ipFrrIfRowStatus,
              ipFrrStatsTotalRoutes,
              ipFrrStatsUnprotectedRoutes,
              ipFrrStatsProtectedRoutes,
              ipFrrStatsLinkProtectedRoutes,
              ipFrrStatsNodeProtectedRoutes,
              ipv6FrrStatsTotalRoutes,
              ipv6FrrStatsUnprotectedRoutes,
              ipv6FrrStatsProtectedRoutes,
              ipv6FrrStatsLinkProtectedRoutes,
              ipv6FrrStatsNodeProtectedRoutes
    }
    STATUS current
    DESCRIPTION
        "The entire collection of objects defined in
         this MIB for management of IP Fast Reroute ."
```

```
::= { ipFrrMIBGroups 1 }
```

END

4. Security Considerations

There are a number of management objects defined in this MIB module with a MAX-ACCESS clause of read-write and/or read-create. Such objects may be considered sensitive or vulnerable in some network environments. The support for SET operations in a non-secure environment without proper protection can have a negative effect on network operations. The ipFrrAltTable contains routing and forwarding information that is critical to the operation of the network in the event of a local failure. Allowing unauthenticated write access to this table can compromise the validity of the alternate forwarding information.

Some of the readable objects in this MIB module (i.e. objects with a MAX-ACCESS other than not-accessible) may be considered sensitive or vulnerable in some network environments. It is thus important to control even GET access to these objects and possibly to even encrypt the values of these objects when sending them over the network via SNMP.

SNMP versions prior to SNMPv3 did not include adequate security. Even if the network itself is secure (for example by using IPsec), even then, there is no control as to who on the secure network is allowed to access and GET the objects in this MIB module.

It is RECOMMENDED that implementers consider the security features as provided by the SNMPv3 framework (see [RFC3410], section 8), including full support for the SNMPv3 cryptographic mechanisms (for authentication and privacy).

Further, deployment of SNMP versions prior to SNMPv3 is NOT RECOMMENDED. Instead, it is RECOMMENDED to deploy SNMPv3 and to enable cryptographic security. It is then a customer/operator responsibility to ensure that the SNMP entity giving access to an instance of this MIB module is properly configured to give access to the objects only to those principals (users) that have legitimate rights to indeed GET them.

5. Acknowledgements

The authors would like to acknowledge contributions made by Bill Anderson, Don Fedyk, John Flick and Bruno Decraene.

6. IANA Considerations

The MIB module in this document uses the following IANA-assigned OBJECT IDENTIFIER value recorded in the SMI Numbers registry.

The IANA is requested to assign { ip ZZZ } to the IPFRR-MIB MIB module specified in this document.

Editor's Note (to be removed prior to publication): the IANA is requested to assign a value for "ZZZ" under the ip subtree and to record the assignments in the SMI Numbers registry. When the assignments have been made, the RFC Editor is asked to replace "ZZZ" (here and in the MIB modules) with the assigned value and to remove this note.

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2863] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB", RFC 2863, DOI 10.17487/RFC2863, June 2000, <<http://www.rfc-editor.org/info/rfc2863>>.
- [RFC4001] Daniele, M., Haberman, B., Routhier, S., and J. Schoenwaelder, "Textual Conventions for Internet Network Addresses", RFC 4001, DOI 10.17487/RFC4001, February 2005, <<http://www.rfc-editor.org/info/rfc4001>>.
- [RFC4292] Haberman, B., "IP Forwarding Table MIB", RFC 4292, DOI 10.17487/RFC4292, April 2006, <<http://www.rfc-editor.org/info/rfc4292>>.
- [RFC4293] Routhier, S., Ed., "Management Information Base for the Internet Protocol (IP)", RFC 4293, DOI 10.17487/RFC4293, April 2006, <<http://www.rfc-editor.org/info/rfc4293>>.

- [RFC5286] Atlas, A., Ed. and A. Zinin, Ed., "Basic Specification for IP Fast Reroute: Loop-Free Alternates", RFC 5286, DOI 10.17487/RFC5286, September 2008, <<http://www.rfc-editor.org/info/rfc5286>>.

7.2. Informative References

- [RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, DOI 10.17487/RFC2578, April 1999, <<http://www.rfc-editor.org/info/rfc2578>>.
- [RFC2579] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Textual Conventions for SMIv2", STD 58, RFC 2579, DOI 10.17487/RFC2579, April 1999, <<http://www.rfc-editor.org/info/rfc2579>>.
- [RFC2580] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Conformance Statements for SMIv2", STD 58, RFC 2580, DOI 10.17487/RFC2580, April 1999, <<http://www.rfc-editor.org/info/rfc2580>>.
- [RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, DOI 10.17487/RFC3410, December 2002, <<http://www.rfc-editor.org/info/rfc3410>>.
- [RFC5036] Andersson, L., Ed., Minei, I., Ed., and B. Thomas, Ed., "LDP Specification", RFC 5036, DOI 10.17487/RFC5036, October 2007, <<http://www.rfc-editor.org/info/rfc5036>>.
- [RFC5714] Shand, M. and S. Bryant, "IP Fast Reroute Framework", RFC 5714, DOI 10.17487/RFC5714, January 2010, <<http://www.rfc-editor.org/info/rfc5714>>.

Authors' Addresses

Alia Atlas
Juniper Networks

Email: akatlas@juniper.net

A S Kiran Koushik
Cisco Systems

Email: kkoushik@cisco.com

Stephane Litkowski
Orange

Email: stephane.litkowski@orange.com

Routing Area Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 19, 2016

G. Enyedi
A. Csaszar
Ericsson
A. Atlas
C. Bowers
Juniper Networks
A. Gopalan
University of Arizona
February 16, 2016

An Algorithm for Computing Maximally Redundant Trees for IP/LDP Fast-
Reroute
draft-ietf-rtgwg-mrt-frr-algorithm-09

Abstract

A solution for IP and LDP Fast-Reroute using Maximally Redundant Trees is presented in draft-ietf-rtgwg-mrt-frr-architecture. This document defines the associated MRT Lowpoint algorithm that is used in the Default MRT profile to compute both the necessary Maximally Redundant Trees with their associated next-hops and the alternates to select for MRT-FRR.

Status of This Memo

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

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

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

This Internet-Draft will expire on August 19, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	3
2. Requirements Language	5
3. Terminology and Definitions	5
4. Algorithm Key Concepts	6
4.1. Partial Ordering for Disjoint Paths	7
4.2. Finding an Ear and the Correct Direction	8
4.3. Low-Point Values and Their Uses	11
4.4. Blocks in a Graph	14
4.5. Determining Local-Root and Assigning Block-ID	16
5. MRT Lowpoint Algorithm Specification	18
5.1. Interface Ordering	18
5.2. MRT Island Identification	21
5.3. GADAG Root Selection	21
5.4. Initialization	22
5.5. Constructing the GADAG using lowpoint inheritance	23
5.6. Augmenting the GADAG by directing all links	25
5.7. Compute MRT next-hops	29
5.7.1. MRT next-hops to all nodes ordered with respect to the computing node	29
5.7.2. MRT next-hops to all nodes not ordered with respect to the computing node	30
5.7.3. Computing Redundant Tree next-hops in a 2-connected Graph	31
5.7.4. Generalizing for a graph that isn't 2-connected	33
5.7.5. Complete Algorithm to Compute MRT Next-Hops	34
5.8. Identify MRT alternates	36
5.9. Named Proxy-Nodes	43
5.9.1. Determining Proxy-Node Attachment Routers	43
5.9.2. Computing if an Island Neighbor (IN) is loop-free	44
5.9.3. Computing MRT Next-Hops for Proxy-Nodes	46
5.9.4. Computing MRT Alternates for Proxy-Nodes	52
6. MRT Lowpoint Algorithm: Next-hop conformance	60
7. Broadcast interfaces	60
7.1. Computing MRT next-hops on broadcast networks	61
7.2. Using MRT next-hops as alternates in the event of failures on broadcast networks	62
8. Evaluation of Alternative Methods for Constructing GADAGs	63
9. Implementation Status	64

10. Operational Considerations	65
10.1. GADAG Root Selection	65
10.2. Destination-rooted GADAGs	65
11. Acknowledgements	66
12. IANA Considerations	66
13. Security Considerations	66
14. References	66
14.1. Normative References	66
14.2. Informative References	66
Appendix A. Python Implementation of MRT Lowpoint Algorithm . .	67
Appendix B. Constructing a GADAG using SPFs	108
Appendix C. Constructing a GADAG using a hybrid method	113
Authors' Addresses	115

1. Introduction

MRT Fast-Reroute requires that packets can be forwarded not only on the shortest-path tree, but also on two Maximally Redundant Trees (MRTs), referred to as the MRT-Blue and the MRT-Red. A router which experiences a local failure must also have pre-determined which alternate to use. This document defines how to compute these three things for use in MRT-FRR and describes the algorithm design decisions and rationale. The algorithm is based on those presented in [MRTLinear] and expanded in [EnyediThesis]. The MRT Lowpoint algorithm is required for implementation when the Default MRT profile is implemented.

Just as packets routed on a hop-by-hop basis require that each router compute a shortest-path tree which is consistent, it is necessary for each router to compute the MRT-Blue next-hops and MRT-Red next-hops in a consistent fashion. This document defines the MRT Lowpoint algorithm to be used as a standard in the default MRT profile for MRT-FRR.

As now, a router's FIB will contain primary next-hops for the current shortest-path tree for forwarding traffic. In addition, a router's FIB will contain primary next-hops for the MRT-Blue for forwarding received traffic on the MRT-Blue and primary next-hops for the MRT-Red for forwarding received traffic on the MRT-Red.

What alternate next-hops a point-of-local-repair (PLR) selects need not be consistent - but loops must be prevented. To reduce congestion, it is possible for multiple alternate next-hops to be selected; in the context of MRT alternates, each of those alternate next-hops would be equal-cost paths.

This document defines an algorithm for selecting an appropriate MRT alternate for consideration. Other alternates, e.g. LFAs that are

downstream paths, may be preferred when available. See the Operational Considerations section of [I-D.ietf-rtgwg-mrt-frr-architecture] for a more detailed discussion of combining MRT alternates with those produced by other FRR technologies.

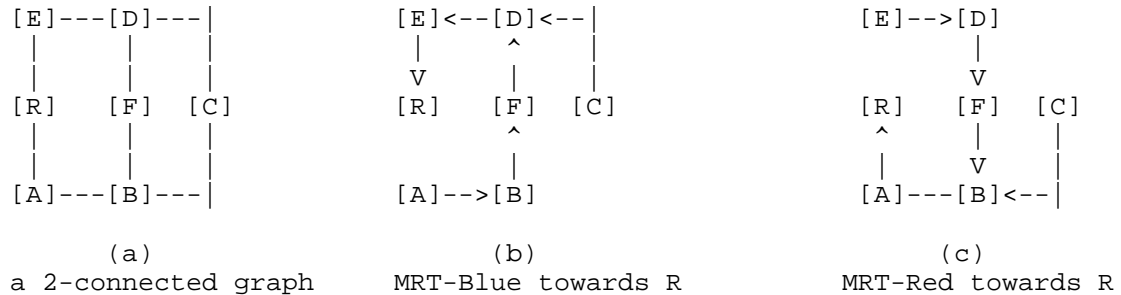
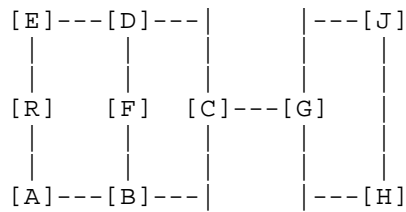
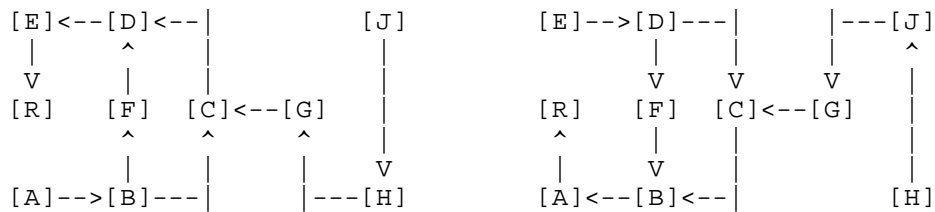


Figure 1

The MRT Lowpoint algorithm can handle arbitrary network topologies where the whole network graph is not 2-connected, as in Figure 2, as well as the easier case where the network graph is 2-connected (Figure 1). Each MRT is a spanning tree. The pair of MRTs provide two paths from every node X to the root of the MRTs. Those paths share the minimum number of nodes and the minimum number of links. Each such shared node is a cut-vertex. Any shared links are cut-links.



(a) a graph that isn't 2-connected



(b) MRT-Blue towards R

(c) MRT-Red towards R

Figure 2

2. Requirements Language

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

3. Terminology and Definitions

Please see the Terminology section of [I-D.ietf-rtgwg-mrt-frr-architecture] for a complete list of terminology relevant to this draft. The list below does not repeat terminology introduced in that draft.

spanning tree: A tree containing links that connects all nodes in the network graph.

back-edge: In the context of a spanning tree computed via a depth-first search, a back-edge is a link that connects a descendant of a node *x* with an ancestor of *x*.

partial ADAG: A subset of an ADAG that doesn't yet contain all the nodes in the block. A partial ADAG is created during the MRT algorithm and then expanded until all nodes in the block are included and it is an ADAG.

DFS: Depth-First Search

DFS ancestor: A node *n* is a DFS ancestor of *x* if *n* is on the DFS-tree path from the DFS root to *x*.

DFS descendant: A node *n* is a DFS descendant of *x* if *x* is on the DFS-tree path from the DFS root to *n*.

ear: A path along not-yet-included-in-the-GADAG nodes that starts at a node that is already-included-in-the-GADAG and that ends at a node that is already-included-in-the-GADAG. The starting and ending nodes may be the same node if it is a cut-vertex.

X >> Y or Y << X: Indicates the relationship between *X* and *Y* in a partial order, such as found in a GADAG. *X >> Y* means that *X* is higher in the partial order than *Y*. *Y << X* means that *Y* is lower in the partial order than *X*.

X > Y or Y < X: Indicates the relationship between *X* and *Y* in the total order, such as found via a topological sort. *X > Y* means that *X* is higher in the total order than *Y*. *Y < X* means that *Y* is lower in the total order than *X*.

X ?? Y: Indicates that *X* is unordered with respect to *Y* in the partial order.

UNDIRECTED: In the GADAG, each link is marked as OUTGOING, INCOMING or both. Until the directionality of the link is determined, the link is marked as UNDIRECTED to indicate that its direction hasn't been determined.

OUTGOING: A link marked as OUTGOING has direction in the GADAG from the interface's router to the remote end.

INCOMING: A link marked as INCOMING has direction in the GADAG from the remote end to the interface's router.

4. Algorithm Key Concepts

There are five key concepts that are critical for understanding the MRT Lowpoint algorithm. The first is the idea of partially ordering the nodes in a network graph with regard to each other and to the GADAG root. The second is the idea of finding an ear of nodes and adding them in the correct direction. The third is the idea of a Low-Point value and how it can be used to identify cut-vertices and to find a second path towards the root. The fourth is the idea that a non-2-connected graph is made up of blocks, where a block is a 2-connected cluster, a cut-link or an isolated node. The fifth is the idea of a local-root for each node; this is used to compute ADAGs in each block.

4.1. Partial Ordering for Disjoint Paths

Given any two nodes X and Y in a graph, a particular total order means that either $X < Y$ or $X > Y$ in that total order. An example would be a graph where the nodes are ranked based upon their unique IP loopback addresses. In a partial order, there may be some nodes for which it can't be determined whether $X \ll Y$ or $X \gg Y$. A partial order can be captured in a directed graph, as shown in Figure 3. In a graphical representation, a link directed from X to Y indicates that X is a neighbor of Y in the network graph and $X \ll Y$.

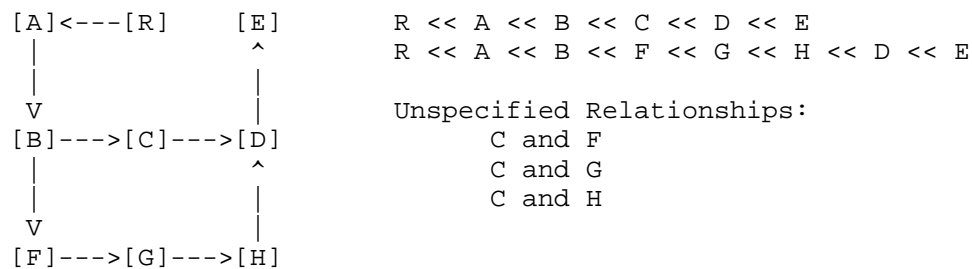


Figure 3: Directed Graph showing a Partial Order

To compute MRTs, the root of the MRTs is at both the very bottom and the very top of the partial ordering. This means that from any node X, one can pick nodes higher in the order until the root is reached. Similarly, from any node X, one can pick nodes lower in the order until the root is reached. For instance, in Figure 4, from G the higher nodes picked can be traced by following the directed links and are H, D, E and R. Similarly, from G the lower nodes picked can be traced by reversing the directed links and are F, B, A, and R. A graph that represents this modified partial order is no longer a DAG; it is termed an Almost DAG (ADAG) because if the links directed to the root were removed, it would be a DAG.

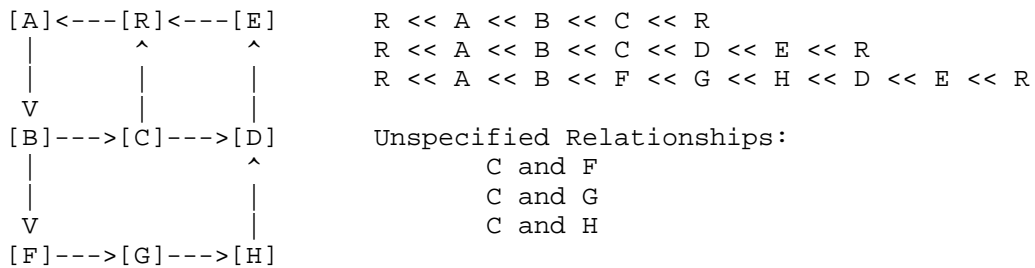


Figure 4: ADAG showing a Partial Order with R lowest and highest

Most importantly, if a node $Y \gg X$, then Y can only appear on the increasing path from X to the root and never on the decreasing path. Similarly, if a node $Z \ll X$, then Z can only appear on the decreasing path from X to the root and never on the increasing path.

When following the increasing paths, it is possible to pick multiple higher nodes and still have the certainty that those paths will be disjoint from the decreasing paths. E.g. in the previous example node B has multiple possibilities to forward packets along an increasing path: it can either forward packets to C or F .

4.2. Finding an Ear and the Correct Direction

For simplicity, the basic idea of creating a GADAG by adding ears is described assuming that the network graph is a single 2-connected cluster so that an ADAG is sufficient. Generalizing to multiple blocks is done by considering the block-roots instead of the GADAG root - and the actual algorithm is given in Section 5.5.

In order to understand the basic idea of finding an ADAG, first suppose that we have already a partial ADAG, which doesn't contain all the nodes in the block yet, and we want to extend it to cover all the nodes. Suppose that we find a path from a node X to Y such that X and Y are already contained by our partial ADAG, but all the remaining nodes along the path are not added to the ADAG yet. We refer to such a path as an ear.

Recall that our ADAG is closely related to a partial order. More precisely, if we remove root R , the remaining DAG describes a partial order of the nodes. If we suppose that neither X nor Y is the root, we may be able to compare them. If one of them is definitely lesser with respect to our partial order (say $X \ll Y$), we can add the new path to the ADAG in a direction from X to Y . As an example consider Figure 5.

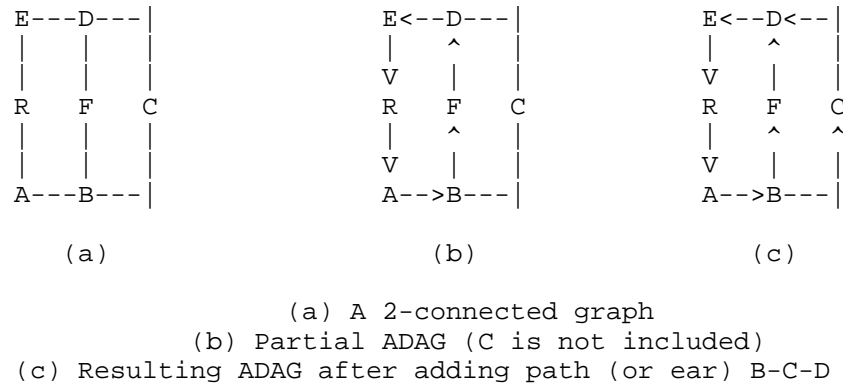


Figure 5

In this partial ADAG, node C is not yet included. However, we can find path B-C-D, where both endpoints are contained by this partial ADAG (we say those nodes are "ready" in the following text), and the remaining node (node C) is not contained yet. If we remove R, the remaining DAG defines a partial order, and with respect to this partial order we can say that $B \ll D$, so we can add the path to the ADAG in the direction from B to D (arcs B->C and C->D are added). If $B \gg D$, we would add the same path in reverse direction.

If in the partial order where an ear's two ends are X and Y, $X \ll Y$, then there must already be a directed path from X to Y in the ADAG. The ear must be added in a direction such that it doesn't create a cycle; therefore the ear must go from X to Y.

In the case, when X and Y are not ordered with each other, we can select either direction for the ear. We have no restriction since neither of the directions can result in a cycle. In the corner case when one of the endpoints of an ear, say X, is the root (recall that the two endpoints must be different), we could use both directions again for the ear because the root can be considered both as smaller and as greater than Y. However, we strictly pick that direction in which the root is lower than Y. The logic for this decision is explained in Section 5.7

A partial ADAG is started by finding a cycle from the root R back to itself. This can be done by selecting a non-ready neighbor N of R and then finding a path from N to R that doesn't use any links between R and N. The direction of the cycle can be assigned either way since it is starting the ordering.

Once a partial ADAG is already present, it will always have a node that is not the root R in it. As a brief proof that a partial ADAG

can always have ears added to it: just select a non-ready neighbor N of a ready node Q , such that Q is not the root R , find a path from N to the root R in the graph with Q removed. This path is an ear where the first node of the ear is Q , the next is N , then the path until the first ready node the path reached (that ready node is the other endpoint of the path). Since the graph is 2-connected, there must be a path from N to R without Q .

It is always possible to select a non-ready neighbor N of a ready node Q so that Q is not the root R . Because the network is 2-connected, N must be connected to two different nodes and only one can be R . Because the initial cycle has already been added to the ADAG, there are ready nodes that are not R . Since the graph is 2-connected, while there are non-ready nodes, there must be a non-ready neighbor N of a ready node that is not R .

```
Generic_Find_Ears_ADAG(root)
  Create an empty ADAG. Add root to the ADAG.
  Mark root as IN_GADAG.
  Select an arbitrary cycle containing root.
  Add the arbitrary cycle to the ADAG.
  Mark cycle's nodes as IN_GADAG.
  Add cycle's non-root nodes to process_list.
  while there exists connected nodes in graph that are not IN_GADAG
    Select a new ear. Let its endpoints be  $X$  and  $Y$ .
    if  $Y$  is root or ( $Y < X$ )
      add the ear towards  $X$  to the ADAG
    else // (a)  $X$  is root or (b)  $X < Y$  or (c)  $X, Y$  not ordered
      Add the ear towards  $Y$  to the ADAG
```

Figure 6: Generic Algorithm to find ears and their direction in 2-connected graph

The algorithm in Figure 6 merely requires that a cycle or ear be selected without specifying how. Regardless of the method for selecting the path, we will get an ADAG. The method used for finding and selecting the ears is important; shorter ears result in shorter paths along the MRTs. The MRT Lowpoint algorithm uses the Low-Point Inheritance method for constructing an ADAG (and ultimately a GADAG). This method is defined in Section 5.5. Other methods for constructing GADAGs are described in Appendix B and Appendix C. An evaluation of these different methods is given in Section 8

As an example, consider Figure 5 again. First, we select the shortest cycle containing R , which can be $R-A-B-F-D-E$ (uniform link costs were assumed), so we get to the situation depicted in Figure 5 (b). Finally, we find a node next to a ready node; that must be node C and assume we reached it from ready node B . We search a path from

C to R without B in the original graph. The first ready node along this is node D, so the open ear is B-C-D. Since $B < D$, we add arc B->C and C->D to the ADAG. Since all the nodes are ready, we stop at this point.

4.3. Low-Point Values and Their Uses

A basic way of computing a spanning tree on a network graph is to run a depth-first-search, such as given in Figure 7. This tree has the important property that if there is a link (x, n), then either n is a DFS ancestor of x or n is a DFS descendant of x. In other words, either n is on the path from the root to x or x is on the path from the root to n.

```

global_variable: dfs_number

DFS_Visit(node x, node parent)
    D(x) = dfs_number
    dfs_number += 1
    x.dfs_parent = parent
    for each link (x, w)
        if D(w) is not set
            DFS_Visit(w, x)

Run_DFS(node gadag_root)
    dfs_number = 0
    DFS_Visit(gadag_root, NONE)

```

Figure 7: Basic Depth-First Search algorithm

Given a node x, one can compute the minimal DFS number of the neighbours of x, i.e. $\min(D(w) \text{ if } (x,w) \text{ is a link})$. This gives the earliest attachment point neighbouring x. What is interesting, though, is what is the earliest attachment point from x and x's descendants. This is what is determined by computing the Low-Point value.

In order to compute the low point value, the network is traversed using DFS and the vertices are numbered based on the DFS walk. Let this number be represented as DFS(x). All the edges that lead to already visited nodes during DFS walk are back-edges. The back-edges are important because they give information about reachability of a node via another path.

The low point number is calculated by finding:

```

Low(x) = Minimum of ( DFS(x),
    Lowest DFS(n, x->n is a back-edge),

```


Lowest Low(n , $x \rightarrow n$ is tree edge in DFS walk)).

A detailed algorithm for computing the low-point value is given in Figure 8. Figure 9 illustrates how the lowpoint algorithm applies to a example graph.

```

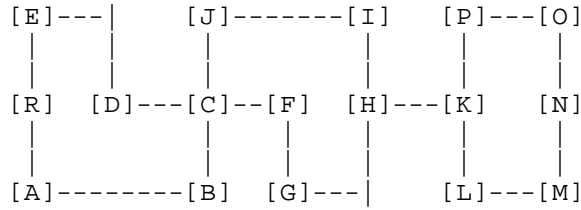
global_variable: dfs_number

Lowpoint_Visit(node x, node parent, interface p_to_x)
  D(x) = dfs_number
  L(x) = D(x)
  dfs_number += 1
  x.dfs_parent = parent
  x.dfs_parent_intf = p_to_x.remote_intf
  x.lowpoint_parent = NONE
  for each ordered_interface intf of x
    if D(intf.remote_node) is not set
      Lowpoint_Visit(intf.remote_node, x, intf)
      if L(intf.remote_node) < L(x)
        L(x) = L(intf.remote_node)
        x.lowpoint_parent = intf.remote_node
        x.lowpoint_parent_intf = intf
    else if intf.remote_node is not parent
      if D(intf.remote_node) < L(x)
        L(x) = D(intf.remote_node)
        x.lowpoint_parent = intf.remote_node
        x.lowpoint_parent_intf = intf

Run_Lowpoint(node gadag_root)
  dfs_number = 0
  Lowpoint_Visit(gadag_root, NONE, NONE)

```

Figure 8: Computing Low-Point value



(a) a non-2-connected graph

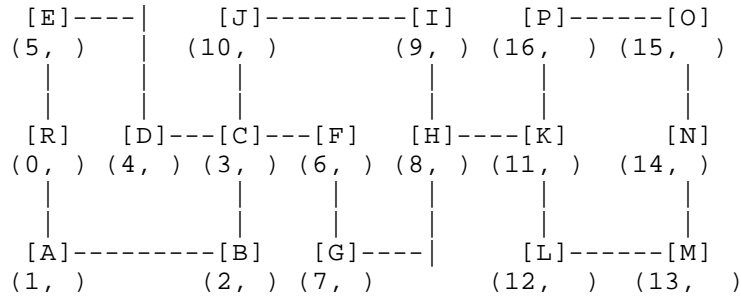
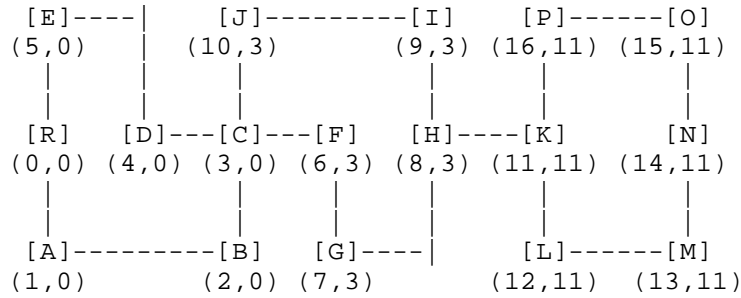
(b) with DFS values assigned $(D(x), L(x))$ (c) with low-point values assigned $(D(x), L(x))$

Figure 9: Example lowpoint value computation

From the low-point value and lowpoint parent, there are three very useful things which motivate our computation.

First, if there is a child c of x such that $L(c) \geq D(x)$, then there are no paths in the network graph that go from c or its descendants to an ancestor of x - and therefore x is a cut-vertex. In Figure 9, this can be seen by looking at the DFS children of C . C has two children - D and F and $L(F) = 3 = D(C)$ so it is clear that C is a cut-vertex and F is in a block where C is the block's root. $L(D) = 0$

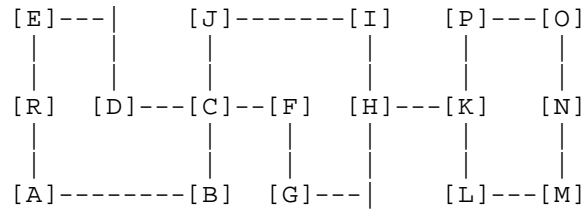
$< 3 = D(C)$ so D has a path to the ancestors of C; in this case, D can go via E to reach R. Comparing the low-point values of all a node's DFS-children with the node's DFS-value is very useful because it allows identification of the cut-vertices and thus the blocks.

Second, by repeatedly following the path given by `lowpoint_parent`, there is a path from `x` back to an ancestor of `x` that does not use the link `[x, x.dfs_parent]` in either direction. The full path need not be taken, but this gives a way of finding an initial cycle and then ears.

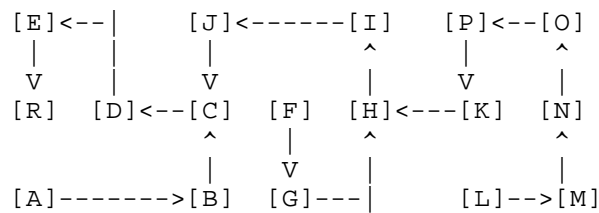
Third, as seen in Figure 9, even if $L(x) < D(x)$, there may be a block that contains both the root and a DFS-child of a node while other DFS-children might be in different blocks. In this example, C's child D is in the same block as R while F is not. It is important to realize that the root of a block may also be the root of another block.

4.4. Blocks in a Graph

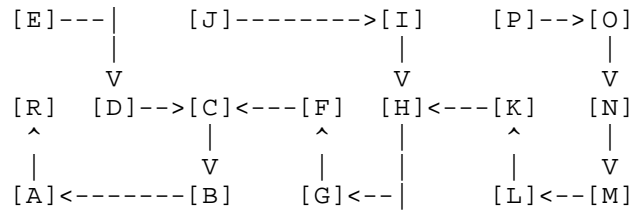
A key idea for the MRT Lowpoint algorithm is that any non-2-connected graph is made up by blocks (e.g. 2-connected clusters, cut-links, and/or isolated nodes). To compute GADAGs and thus MRTs, computation is done in each block to compute ADAGs or Redundant Trees and then those ADAGs or Redundant Trees are combined into a GADAG or MRT.



(a) A graph with four blocks that are:
three 2-connected clusters
and one cut-link



(b) MRT-Blue for destination R



(c) MRT-Red for destination R

Figure 10

Consider the example depicted in Figure 10 (a). In this figure, a special graph is presented, showing us all the ways 2-connected clusters can be connected. It has four blocks: block 1 contains R, A, B, C, D, E, block 2 contains C, F, G, H, I, J, block 3 contains K, L, M, N, O, P, and block 4 is a cut-link containing H and K. As can be observed, the first two blocks have one common node (node C) and blocks 2 and 3 do not have any common node, but they are connected through a cut-link that is block 4. No two blocks can have more than one common node, since two blocks with at least two common nodes would qualify as a single 2-connected cluster.

Moreover, observe that if we want to get from one block to another, we must use a cut-vertex (the cut-vertices in this graph are C, H, K), regardless of the path selected, so we can say that all the paths from block 3 along the MRTs rooted at R will cross K first. This observation means that if we want to find a pair of MRTs rooted at R, then we need to build up a pair of RTs in block 3 with K as a root. Similarly, we need to find another pair of RTs in block 2 with C as a root, and finally, we need the last pair of RTs in block 1 with R as a root. When all the trees are selected, we can simply combine them; when a block is a cut-link (as in block 4), that cut-link is added in the same direction to both of the trees. The resulting trees are depicted in Figure 10 (b) and (c).

Similarly, to create a GADAG it is sufficient to compute ADAGs in each block and connect them.

It is necessary, therefore, to identify the cut-vertices, the blocks and identify the appropriate local-root to use for each block.

4.5. Determining Local-Root and Assigning Block-ID

Each node in a network graph has a local-root, which is the cut-vertex (or root) in the same block that is closest to the root. The local-root is used to determine whether two nodes share a common block.

```

Compute_Localroot(node x, node localroot)
  x.localroot = localroot
  for each DFS child node c of x
    if L(c) < D(x)    //x is not a cut-vertex
      Compute_Localroot(c, x.localroot)
    else
      mark x as cut-vertex
      Compute_Localroot(c, x)

Compute_Localroot(gadag_root, gadag_root)

```

Figure 11: A method for computing local-roots

There are two different ways of computing the local-root for each node. The stand-alone method is given in Figure 11 and better illustrates the concept; it is used by the GADAG construction methods given in Appendix B and Appendix C. The MRT Lowpoint algorithm computes the local-root for a block as part of computing the GADAG using lowpoint inheritance; the essence of this computation is given in Figure 12. Both methods for computing the local-root produce the same results.

```

Get the current node, s.
Compute an ear(either through lowpoint inheritance
or by following dfs parents) from s to a ready node e.
(Thus, s is not e, if there is such ear.)
if s is e
    for each node x in the ear that is not s
        x.localroot = s
else
    for each node x in the ear that is not s or e
        x.localroot = e.localroot

```

Figure 12: Ear-based method for computing local-roots

Once the local-roots are known, two nodes X and Y are in a common block if and only if one of the following three conditions apply.

- o Y's local-root is X's local-root : They are in the same block and neither is the cut-vertex closest to the root.
- o Y's local-root is X: X is the cut-vertex closest to the root for Y's block
- o Y is X's local-root: Y is the cut-vertex closest to the root for X's block

Once we have computed the local-root for each node in the network graph, we can assign for each node, a block id that represents the block in which the node is present. This computation is shown in Figure 13.

```

global_var: max_block_id

Assign_Block_ID(x, cur_block_id)
    x.block_id = cur_block_id
    foreach DFS child c of x
        if (c.local_root is x)
            max_block_id += 1
            Assign_Block_ID(c, max_block_id)
        else
            Assign_Block_ID(c, cur_block_id)

max_block_id = 0
Assign_Block_ID(gadag_root, max_block_id)

```

Figure 13: Assigning block id to identify blocks

5. MRT Lowpoint Algorithm Specification

The MRT Lowpoint algorithm computes one GADAG that is then used by a router to determine its MRT-Blue and MRT-Red next-hops to all destinations. Finally, based upon that information, alternates are selected for each next-hop to each destination. The different parts of this algorithm are described below.

- o Order the interfaces in the network graph. [See Section 5.1]
- o Compute the local MRT Island for the particular MRT Profile. [See Section 5.2]
- o Select the root to use for the GADAG. [See Section 5.3]
- o Initialize all interfaces to UNDIRECTED. [See Section 5.4]
- o Compute the DFS value, e.g. $D(x)$, and lowpoint value, $L(x)$. [See Figure 8]
- o Construct the GADAG. [See Section 5.5]
- o Assign directions to all interfaces that are still UNDIRECTED. [See Section 5.6]
- o From the computing router x , compute the next-hops for the MRT-Blue and MRT-Red. [See Section 5.7]
- o Identify alternates for each next-hop to each destination by determining which one of the blue MRT and the red MRT the computing router x should select. [See Section 5.8]

A Python implementation of this algorithm is given in Appendix A.

5.1. Interface Ordering

To ensure consistency in computation, all routers MUST order interfaces identically down to the set of links with the same metric to the same neighboring node. This is necessary for the DFS in `Lowpoint_Visit` in Section 4.3, where the selection order of the interfaces to explore results in different trees. Consistent interface ordering is also necessary for computing the GADAG, where the selection order of the interfaces to use to form ears can result in different GADAGs. It is also necessary for the topological sort described in Section 5.8, where different topological sort orderings can result in undirected links being added to the GADAG in different directions.

The required ordering between two interfaces from the same router *x* is given in Figure 14.

```
Interface_Compare(interface a, interface b)
  if a.metric < b.metric
    return A_LESS_THAN_B
  if b.metric < a.metric
    return B_LESS_THAN_A
  if a.neighbor.mrt_node_id < b.neighbor.mrt_node_id
    return A_LESS_THAN_B
  if b.neighbor.mrt_node_id < a.neighbor.mrt_node_id
    return B_LESS_THAN_A
  // Same metric to same node, so the order doesn't matter for
  // interoperability.
  return A_EQUAL_TO_B
```

Figure 14: Rules for ranking multiple interfaces. Order is from low to high.

In Figure 14, if two interfaces on a router connect to the same remote router with the same metric, the `Interface_Compare` function returns `A_EQUAL_TO_B`. This is because the order in which those interfaces are initially explored does not affect the final GADAG produced by the algorithm described here. While only one of the links will be added to the GADAG in the initial traversal, the other parallel links will be added to the GADAG with the same direction assigned during the procedure for assigning direction to `UNDIRECTED` links described in Section 5.6. An implementation is free to apply some additional criteria to break ties in interface ordering in this situation, but that criteria is not specified here since it will not affect the final GADAG produced by the algorithm.

The `Interface_Compare` function in Figure 14 relies on the `interface.metric` and the `interface.neighbor.mrt_node_id` values to order interfaces. The exact source of these values for different IGPs and applications is specified in Figure 15. The metric and `mrt_node_id` values for OSPFv2, OSPFv3, and IS-IS provided here is normative. The metric and `mrt_node_id` values for ISIS-PCR in this table should be considered informational. The normative values are specified in [IEEE8021Qca] .

IGP/flooding protocol and application	mrt_node_id of neighbor on interface	metric of interface
OSPFv2 for IP/LDP FRR	4 octet Neighbor Router ID in Link ID field for corresponding point-to-point link in Router-LSA	2 octet Metric field for corresponding point-to-point link in Router-LSA
OSPFv3 for IP/LDP FRR	4 octet Neighbor Router ID field for corresponding point-to-point link in Router-LSA	2 octet Metric field for corresponding point-to-point link in Router-LSA
IS-IS for IP/LDP FRR	7 octet neighbor system ID and pseudonode number in Extended IS Reachability TLV #22 or Multi-Topology IS Neighbor TLV #222	3 octet metric field in Extended IS Reachability TLV #22 or Multi-Topology IS Neighbor TLV #222
ISIS-PCR for protection of traffic in bridged networks	8 octet Bridge ID created from 2 octet Bridge Priority in SPB Instance sub-TLV (type 1) carried in MT-Capability TLV #144 and 6 octet neighbor system ID in Extended IS Reachability TLV #22 or Multi-Topology Intermediate Systems TLV #222 (informational)	3 octet SPB-LINK-METRIC in SPB-Metric sub-TLV (type 29) in Extended IS Reachability TLV #22 or Multi-Topology Intermediate Systems TLV #222. In the case of asymmetric link metrics, the larger link metric is used for both link directions. (informational)

Figure 15: value of interface.neighbor.mrt_node_id and interface.metric to be used for ranking interfaces, for different flooding protocols and applications

The metrics are unsigned integers and MUST be compared as unsigned integers. The results of `mrt_node_id` comparisons MUST be the same as would be obtained by converting the `mrt_node_ids` to unsigned integers using network byte order and performing the comparison as unsigned integers. In the case of IS-IS for IP/LDP FRR with point-to-point links, the pseudonode number (the 7th octet) is zero. Broadcast interfaces will be discussed in Section 7.

5.2. MRT Island Identification

The local MRT Island for a particular MRT profile can be determined by starting from the computing router in the network graph and doing a breadth-first-search (BFS). The BFS explores only links that are in the same area/level, are not IGP-excluded, and are not MRT-ineligible. The BFS explores only nodes that are are not IGP-excluded, and that support the particular MRT profile. See section 7 of [I-D.ietf-rtgwg-mrt-frr-architecture] for more precise definitions of these criteria.

```
MRT_Island_Identification(topology, computing_rtr, profile_id, area)
  for all routers in topology
    rtr.IN_MRT_ISLAND = FALSE
  computing_rtr.IN_MRT_ISLAND = TRUE
  explore_list = { computing_rtr }
  while (explore_list is not empty)
    next_rtr = remove_head(explore_list)
    for each intf in next_rtr
      if (not intf.MRT-ineligible
          and not intf.remote_intf.MRT-ineligible
          and not intf.IGP-excluded and (intf in area)
          and (intf.remote_node supports profile_id) )
        intf.IN_MRT_ISLAND = TRUE
        intf.remote_node.IN_MRT_ISLAND = TRUE
        if (not intf.remote_node.IN_MRT_ISLAND))
          intf.remote_node.IN_MRT_ISLAND = TRUE
          add_to_tail(explore_list, intf.remote_node)
```

Figure 16: MRT Island Identification

5.3. GADAG Root Selection

In Section 8.3 of [I-D.ietf-rtgwg-mrt-frr-architecture], the GADAG Root Selection Policy is described for the MRT default profile. This selection policy allows routers to consistently select a common GADAG Root inside the local MRT Island, based on advertised priority values. The MRT Lowpoint algorithm simply requires that all routers in the MRT Island MUST select the same GADAG Root; the mechanism can vary based upon the MRT profile description. Before beginning

computation, the network graph is reduced to contain only the set of routers that support the specific MRT profile whose MRTs are being computed.

As noted in Section 7, pseudonodes MUST NOT be considered for GADAG root selection.

It is expected that an operator will designate a set of routers as good choices for selection as GADAG root by setting the GADAG Root Selection Priority for that set of routers to lower (more preferred) numerical values. For guidance on setting the GADAG Root Selection Priority values, refer to Section 10.1.

5.4. Initialization

Before running the algorithm, there is the standard type of initialization to be done, such as clearing any computed DFS-values, lowpoint-values, DFS-parents, lowpoint-parents, any MRT-computed next-hops, and flags associated with algorithm.

It is assumed that a regular SPF computation has been run so that the primary next-hops from the computing router to each destination are known. This is required for determining alternates at the last step.

Initially, all interfaces MUST be initialized to UNDIRECTED. Whether they are OUTGOING, INCOMING or both is determined when the GADAG is constructed and augmented.

It is possible that some links and nodes will be marked using standard IGP mechanisms to discourage or prevent transit traffic. Section 7.3.1 of [I-D.ietf-rtgwg-mrt-frr-architecture] describes how those links and nodes are excluded from MRT Island formation.

MRT-FRR also has the ability to advertise links MRT-Ineligible, as described in Section 7.3.2 of [I-D.ietf-rtgwg-mrt-frr-architecture]. These links are excluded from the MRT Island and the GADAG. Computation of MRT next-hops will therefore not use any MRT-ineligible links. The MRT algorithm does still need to consider MRT-ineligible links when computing FRR alternates, because an MRT-ineligible link can still be the shortest-path next-hop to reach a destination.

When a broadcast interface is advertised as MRT-ineligible, then the pseudo-node representing the entire broadcast network MUST NOT be included in the MRT Island. This is equivalent to excluding all of the broadcast interfaces on that broadcast network from the MRT Island.

5.5. Constructing the GADAG using lowpoint inheritance

As discussed in Section 4.2, it is necessary to find ears from a node *x* that is already in the GADAG (known as IN_GADAG). Two different methods are used to find ears in the algorithm. The first is by going to a not IN_GADAG DFS-child and then following the chain of low-point parents until an IN_GADAG node is found. The second is by going to a not IN_GADAG neighbor and then following the chain of DFS parents until an IN_GADAG node is found. As an ear is found, the associated interfaces are marked based on the direction taken. The nodes in the ear are marked as IN_GADAG. In the algorithm, first the ears via DFS-children are found and then the ears via DFS-neighbors are found.

By adding both types of ears when an IN_GADAG node is processed, all ears that connect to that node are found. The order in which the IN_GADAG nodes is processed is, of course, key to the algorithm. The order is a stack of ears so the most recent ear is found at the top of the stack. Of course, the stack stores nodes and not ears, so an ordered list of nodes, from the first node in the ear to the last node in the ear, is created as the ear is explored and then that list is pushed onto the stack.

Each ear represents a partial order (see Figure 4) and processing the nodes in order along each ear ensures that all ears connecting to a node are found before a node higher in the partial order has its ears explored. This means that the direction of the links in the ear is always from the node *x* being processed towards the other end of the ear. Additionally, by using a stack of ears, this means that any unprocessed nodes in previous ears can only be ordered higher than nodes in the ears below it on the stack.

In this algorithm that depends upon Low-Point inheritance, it is necessary that every node have a low-point parent that is not itself. If a node is a cut-vertex, that may not yet be the case. Therefore, any nodes without a low-point parent will have their low-point parent set to their DFS parent and their low-point value set to the DFS-value of their parent. This assignment also properly allows an ear between two cut-vertices.

Finally, the algorithm simultaneously computes each node's local-root, as described in Figure 12. This is further elaborated as follows. The local-root can be inherited from the node at the end of the ear unless the end of the ear is *x* itself, in which case the local-root for all the nodes in the ear would be *x*. This is because whenever the first cycle is found in a block, or an ear involving a bridge is computed, the cut-vertex closest to the root would be *x* itself. In all other scenarios, the properties of lowpoint/dfs

parents ensure that the end of the ear will be in the same block, and thus inheriting its local-root would be the correct local-root for all newly added nodes.

The pseudo-code for the GADAG algorithm (assuming that the adjustment of lowpoint for cut-vertices has been made) is shown in Figure 17.

```

Construct_Ear(x, Stack, intf, ear_type)
    ear_list = empty
    cur_node = intf.remote_node
    cur_intf = intf
    not_done = true

    while not_done
        cur_intf.UNDIRECTED = false
        cur_intf.OUTGOING = true
        cur_intf.remote_intf.UNDIRECTED = false
        cur_intf.remote_intf.INCOMING = true

        if cur_node.IN_GADAG is false
            cur_node.IN_GADAG = true
            add_to_list_end(ear_list, cur_node)
            if ear_type is CHILD
                cur_intf = cur_node.lowpoint_parent_intf
                cur_node = cur_node.lowpoint_parent
            else // ear_type must be NEIGHBOR
                cur_intf = cur_node.dfs_parent_intf
                cur_node = cur_node.dfs_parent
        else
            not_done = false

    if (ear_type is CHILD) and (cur_node is x)
        // x is a cut-vertex and the local root for
        // the block in which the ear is computed
        x.IS_CUT_VERTEX = true
        localroot = x
    else
        // Inherit local-root from the end of the ear
        localroot = cur_node.localroot
    while ear_list is not empty
        y = remove_end_item_from_list(ear_list)
        y.localroot = localroot
        push(Stack, y)

Construct_GADAG_via_Lowpoint(topology, gadag_root)
    gadag_root.IN_GADAG = true
    gadag_root.localroot = None
    Initialize Stack to empty

```

```

push gadag_root onto Stack
while (Stack is not empty)
  x = pop(Stack)
  foreach ordered_interface intf of x
    if ((intf.remote_node.IN_GADAG == false) and
        (intf.remote_node.dfs_parent is x))
      Construct_Ear(x, Stack, intf, CHILD)
  foreach ordered_interface intf of x
    if ((intf.remote_node.IN_GADAG == false) and
        (intf.remote_node.dfs_parent is not x))
      Construct_Ear(x, Stack, intf, NEIGHBOR)

Construct_GADAG_via_Lowpoint(topology, gadag_root)

```

Figure 17: Low-point Inheritance GADAG algorithm

5.6. Augmenting the GADAG by directing all links

The GADAG, regardless of the method used to construct it, at this point could be used to find MRTs, but the topology does not include all links in the network graph. That has two impacts. First, there might be shorter paths that respect the GADAG partial ordering and so the alternate paths would not be as short as possible. Second, there may be additional paths between a router *x* and the root that are not included in the GADAG. Including those provides potentially more bandwidth to traffic flowing on the alternates and may reduce congestion compared to just using the GADAG as currently constructed.

The goal is thus to assign direction to every remaining link marked as `UNDIRECTED` to improve the paths and number of paths found when the MRTs are computed.

To do this, we need to establish a total order that respects the partial order described by the GADAG. This can be done using Kahn's topological sort [Kahn_1962_topo_sort] which essentially assigns a number to a node *x* only after all nodes before it (e.g. with a link incoming to *x*) have had their numbers assigned. The only issue with the topological sort is that it works on DAGs and not ADAGs or GADAGs.

To convert a GADAG to a DAG, it is necessary to remove all links that point to a root of block from within that block. That provides the necessary conversion to a DAG and then a topological sort can be done. When adding undirected links to the GADAG, links connecting the block root to other nodes in that block need special handling because the topological order will not always give the right answer for those links. There are three cases to consider. If the undirected link in question has another parallel link between the

same two nodes that is already directed, then the direction of the undirected link can be inherited from the previously directed link. In the case of parallel cut links, we set all of the parallel links to both INCOMING and OUTGOING. Otherwise, the undirected link in question is set to OUTGOING from the block root node. A cut-link can then be identified by the fact that it will be directed both INCOMING and OUTGOING in the GADAG. The exact details of this whole process are captured in Figure 18

```

Add_Undirected_Block_Root_Links(topo, gadag_root)
  foreach node x in topo
    if x.IS_CUT_VERTEX or x is gadag_root
      foreach interface i of x
        if (i.remote_node.localroot is not x
            or i.PROCESSED )
          continue
        Initialize bundle_list to empty
        bundle.UNDIRECTED = true
        bundle.OUTGOING = false
        bundle.INCOMING = false
        foreach interface i2 in x
          if i2.remote_node is i.remote_node
            add_to_list_end(bundle_list, i2)
          if not i2.UNDIRECTED:
            bundle.UNDIRECTED = false
            if i2.INCOMING:
              bundle.INCOMING = true
            if i2.OUTGOING:
              bundle.OUTGOING = true
        if bundle.UNDIRECTED
          foreach interface i3 in bundle_list
            i3.UNDIRECTED = false
            i3.remote_intf.UNDIRECTED = false
            i3.PROCESSED = true
            i3.remote_intf.PROCESSED = true
            i3.OUTGOING = true
            i3.remote_intf.INCOMING = true
        else
          if (bundle.OUTGOING and bundle.INCOMING)
            foreach interface i3 in bundle_list
              i3.UNDIRECTED = false
              i3.remote_intf.UNDIRECTED = false
              i3.PROCESSED = true
              i3.remote_intf.PROCESSED = true
              i3.OUTGOING = true
              i3.INCOMING = true
              i3.remote_intf.INCOMING = true
              i3.remote_intf.OUTGOING = true

```

```

        else if bundle.OUTGOING
            foreach interface i3 in bundle_list
                i3.UNDIRECTED = false
                i3.remote_intf.UNDIRECTED = false
                i3.PROCESSED = true
                i3.remote_intf.PROCESSED = true
                i3.OUTGOING = true
                i3.remote_intf.INCOMING = true
        else if bundle.INCOMING
            foreach interface i3 in bundle_list
                i3.UNDIRECTED = false
                i3.remote_intf.UNDIRECTED = false
                i3.PROCESSED = true
                i3.remote_intf.PROCESSED = true
                i3.INCOMING = true
                i3.remote_intf.OUTGOING = true

Modify_Block_Root_Incoming_Links(topo, gadag_root)
    foreach node x in topo
        if x.IS_CUT_VERTEX or x is gadag_root
            foreach interface i of x
                if i.remote_node.localroot is x
                    if i.INCOMING:
                        i.INCOMING = false
                        i.INCOMING_STORED = true
                        i.remote_intf.OUTGOING = false
                        i.remote_intf.OUTGOING_STORED = true

Revert_Block_Root_Incoming_Links(topo, gadag_root)
    foreach node x in topo
        if x.IS_CUT_VERTEX or x is gadag_root
            foreach interface i of x
                if i.remote_node.localroot is x
                    if i.INCOMING_STORED
                        i.INCOMING = true
                        i.remote_intf.OUTGOING = true
                        i.INCOMING_STORED = false
                        i.remote_intf.OUTGOING_STORED = false

Run_Topological_Sort_GADAG(topo, gadag_root)
    Modify_Block_Root_Incoming_Links(topo, gadag_root)
    foreach node x in topo
        node.unvisited = 0
        foreach interface i of x
            if (i.INCOMING)
                node.unvisited += 1
    Initialize working_list to empty
    Initialize topo_order_list to empty

```



```

    add_to_list_end(working_list, gadag_root)
    while working_list is not empty
        y = remove_start_item_from_list(working_list)
        add_to_list_end(topo_order_list, y)
        foreach ordered_interface i of y
            if intf.OUTGOING
                i.remote_node.unvisited -= 1
                if i.remote_node.unvisited is 0
                    add_to_list_end(working_list, i.remote_node)
    next_topo_order = 1
    while topo_order_list is not empty
        y = remove_start_item_from_list(topo_order_list)
        y.topo_order = next_topo_order
        next_topo_order += 1
    Revert_Block_Root_Incoming_Links(topo, gadag_root)

def Set_Other_Undirected_Links_Based_On_Topo_Order(topo)
    foreach node x in topo
        foreach interface i of x
            if i.UNDIRECTED:
                if x.topo_order < i.remote_node.topo_order
                    i.OUTGOING = true
                    i.UNDIRECTED = false
                    i.remote_intf.INCOMING = true
                    i.remote_intf.UNDIRECTED = false
                else
                    i.INCOMING = true
                    i.UNDIRECTED = false
                    i.remote_intf.OUTGOING = true
                    i.remote_intf.UNDIRECTED = false

Add_Undirected_Links(topo, gadag_root)
Add_Undirected_Block_Root_Links(topo, gadag_root)
Run_Topological_Sort_GADAG(topo, gadag_root)
Set_Other_Undirected_Links_Based_On_Topo_Order(topo)

Add_Undirected_Links(topo, gadag_root)

```

Figure 18: Assigning direction to UNDIRECTED links

Proxy-nodes do not need to be added to the network graph. They cannot be transited and do not affect the MRTs that are computed. The details of how the MRT-Blue and MRT-Red next-hops are computed for proxy-nodes and how the appropriate alternate next-hops are selected is given in Section 5.9.

5.7. Compute MRT next-hops

As was discussed in Section 4.1, once a ADAG is found, it is straightforward to find the next-hops from any node X to the ADAG root. However, in this algorithm, we will reuse the common GADAG and find not only the one pair of MRTs rooted at the GADAG root with it, but find a pair rooted at each node. This is useful since it is significantly faster to compute.

The method for computing differently rooted MRTs from the common GADAG is based on two ideas. First, if two nodes X and Y are ordered with respect to each other in the partial order, then an SPF along OUTGOING links (an increasing-SPF) and an SPF along INCOMING links (a decreasing-SPF) can be used to find the increasing and decreasing paths. Second, if two nodes X and Y aren't ordered with respect to each other in the partial order, then intermediary nodes can be used to create the paths by increasing/decreasing to the intermediary and then decreasing/increasing to reach Y.

As usual, the two basic ideas will be discussed assuming the network is two-connected. The generalization to multiple blocks is discussed in Section 5.7.4. The full algorithm is given in Section 5.7.5.

5.7.1. MRT next-hops to all nodes ordered with respect to the computing node

To find two node-disjoint paths from the computing router X to any node Y, depends upon whether $Y \gg X$ or $Y \ll X$. As shown in Figure 19, if $Y \gg X$, then there is an increasing path that goes from X to Y without crossing R; this contains nodes in the interval $[X, Y]$. There is also a decreasing path that decreases towards R and then decreases from R to Y; this contains nodes in the interval $[X, R\text{-small}]$ or $[R\text{-great}, Y]$. The two paths cannot have common nodes other than X and Y.

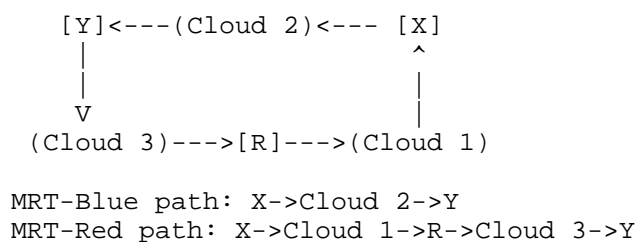


Figure 19: $Y \gg X$

Similar logic applies if $Y < X$, as shown in Figure 20. In this case, the increasing path from X increases to R and then increases from R to Y to use nodes in the intervals $[X, R\text{-great}]$ and $[R\text{-small}, Y]$. The decreasing path from X reaches Y without crossing R and uses nodes in the interval $[Y, X]$.

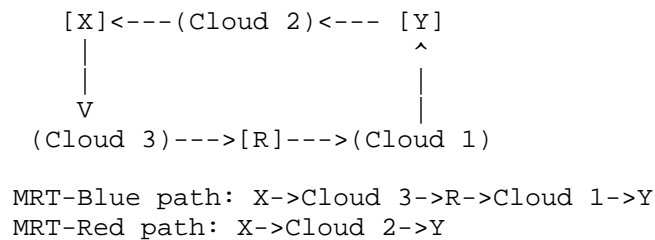


Figure 20: $Y < X$

5.7.2. MRT next-hops to all nodes not ordered with respect to the computing node

When X and Y are not ordered, the first path should increase until we get to a node G, where $G > Y$. At G, we need to decrease to Y. The other path should be just the opposite: we must decrease until we get to a node H, where $H < Y$, and then increase. Since R is smaller and greater than Y, such G and H must exist. It is also easy to see that these two paths must be node disjoint: the first path contains nodes in interval $[X, G]$ and $[Y, G]$, while the second path contains nodes in interval $[H, X]$ and $[H, Y]$. This is illustrated in Figure 21. It is necessary to decrease and then increase for the MRT-Blue and increase and then decrease for the MRT-Red; if one simply increased for one and decreased for the other, then both paths would go through the root R.

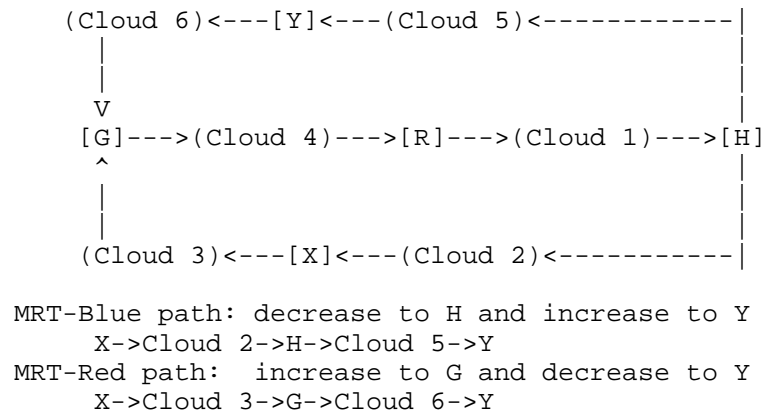


Figure 21: X and Y unordered

This gives disjoint paths as long as G and H are not the same node. Since $G \gg Y$ and $H \ll Y$, if G and H could be the same node, that would have to be the root R. This is not possible because there is only one incoming interface to the root R which is created when the initial cycle is found. Recall from Figure 6 that whenever an ear was found to have an end that was the root R, the ear was directed from R so that the associated interface on R is outgoing and not incoming. Therefore, there must be exactly one node M which is the largest one before R, so the MRT-Red path will never reach R; it will turn at M and decrease to Y.

5.7.3. Computing Redundant Tree next-hops in a 2-connected Graph

The basic ideas for computing RT next-hops in a 2-connected graph were given in Section 5.7.1 and Section 5.7.2. Given these two ideas, how can we find the trees?

If some node X only wants to find the next-hops (which is usually the case for IP networks), it is enough to find which nodes are greater and less than X, and which are not ordered; this can be done by running an increasing-SPF and a decreasing-SPF rooted at X and not exploring any links from the ADAG root.

In principle, an traversal method other than SPF could be used to traverse the GADAG in the process of determining blue and red next-hops that result in maximally redundant trees. This will be the case as long as one traversal uses the links in the direction specified by the GADAG and the other traversal uses the links in the direction opposite of that specified by the GADAG. However, a different traversal algorithm will generally result in different blue and red next-hops. Therefore, the algorithm specified here requires the use

of SPF to traverse the GADAG to generate MRT blue and red next-hops, as described below.

An increasing-SPF rooted at X and not exploring links from the root will find the increasing next-hops to all $Y \gg X$. Those increasing next-hops are X's next-hops on the MRT-Blue to reach Y. A decreasing-SPF rooted at X and not exploring links from the root will find the decreasing next-hops to all $Z \ll X$. Those decreasing next-hops are X's next-hops on the MRT-Red to reach Z. Since the root R is both greater than and less than X, after this increasing-SPF and decreasing-SPF, X's next-hops on the MRT-Blue and on the MRT-Red to reach R are known. For every node $Y \gg X$, X's next-hops on the MRT-Red to reach Y are set to those on the MRT-Red to reach R. For every node $Z \ll X$, X's next-hops on the MRT-Blue to reach Z are set to those on the MRT-Blue to reach R.

For those nodes which were not reached by either the increasing-SPF or the decreasing-SPF, we can determine the next-hops as well. The increasing MRT-Blue next-hop for a node which is not ordered with respect to X is the next-hop along the decreasing MRT-Red towards R, and the decreasing MRT-Red next-hop is the next-hop along the increasing MRT-Blue towards R. Naturally, since R is ordered with respect to all the nodes, there will always be an increasing and a decreasing path towards it. This algorithm does not provide the complete specific path taken but just the appropriate next-hops to use. The identities of G and H are not determined by the computing node X.

The final case to consider is when the GADAG root R computes its own next-hops. Since the GADAG root R is \ll all other nodes, running an increasing-SPF rooted at R will reach all other nodes; the MRT-Blue next-hops are those found with this increasing-SPF. Similarly, since the GADAG root R is \gg all other nodes, running a decreasing-SPF rooted at R will reach all other nodes; the MRT-Red next-hops are those found with this decreasing-SPF.

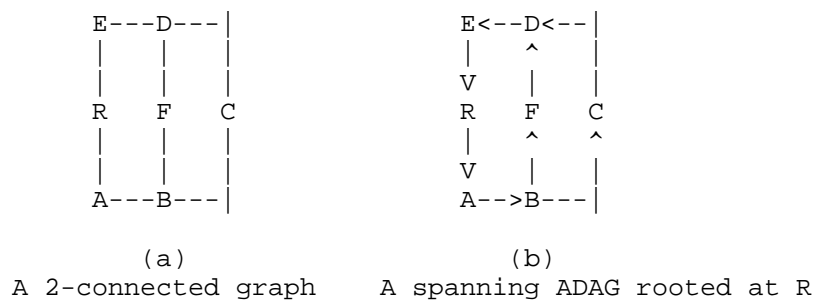


Figure 22

As an example consider the situation depicted in Figure 22. Node C runs an increasing-SPF and a decreasing-SPF on the ADAG. The increasing-SPF reaches D, E and R and the decreasing-SPF reaches B, A and R. $E \gg C$. So towards E the MRT-Blue next-hop is D, since E was reached on the increasing path through D. And the MRT-Red next-hop towards E is B, since R was reached on the decreasing path through B. Since $E \gg D$, D will similarly compute its MRT-Blue next-hop to be E, ensuring that a packet on MRT-Blue will use path C-D-E. B, A and R will similarly compute the MRT-Red next-hops towards E (which is ordered less than B, A and R), ensuring that a packet on MRT-Red will use path C-B-A-R-E.

C can determine the next-hops towards F as well. Since F is not ordered with respect to C, the MRT-Blue next-hop is the decreasing one towards R (which is B) and the MRT-Red next-hop is the increasing one towards R (which is D). Since $F \gg B$, for its MRT-Blue next-hop towards F, B will use the real increasing next-hop towards F. So a packet forwarded to B on MRT-Blue will get to F on path C-B-F. Similarly, D will use the real decreasing next-hop towards F as its MRT-Red next-hop, a packet on MRT-Red will use path C-D-F.

5.7.4. Generalizing for a graph that isn't 2-connected

If a graph isn't 2-connected, then the basic approach given in Section 5.7.3 needs some extensions to determine the appropriate MRT next-hops to use for destinations outside the computing router X's blocks. In order to find a pair of maximally redundant trees in that graph we need to find a pair of RTs in each of the blocks (the root of these trees will be discussed later), and combine them.

When computing the MRT next-hops from a router X, there are three basic differences:

1. Only nodes in a common block with X should be explored in the increasing-SPF and decreasing-SPF.
2. Instead of using the GADAG root, X's local-root should be used. This has the following implications:
 - A. The links from X's local-root should not be explored.
 - B. If a node is explored in the outgoing SPF so $Y \gg X$, then X's MRT-Red next-hops to reach Y uses X's MRT-Red next-hops to reach X's local-root and if $Z \ll X$, then X's MRT-Blue next-hops to reach Z uses X's MRT-Blue next-hops to reach X's local-root.

- C. If a node W in a common block with X was not reached in the increasing-SPF or decreasing-SPF, then W is unordered with respect to X. X's MRT-Blue next-hops to W are X's decreasing (aka MRT-Red) next-hops to X's local-root. X's MRT-Red next-hops to W are X's increasing (aka MRT-Blue) next-hops to X's local-root.
- 3. For nodes in different blocks, the next-hops must be inherited via the relevant cut-vertex.

These are all captured in the detailed algorithm given in Section 5.7.5.

5.7.5. Complete Algorithm to Compute MRT Next-Hops

The complete algorithm to compute MRT Next-Hops for a particular router X is given in Figure 23. In addition to computing the MRT-Blue next-hops and MRT-Red next-hops used by X to reach each node Y, the algorithm also stores an "order_proxy", which is the proper cut-vertex to reach Y if it is outside the block, and which is used later in deciding whether the MRT-Blue or the MRT-Red can provide an acceptable alternate for a particular primary next-hop.

```

In_Common_Block(x, y)
  if ( (x.block_id is y.block_id)
      or (x is y.localroot) or (y is x.localroot) )
    return true
  return false

Store_Results(y, direction)
  if direction is FORWARD
    y.higher = true
    y.blue_next_hops = y.next_hops
  if direction is REVERSE
    y.lower = true
    y.red_next_hops = y.next_hops

SPF_No_Traverse_Block_Root(spfx_root, block_root, direction)
  Initialize spfx_heap to empty
  Initialize nodes' spfx_metric to infinity and next_hops to empty
  spfx_root.spfx_metric = 0
  insert(spfx_heap, spfx_root)
  while (spfx_heap is not empty)
    min_node = remove_lowest(spfx_heap)
    Store_Results(min_node, direction)
    if ((min_node is spfx_root) or (min_node is not block_root))
      foreach interface intf of min_node
        if ( ( (direction is FORWARD) and intf.OUTGOING) or

```

```

        ((direction is REVERSE) and intf.INCOMING) )
        and In_Common_Block(spf_root, intf.remote_node) )
    path_metric = min_node.spf_metric + intf.metric
    if path_metric < intf.remote_node.spf_metric
        intf.remote_node.spf_metric = path_metric
        if min_node is spf_root
            intf.remote_node.next_hops = make_list(intf)
        else
            intf.remote_node.next_hops = min_node.next_hops
            insert_or_update(spf_heap, intf.remote_node)
    else if path_metric == intf.remote_node.spf_metric
        if min_node is spf_root
            add_to_list(intf.remote_node.next_hops, intf)
        else
            add_list_to_list(intf.remote_node.next_hops,
                            min_node.next_hops)

SetEdge(y)
    if y.blue_next_hops is empty and y.red_next_hops is empty
        SetEdge(y.localroot)
        y.blue_next_hops = y.localroot.blue_next_hops
        y.red_next_hops = y.localroot.red_next_hops
        y.order_proxy = y.localroot.order_proxy

Compute_MRT_NextHops(x, gadag_root)
    foreach node y
        y.higher = y.lower = false
        clear y.red_next_hops and y.blue_next_hops
        y.order_proxy = y
    SPF_No_Traverse_Block_Root(x, x.localroot, FORWARD)
    SPF_No_Traverse_Block_Root(x, x.localroot, REVERSE)

    // red and blue next-hops are stored to x.localroot as different
    // paths are found via the SPF and reverse-SPF.
    // Similarly any nodes whose local-root is x will have their
    // red_next_hops and blue_next_hops already set.

    // Handle nodes in the same block that aren't the local-root
    foreach node y
        if (y.IN_MRT_ISLAND and (y is not x) and
            (y.block_id is x.block_id) )
            if y.higher
                y.red_next_hops = x.localroot.red_next_hops
            else if y.lower
                y.blue_next_hops = x.localroot.blue_next_hops
            else
                y.blue_next_hops = x.localroot.red_next_hops
                y.red_next_hops = x.localroot.blue_next_hops

```



```

// Inherit next-hops and order_proxies to other components
if (x is not gadag_root) and (x.localroot is not gadag_root)
    gadag_root.blue_next_hops = x.localroot.blue_next_hops
    gadag_root.red_next_hops = x.localroot.red_next_hops
    gadag_root.order_proxy = x.localroot
foreach node y
    if (y is not gadag_root) and (y is not x) and y.IN_MRT_ISLAND
        SetEdge(y)

max_block_id = 0
Assign_Block_ID(gadag_root, max_block_id)
Compute_MRT_NextHops(x, gadag_root)

```

Figure 23

5.8. Identify MRT alternates

At this point, a computing router *S* knows its MRT-Blue next-hops and MRT-Red next-hops for each destination in the MRT Island. The primary next-hops along the SPT are also known. It remains to determine for each primary next-hop to a destination *D*, which of the MRTs avoids the primary next-hop node *F*. This computation depends upon data set in `Compute_MRT_NextHops` such as each node *y*'s `y.blue_next_hops`, `y.red_next_hops`, `y.order_proxy`, `y.higher`, `y.lower` and `topo_orders`. Recall that any router knows only which are the nodes greater and lesser than itself, but it cannot decide the relation between any two given nodes easily; that is why we need topological ordering.

For each primary next-hop node *F* to each destination *D*, *S* can call `Select_Alternates(S, D, F, primary_intf)` to determine whether to use the MRT-Blue or MRT-Red next-hops as the alternate next-hop(s) for that primary next hop. The algorithm is given in Figure 24 and discussed afterwards.

```

Select_Alternates_Internal(D, F, primary_intf,
                          D_lower, D_higher, D_topo_order):
    if D_higher and D_lower
        if F.HIGHER and F.LOWER
            if F.topo_order < D_topo_order
                return USE_RED
            else
                return USE_BLUE
        if F.HIGHER
            return USE_RED
        if F.LOWER
            return USE_BLUE
    //F unordered wrt S

```

```
        return USE_RED_OR_BLUE

    else if D_higher
        if F.HIGHER and F.LOWER
            return USE_BLUE
        if F.LOWER
            return USE_BLUE
        if F.HIGHER
            if (F.topo_order > D_topo_order)
                return USE_BLUE
            if (F.topo_order < D_topo_order)
                return USE_RED
        //F unordered wrt S
        return USE_RED_OR_BLUE

    else if D_lower
        if F.HIGHER and F.LOWER
            return USE_RED
        if F.HIGHER
            return USE_RED
        if F.LOWER
            if F.topo_order > D_topo_order
                return USE_BLUE
            if F.topo_order < D_topo_order
                return USE_RED
        //F unordered wrt S
        return USE_RED_OR_BLUE

    else //D is unordered wrt S
        if F.HIGHER and F.LOWER
            if primary_intf.OUTGOING and primary_intf.INCOMING
                return USE_RED_OR_BLUE
            if primary_intf.OUTGOING
                return USE_BLUE
            if primary_intf.INCOMING
                return USE_RED
            //primary_intf not in GADAG
            return USE_RED
        if F.LOWER
            return USE_RED
        if F.HIGHER
            return USE_BLUE
        //F unordered wrt S
        if F.topo_order > D_topo_order:
            return USE_BLUE
        else:
            return USE_RED
```

```

Select_Alternates(D, F, primary_intf)
  if not In_Common_Block(F, S)
    return PRIM_NH_IN_DIFFERENT_BLOCK
  if (D is F) or (D.order_proxy is F)
    return PRIM_NH_IS_D_OR_OP_FOR_D
  D_lower = D.order_proxy.LOWER
  D_higher = D.order_proxy.HIGHER
  D_topo_order = D.order_proxy.topo_order
  return Select_Alternates_Internal(D, F, primary_intf,
                                    D_lower, D_higher, D_topo_order)

```

Figure 24: Select_Alternates() and Select_Alternates_Internal()

It is useful to first handle the case where F is also D , or F is the order proxy for D . In this case, only link protection is possible. The MRT that doesn't use the failed primary next-hop is used. If both MRTs use the primary next-hop, then the primary next-hop must be a cut-link, so either MRT could be used but the set of MRT next-hops must be pruned to avoid the failed primary next-hop interface. To indicate this case, `Select_Alternates` returns `PRIM_NH_IS_D_OR_OP_FOR_D`. Explicit pseudo-code to handle the three sub-cases above is not provided.

The logic behind `Select_Alternates_Internal` is described in Figure 25. As an example, consider the first case described in the table, where the $D \gg S$ and $D \ll S$. If this is true, then either S or D must be the block root, R . If $F \gg S$ and $F \ll S$, then S is the block root. So the blue path from S to D is the increasing path to D , and the red path S to D is the decreasing path to D . If the $F.topo_order < D.topo_order$, then either F is ordered higher than D or F is unordered with respect to D . Therefore, F is either on a decreasing path from S to D , or it is on neither an increasing nor a decreasing path from S to D . In either case, it is safe to take an increasing path from S to D to avoid F . We know that when S is R , the increasing path is the blue path, so it is safe to use the blue path to avoid F .

If instead $F.topo_order > D.topo_order$, then either F is ordered lower than D , or F is unordered with respect to D . Therefore, F is either on an increasing path from S to D , or it is on neither an increasing nor a decreasing path from S to D . In either case, it is safe to take a decreasing path from S to D to avoid F . We know that when S is R , the decreasing path is the red path, so it is safe to use the red path to avoid F .

If $F \gg S$ or $F \ll S$ (but not both), then D is the block root. We then know that the blue path from S to D is the increasing path to R , and the red path is the decreasing path to R . When $F \gg S$, we deduce that

F is on an increasing path from S to R. So in order to avoid F, we use a decreasing path from S to R, which is the red path. Instead, when $F \ll S$, we deduce that F is on a decreasing path from S to R. So in order to avoid F, we use an increasing path from S to R, which is the blue path.

All possible cases are systematically described in the same manner in the rest of the table.

D wrt S	MRT blue and red path properties	F wrt S	additional criteria	F wrt MRT (deduced)	Alternate
D >> S and D << S, D is R, or S is R	Blue path: Increasing path to R. Red path: Decreasing path to R.	F >> S only	additional criteria not needed	F on an increasing path from S to R	Use Red to avoid F
		F << S only	additional criteria not needed	F on a decreasing path from S to R	Use Blue to avoid F
	Blue path: Increasing path to D. Red path: Decreasing path to D.	F >> S and F << S, F is R	topo(F) > topo(D) implies that F >> D or F ?? D	F on a decreasing path from S to D or neither	Use Blue to avoid F
			topo(F) < topo(D) implies that F << D or F ?? D	F on an increasing path from S to D or neither	Use Red to avoid F
		F ?? S	Can only occur when link between F and S is marked MRT_INELIGIBLE	F is on neither increasing nor decr. path from S to D or R	Use Red or Blue to avoid F
	Blue path: Increasing shortest path from	F << S only	additional criteria not needed	F on decreasing path from S to R	Use Blue to avoid F

	S to D. Red path: Decreasing shortest path from S to R, then decreasing shortest path from R to D.	F>>S only	topo(F)>topo(D) implies that F>>D or F??D	F on decreasing path from R to D or neither	Use Blue to avoid F
			topo(F)<topo(D) implies that F<<D or F??D	F on increasing path from S to D or neither	Use Red to avoid F
		F>>S and F<<S, F is R	additional criteria not needed	F on Red	Use Blue to avoid F
		F??S	Can only occur when link between F and S is marked MRT_INELIGIBLE	F is on neither increasing nor decr. path from S to D or R	Use Red or Blue to avoid F
D<<S only	Blue path: Increasing shortest path from S to R, then increasing shortest path from R to D. Red path: Decreasing shortest path from S to D.	F>>S only	additional criteria not needed	F on increasing path from S to R	Use Red to avoid F
		F<<S only	topo(F)>topo(D) implies that F>>D or F??D	F on decreasing path from R to D or neither	Use Blue to avoid F
			topo(F)<topo(D) implies that F<<D or F??D	F on increasing path from S to D or neither	Use Red to avoid F
		F>>S	additional	F on Blue	Use Red

		and F<<S, F is R	criteria not needed		to avoid F
		F??S	Can only occur when link between F and S is marked MRT_INELIGIBLE	F is on neither increasing nor decr. path from S to D or R	Use Red or Blue to avoid F
D??S	Blue path: Decr. from S to first node K<<D, then incr. to D. Red path: Incr. from S to first node L>>D, then decr.	F<<S only	additional criteria not needed	F on a decreasing path from S to K.	Use Red to avoid F
		F>>S only	additional criteria not needed	F on an increasing path from S to L	Use Blue to avoid F
		F??S	F<-->S link is MRT_INELIGIBLE		
			topo(F)>topo(D) implies that F>>D or F??D	F on decr. path from L to D or neither	Use Blue to avoid F
			topo(F)<topo(D) implies that F<<D or F??D	F on incr. path from K to D or neither	Use Red to avoid F
		F>>S and F<<S, F is R	GADAG link direction S->F	F on an incr. path from S	Use Blue to avoid F
			GADAG link direction S<-F	F on a decr. path from S	Use Red to avoid F
			GADAG link direction S<-->F	Either F is the order proxy for D (case already handled) or D	

				is in a different block from F, in which case Red or Blue avoids F
			S-F link not in GADAG, only when S-F link is MRT_INELIGIBLE	Relies on special construction of GADAG to demonstrate that using Red avoids F (see text)

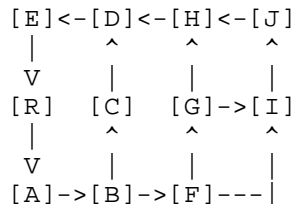
Figure 25: determining MRT next-hops and alternates based on the partial order and topological sort relationships between the source(S), destination(D), primary next-hop(F), and block root(R).

topo(N) indicates the topological sort value of node N. X??Y indicates that node X is unordered with respect to node Y. It is assumed that the case where F is D, or where F is the order proxy for D, has already been handled.

The last case in Figure 25 requires additional explanation. The fact that the red path from S to D in this case avoids F relies on a special property of the GADAGs that we have constructed in this algorithm, a property not shared by all GADAGs in general. When D is unordered with respect to S, and F is the localroot for S, it can occur that the link between S and F is not in the GADAG only when that link has been marked MRT_INELIGIBLE. For an arbitrary GADAG, S doesn't have enough information based on the computed order relationships to determine if the red path or blue path will hit F (which is also the localroot) before hitting K or L, and making it safely to D. However, the GADAGs that we construct using the algorithm in this document are not arbitrary GADAGs. They have the additional property that incoming links to a localroot come from only one other node in the same block. This is a result of the method of construction. This additional property guarantees that the red path from S to D will never pass through the localroot of S. (That would require the localroot to play the role of L, the first node in the path ordered higher than D, which would in turn require the localroot to have two incoming links in the GADAG, which cannot happen.) Therefore it is safe to use the red path to avoid F with these specially constructed GADAGs.

As an example of how `Select_Alternates_Internal()` operates, consider the ADAG depicted in Figure 26 and first suppose that G is the source, D is the destination and H is the failed next-hop. Since $D \gg G$, we need to compare `H.topo_order` and `D.topo_order`. Since $D.topo_order > H.topo_order$, D must be either higher than H or unordered with respect to H, so we should select the decreasing path towards the root. If, however, the destination were instead J, we

must find that $H.topo_order > J.topo_order$, so we must choose the increasing Blue next-hop to J, which is I. In the case, when instead the destination is C, we find that we need to first decrease to avoid using H, so the Blue, first decreasing then increasing, path is selected.



(a) ADAG rooted at R for
a 2-connected graph

Figure 26

5.9. Named Proxy-Nodes

As discussed in Section 11.2 of [I-D.ietf-rtgwg-mrt-frr-architecture], it is necessary to find MRT-Blue and MRT-Red next-hops and MRT-FRR alternates for named proxy-nodes. An example use case is for a router that is not part of that local MRT Island, when there is only partial MRT support in the domain.

5.9.1. Determining Proxy-Node Attachment Routers

Section 11.2 of [I-D.ietf-rtgwg-mrt-frr-architecture] discusses general considerations for determining the two proxy-node attachment routers for a given proxy-node, corresponding to a prefix. A router in the MRT Island that advertises the prefix is a candidate for being a proxy-node attachment router, with the associated named-proxy-cost equal to the advertised cost to the prefix.

An Island Border Router (IBR) is a router in the MRT Island that is connected to an Island Neighbor(IN), which is a router not in the MRT Island but in the same area/level. An (IBR,IN) pair is a candidate for being a proxy-node attachment router, if the shortest path from the IN to the prefix does not enter the MRT Island. A method for identifying such loop-free Island Neighbors(LFINs) is given below. The named-proxy-cost assigned to each (IBR, IN) pair is $\text{cost}(\text{IBR}, \text{IN}) + D_{\text{opt}}(\text{IN}, \text{prefix})$.

From the set of prefix-advertising routers and the set of IBRs with at least one LFIN, the two routers with the lowest named-proxy-cost

are selected. Ties are broken based upon the lowest Router ID. For ease of discussion, the two selected routers will be referred to as proxy-node attachment routers.

5.9.2. Computing if an Island Neighbor (IN) is loop-free

As discussed above, the Island Neighbor needs to be loop-free with respect to the whole MRT Island for the destination. This can be accomplished by running the usual SPF algorithm while keeping track of which shortest paths have passed through the MRT island. Pseudo-code for this is shown in Figure 27. The `Island_Marking_SPF()` is run for each IN that needs to be evaluated for the loop-free condition, with the IN as the `spf_root`. Whether or not an IN is loop-free with respect to the MRT island can then be determined by evaluating `node.PATH_HITS_ISLAND` for each destination of interest.

```

Island_Marking_SPF(spfx_root)
  Initialize spfx_heap to empty
  Initialize nodes' spfx_metric to infinity and next_hops to empty
  and PATH_HITS_ISLAND to false
  spfx_root.spfx_metric = 0
  insert(spfx_heap, spfx_root)
  while (spfx_heap is not empty)
    min_node = remove_lowest(spfx_heap)
    foreach interface intf of min_node
      path_metric = min_node.spfx_metric + intf.metric
      if path_metric < intf.remote_node.spfx_metric
        intf.remote_node.spfx_metric = path_metric
        if min_node is spfx_root
          intf.remote_node.next_hops = make_list(intf)
        else
          intf.remote_node.next_hops = min_node.next_hops
      if intf.remote_node.IN_MRT_ISLAND
        intf.remote_node.PATH_HITS_ISLAND = true
      else
        intf.remote_node.PATH_HITS_ISLAND =
          min_node.PATH_HITS_ISLAND
        insert_or_update(spfx_heap, intf.remote_node)
      else if path_metric == intf.remote_node.spfx_metric
        if min_node is spfx_root
          add_to_list(intf.remote_node.next_hops, intf)
        else
          add_list_to_list(intf.remote_node.next_hops,
                           min_node.next_hops)
      if intf.remote_node.IN_MRT_ISLAND
        intf.remote_node.PATH_HITS_ISLAND = true
      else
        intf.remote_node.PATH_HITS_ISLAND =
          min_node.PATH_HITS_ISLAND

```

Figure 27: Island_Marking_SPF for determining if an Island Neighbor is loop-free

It is also possible that a given prefix is originated by a combination of non-island routers and island routers. The results of the Island_Marking_SPF computation can be used to determine if the shortest path from an IN to reach that prefix hits the MRT island. The shortest path for the IN to reach prefix P is determined by the total cost to reach prefix P, which is the sum of the cost for the IN to reach a prefix-advertising node and the cost with which that node advertises the prefix. The path with the minimum total cost to prefix P is chosen. If the prefix-advertising node for that minimum total cost path has PATH_HITS_ISLAND set to True, then the IN is not loop-free with respect to the MRT Island for reaching prefix P. If

there multiple minimum total cost paths to reach prefix P, then all of the prefix-advertising routers involved in the minimum total cost paths MUST have `PATH_HITS_ISLAND` set to False for the IN to be considered loop-free to reach P.

Note that there are other computations that could be used to determine if paths from a given IN `_might_` pass through the MRT Island for a given prefix or destination. For example, a previous version of this draft specified running the SPF algorithm on modified topology which treats the MRT island as a single node (with intra-island links set to zero cost) in order to provide input to computations to determine if the path from IN to non-island destination hits the MRT island in this modified topology. This computation is enough to guarantee that a path will not hit the MRT island in the original topology. However, it is possible that a path which is disqualified for hitting the MRT island in the modified topology will not actually hit the MRT Island in the original topology. The algorithm described in `Island_Marking_SPF()` above does not modify the original topology, and will only disqualify a path if the actual path does in fact hit the MRT island.

Since all routers need to come to the same conclusion about which routers qualify as LFINS, this specification requires that all routers computing LFINS MUST use an algorithm whose result is identical to that of the `Island_Marking_SPF()` in Figure 27.

5.9.3. Computing MRT Next-Hops for Proxy-Nodes

Determining the MRT next-hops for a proxy-node in the degenerate case where the proxy-node is attached to only one node in the GADAG is trivial, as all needed information can be derived from that proxy node attachment router. If there are multiple interfaces connecting the proxy node to the single proxy node attachment router, then some can be assigned to MRT-Red and others to MRT-Blue.

Now, consider the proxy-node P that is attached to two proxy-node attachment routers. The pseudo-code for `Select_Proxy_Node_NHs(P,S)` in Figure 28 specifies how a computing-router S MUST compute the MRT red and blue next-hops to reach proxy-node P. The proxy-node attachment router with the lower value of `mrt_node_id` (as defined in Figure 15) is assigned to X, and the other proxy-node attachment router is assigned to Y. We will be using the relative order of X,Y, and S in the partial order defined by the GADAG to determine the MRT red and blue next-hops to reach P, so we also define A and B as the order proxies for X and Y, respectively, with respect to S. The order proxies for all nodes with respect to S were already computed in `Compute_MRT_NextHops()`.

```
def Select_Proxy_Node_NHs(P,S):
    if P.pnar1.node.node_id < P.pnar2.node.node_id:
        X = P.pnar1.node
        Y = P.pnar2.node
    else:
        X = P.pnar2.node
        Y = P.pnar1.node
    P.pnar_X = X
    P.pnar_Y = Y
    A = X.order_proxy
    B = Y.order_proxy
    if (A is S.localroot
        and B is S.localroot):
        // case 1.0
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
    if (A is S.localroot
        and B is not S.localroot):
        // case 2.0
        if B.LOWER:
            // case 2.1
            Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
            Copy_List_Items(P.red_next_hops, Y.red_next_hops)
            return
        if B.HIGHER:
            // case 2.2
            Copy_List_Items(P.blue_next_hops, X.red_next_hops)
            Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
            return
        else:
            // case 2.3
            Copy_List_Items(P.blue_next_hops, X.red_next_hops)
            Copy_List_Items(P.red_next_hops, Y.red_next_hops)
            return
    if (A is not S.localroot
        and B is S.localroot):
        // case 3.0
        if A.LOWER:
            // case 3.1
            Copy_List_Items(P.blue_next_hops, X.red_next_hops)
            Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
            return
        if A.HIGHER:
            // case 3.2
            Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
            Copy_List_Items(P.red_next_hops, Y.red_next_hops)
            return
```

```
    else:
        // case 3.3
        Copy_List_Items(P.blue_next_hops, X.red_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
    if (A is not S.localroot
        and B is not S.localroot):
        // case 4.0
        if (S is A.localroot or S is B.localroot):
            // case 4.05
            if A.topo_order < B.topo_order:
                // case 4.05.1
                Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
                Copy_List_Items(P.red_next_hops, Y.red_next_hops)
                return
            else:
                // case 4.05.2
                Copy_List_Items(P.blue_next_hops, X.red_next_hops)
                Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
                return
        if A.LOWER:
            // case 4.1
            if B.HIGHER:
                // case 4.1.1
                Copy_List_Items(P.blue_next_hops, X.red_next_hops)
                Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
                return
            if B.LOWER:
                // case 4.1.2
                if A.topo_order < B.topo_order:
                    // case 4.1.2.1
                    Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
                    Copy_List_Items(P.red_next_hops, Y.red_next_hops)
                    return
                else:
                    // case 4.1.2.2
                    Copy_List_Items(P.blue_next_hops, X.red_next_hops)
                    Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
                    return
            else:
                // case 4.1.3
                Copy_List_Items(P.blue_next_hops, X.red_next_hops)
                Copy_List_Items(P.red_next_hops, Y.red_next_hops)
                return
        if A.HIGHER:
            // case 4.2
            if B.HIGHER:
                // case 4.2.1
```

```

    if A.topo_order < B.topo_order:
        // case 4.2.1.1
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
    else:
        // case 4.2.1.2
        Copy_List_Items(P.blue_next_hops, X.red_next_hops)
        Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
        return
    if B.LOWER:
        // case 4.2.2
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
    else:
        // case 4.2.3
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
        return
else:
    // case 4.3
    if B.LOWER:
        // case 4.3.1
        Copy_List_Items(P.blue_next_hops, X.red_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
    if B.HIGHER:
        // case 4.3.2
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
        return
    else:
        // case 4.3.3
        if A.topo_order < B.topo_order:
            // case 4.3.3.1
            Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
            Copy_List_Items(P.red_next_hops, Y.red_next_hops)
            return
        else:
            // case 4.3.3.2
            Copy_List_Items(P.blue_next_hops, X.red_next_hops)
            Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
            return
assert(False)

```

Figure 28: Select_Proxy_Node_NHs()

It is useful to understand up front that the blue next-hops to reach proxy-node P produced by `Select_Proxy_Node_NHs()` will always be the next-hops that reach proxy-node attachment router X, while the red next-hops to reach proxy-node P will always be the next-hops that reach proxy-node attachment router Y. This is different from the red and blue next-hops produced by `Compute_MRT_NextHops()` where, for example, blue next-hops to a destination that is ordered with respect to the source will always correspond to an INCREASING next-hop on the GADAG. The exact choice of which next-hops chosen by `Select_Proxy_Node_NHs()` as the blue next-hops to reach P (which will necessarily go through X on its way to P) does depend on the GADAG, but the relationship is more complex than was the case with `Compute_MRT_NextHops()`.

There are twenty-one different relative order relationships between A, B and S that `Select_Proxy_Node_NHs()` uses to determine red and blue next-hops to P. This document does not attempt to provide an exhaustive description of each case considered in `Select_Proxy_Node_NHs()`. Instead we provide a high level overview of the different cases, and we consider a few cases in detail to give an example of the reasoning that can be used to understand each case.

At the highest level, `Select_Proxy_Node_NHs()` distinguishes between four different cases depending on whether or not A or B is the localroot for S. For example, for case 4.0, neither A nor B is the localroot for S. Case 4.05 addresses the case where S is the localroot for either A or B, while cases 4.1, 4.2, and 4.3 address the cases where A is ordered lower than S, A is ordered higher than S, or A is unordered with respect to S on the GADAG. In general, each of these cases is then further subdivided into whether or not B is ordered lower than S, B is ordered higher than S, or B is unordered with respect to S. In some cases we also need a further level of discrimination, where we use the topological sort order of A with respect to B.

As a detailed example, let's consider case 4.1 and all of its sub-cases, and explain why the red and blue next-hops to reach P are chosen as they are in `Select_Proxy_Node_NHs()`. In case 4.1, neither A nor B is the localroot for S, S is not the localroot for A or B, and A is ordered lower than S on the GADAG. In this situation, we know that the red path to reach X (as computed in `Compute_MRT_NextHops()`) will follow DECREASING next-hops towards A, while the blue path to reach X will follow INCREASING next-hops to the localroot, and then INCREASING next-hops to A.

Now consider sub-case 4.1.1 where B is ordered higher than S. In this situation, we know that the blue path to reach Y will follow INCREASING next-hops towards B, while the red next-hops to reach Y

will follow DECREASING next-hops to the localroot, and then DECREASING next-hops to B. So to reach X and Y by two disjoint paths, we can choose the red next-hops to X and the blue next-hops to Y. We have chosen the convention that blue next-hops to P are those that pass through X, and red next-hops to P are those that pass through Y, so we can see that case 4.1.1 produces the desired result. Choosing blue to X and red to Y does not produce disjoint paths because the paths intersect at least at the localroot.

Now consider sub-case 4.1.2 where B is ordered lower than S. In this situation, we know that the red path to reach Y will follow DECREASING next-hops towards B, while the BLUE next-hops to reach Y will follow INCREASING next-hops to the localroot, and then INCREASING next-hops to A. The choice here is more difficult than in 4.1.1 because A and B are both on the DECREASING path from S towards the localroot. We want to use the direct DECREASING(red) path to the one that is nearer to S on the GADAG. We get this extra information by comparing the topological sort order of A and B. If $A.topo_order < B.topo_order$, then we use red to Y and blue to X, since the red path to Y will DECREASE to B without hitting A, and the blue path to X will INCREASE to A without hitting B. Instead, if $A.topo_order > B.topo_order$, then we use red to X and blue to Y.

Note that when A is unordered with respect to B, the result of comparing $A.topo_order$ with $B.topo_order$ could be greater than or less than. In this case, the result doesn't matter because either choice (red to Y and blue to X or red to X and blue to Y) would work. What is required is that all nodes in the network give the same result when comparing $A.topo_order$ with $B.topo_order$. This is guaranteed by having all nodes run the same algorithm (`Run_Topological_Sort_GADAG()`) to compute the topological sort order.

Finally we consider case 4.1.3, where B is unordered with respect to S. In this case, the blue path to reach Y will follow the DECREASING next-hops towards the localroot until it reaches some node (K) which is ordered less than B, after which it will take INCREASING next-hops to B. The red path to reach Y will follow the INCREASING next-hops towards the localroot until it reaches some node (L) which is ordered greater than B, after which it will take DECREASING next-hops to B. Both K and A are reached by DECREASING from S, but we don't have information about whether or not that DECREASING path will hit K or A first. Instead, we do know that the INCREASING path from S will hit L before reaching A. Therefore, we use the red path to reach Y and the red path to reach X.

Similar reasoning can be applied to understand the other seventeen cases used in `Select_Proxy_Node_NHs()`. However, cases 2.3 and 3.3 deserve special attention because the correctness of the solution for

these two cases relies on a special property of the GADAGs that we have constructed in this algorithm, a property not shared by all GADAGs in general. Focusing on case 2.3, we consider the case where A is the localroot for S, while B is not, and B is unordered with respect to S. The red path to X DECREASES from S to the localroot A, while the blue path to X INCREASES from S to the localroot A. The blue path to Y DECREASES towards the localroot A until it reaches some node (K) which is ordered less than B, after which the path INCREASES to B. The red path to Y INCREASES towards the localroot A until it reaches some node (L) which is ordered greater than B, after which the path DECREASES to B. It can be shown that for an arbitrary GADAG, with only the ordering relationships computed so far, we don't have enough information to choose a pair of paths to reach X and Y that are guaranteed to be disjoint. In some topologies, A will play the role of K, the first node ordered less than B on the blue path to Y. In other topologies, A will play the role of L, the first node ordered greater than B on the red path to Y. The basic problem is that we cannot distinguish between these two cases based on the ordering relationships.

As discussed Section 5.8, the GADAGs that we construct using the algorithm in this document are not arbitrary GADAGs. They have the additional property that incoming links to a localroot come from only one other node in the same block. This is a result of the method of construction. This additional property guarantees that localroot A will never play the role of L in the red path to Y, since L must have at least two incoming links from different nodes in the same block in the GADAG. This in turn allows `Select_Proxy_Node_NHs()` to choose the red path to Y and the red path to X as the disjoint MRT paths to reach P.

5.9.4. Computing MRT Alternates for Proxy-Nodes

After finding the red and the blue next-hops for a given proxy-node P, it is necessary to know which one of these to use in the case of failure. This can be done by `Select_Alternates_Proxy_Node()`, as shown in the pseudo-code in Figure 29.

```
def Select_Alternates_Proxy_Node(P,F,primary_intf):
    S = primary_intf.local_node
    X = P.pnar_X
    Y = P.pnar_Y
    A = X.order_proxy
    B = Y.order_proxy
    if F is A and F is B:
        return 'PRIM_NH_IS_OP_FOR_BOTH_X_AND_Y'
    if F is A:
        return 'USE_RED'
```

```

if F is B:
    return 'USE_BLUE'

if not In_Common_Block(A, B):
    if In_Common_Block(F, A):
        return 'USE_RED'
    elif In_Common_Block(F, B):
        return 'USE_BLUE'
    else:
        return 'USE_RED_OR_BLUE'
if (not In_Common_Block(F, A)
    and not In_Common_Block(F, B)):
    return 'USE_RED_OR_BLUE'

alt_to_X = Select_Alternates(X, F, primary_intf)
alt_to_Y = Select_Alternates(Y, F, primary_intf)

if (alt_to_X == 'USE_RED_OR_BLUE'
    and alt_to_Y == 'USE_RED_OR_BLUE'):
    return 'USE_RED_OR_BLUE'
if alt_to_X == 'USE_RED_OR_BLUE':
    return 'USE_BLUE'
if alt_to_Y == 'USE_RED_OR_BLUE':
    return 'USE_RED'

if (A is S.localroot
    and B is S.localroot):
    // case 1.0
    if (alt_to_X == 'USE_BLUE' and alt_to_Y == 'USE_RED'):
        return 'USE_RED_OR_BLUE'
    if alt_to_X == 'USE_BLUE':
        return 'USE_BLUE'
    if alt_to_Y == 'USE_RED':
        return 'USE_RED'
    assert(False)
if (A is S.localroot
    and B is not S.localroot):
    // case 2.0
    if B.LOWER:
        // case 2.1
        if (alt_to_X == 'USE_BLUE' and alt_to_Y == 'USE_RED'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_BLUE':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_RED':
            return 'USE_RED'
        assert(False)
    if B.HIGHER:

```

```
// case 2.2
if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_BLUE'):
    return 'USE_RED_OR_BLUE'
if alt_to_X == 'USE_RED':
    return 'USE_BLUE'
if alt_to_Y == 'USE_BLUE':
    return 'USE_RED'
assert(False)
else:
    // case 2.3
    if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_RED'):
        return 'USE_RED_OR_BLUE'
    if alt_to_X == 'USE_RED':
        return 'USE_BLUE'
    if alt_to_Y == 'USE_RED':
        return 'USE_RED'
    assert(False)
if (A is not S.localroot
and B is S.localroot):
    // case 3.0
    if A.LOWER:
        // case 3.1
        if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_BLUE'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_RED':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_BLUE':
            return 'USE_RED'
        assert(False)
    if A.HIGHER:
        // case 3.2
        if (alt_to_X == 'USE_BLUE' and alt_to_Y == 'USE_RED'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_BLUE':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_RED':
            return 'USE_RED'
        assert(False)
    else:
        // case 3.3
        if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_RED'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_RED':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_RED':
            return 'USE_RED'
        assert(False)
if (A is not S.localroot
```

```
and B is not S.localroot):
// case 4.0
if (S is A.localroot or S is B.localroot):
    // case 4.05
    if A.topo_order < B.topo_order:
        // case 4.05.1
        if (alt_to_X == 'USE_BLUE' and alt_to_Y == 'USE_RED'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_BLUE':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_RED':
            return 'USE_RED'
        assert(False)
    else:
        // case 4.05.2
        if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_BLUE'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_RED':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_BLUE':
            return 'USE_RED'
        assert(False)
if A.LOWER:
    // case 4.1
    if B.HIGHER:
        // case 4.1.1
        if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_BLUE'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_RED':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_BLUE':
            return 'USE_RED'
        assert(False)
    if B.LOWER:
        // case 4.1.2
        if A.topo_order < B.topo_order:
            // case 4.1.2.1
            if (alt_to_X == 'USE_BLUE'
                and alt_to_Y == 'USE_RED'):
                return 'USE_RED_OR_BLUE'
            if alt_to_X == 'USE_BLUE':
                return 'USE_BLUE'
            if alt_to_Y == 'USE_RED':
                return 'USE_RED'
            assert(False)
        else:
            // case 4.1.2.2
            if (alt_to_X == 'USE_RED'
```

```

        and alt_to_Y == 'USE_BLUE'):
            return 'USE_RED_OR_BLUE'
    if alt_to_X == 'USE_RED':
        return 'USE_BLUE'
    if alt_to_Y == 'USE_BLUE':
        return 'USE_RED'
    assert(False)
else:
    // case 4.1.3
    if (F.LOWER and not F.HIGHER
        and F.topo_order > A.topo_order):
        // case 4.1.3.1
        return 'USE_RED'
    else:
        // case 4.1.3.2
        return 'USE_BLUE'
if A.HIGHER:
    // case 4.2
    if B.HIGHER:
        // case 4.2.1
        if A.topo_order < B.topo_order:
            // case 4.2.1.1
            if (alt_to_X == 'USE_BLUE'
                and alt_to_Y == 'USE_RED'):
                return 'USE_RED_OR_BLUE'
            if alt_to_X == 'USE_BLUE':
                return 'USE_BLUE'
            if alt_to_Y == 'USE_RED':
                return 'USE_RED'
            assert(False)
        else:
            // case 4.2.1.2
            if (alt_to_X == 'USE_RED'
                and alt_to_Y == 'USE_BLUE'):
                return 'USE_RED_OR_BLUE'
            if alt_to_X == 'USE_RED':
                return 'USE_BLUE'
            if alt_to_Y == 'USE_BLUE':
                return 'USE_RED'
            assert(False)
    if B.LOWER:
        // case 4.2.2
        if (alt_to_X == 'USE_BLUE'
            and alt_to_Y == 'USE_RED'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_BLUE':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_RED':

```

```
        return 'USE_RED'
    assert(False)
else:
    // case 4.2.3
    if (F.HIGHER and not F.LOWER
        and F.topo_order < A.topo_order):
        return 'USE_RED'
    else:
        return 'USE_BLUE'
else:
    // case 4.3
    if B.LOWER:
        // case 4.3.1
        if (F.LOWER and not F.HIGHER
            and F.topo_order > B.topo_order):
            return 'USE_BLUE'
        else:
            return 'USE_RED'
    if B.HIGHER:
        // case 4.3.2
        if (F.HIGHER and not F.LOWER
            and F.topo_order < B.topo_order):
            return 'USE_BLUE'
        else:
            return 'USE_RED'
    else:
        // case 4.3.3
        if A.topo_order < B.topo_order:
            // case 4.3.3.1
            if (alt_to_X == 'USE_BLUE'
                and alt_to_Y == 'USE_RED'):
                return 'USE_RED_OR_BLUE'
            if alt_to_X == 'USE_BLUE':
                return 'USE_BLUE'
            if alt_to_Y == 'USE_RED':
                return 'USE_RED'
            assert(False)
        else:
            // case 4.3.3.2
            if (alt_to_X == 'USE_RED'
                and alt_to_Y == 'USE_BLUE'):
                return 'USE_RED_OR_BLUE'
            if alt_to_X == 'USE_RED':
                return 'USE_BLUE'
            if alt_to_Y == 'USE_BLUE':
                return 'USE_RED'
            assert(False)
assert(False)
```

Figure 29: `Select_Alternates_Proxy_Node()`

`Select_Alternates_Proxy_Node(P,F,primary_intf)` determines whether it is safe to use the blue path to P (which goes through X), the red path to P (which goes through Y), or either, when the `primary_intf` to node F (and possibly node F) fails. The basic approach is to run `Select_Alternates(X,F,primary_intf)` and `Select_Alternates(Y,F,primary_intf)` to determine which of the two MRT paths to X and which of the two MRT paths to Y is safe to use in the event of the failure of F. In general, we will find that if it is safe to use a particular path to X or Y when F fails, and `Select_Proxy_Node_NHs()` used that path when constructing the red or blue path to reach P, then it will also be safe to use that path to reach P when F fails. This rule has one exception which is covered below. First, we give a concrete example of how `Select_Alternates_Proxy_Node()` works in the common case.

The twenty one ordering relationships used in `Select_Proxy_Node_NHs()` are repeated in `Select_Alternates_Proxy_Node()`. We focus on case 4.1.1 to give a detailed example of the reasoning used in `Select_Alternates_Proxy_Node()`. In `Select_Proxy_Node_NHs()`, we determined for case 4.1.1 that the red next-hops to X and the blue next-hops to Y allow us to reach X and Y by disjoint paths, and are thus the blue and red next-hops to reach P. Therefore, if we run `Select_Alternates(X, F, primary_intf)` and we find that it is safe to `USE_RED` to reach X, then we also conclude that it is safe to use the MRT path through X to reach P (the blue path to P) when F fails. Similarly, if run `Select_Alternates(X, F, primary_intf)` and we find that it is safe to `USE_BLUE` to reach Y, then we also conclude that it is safe to use the MRT path through Y to reach P (the red path to P) when F fails. If both of the paths that were used in `Select_Proxy_Node_NHs()` to construct the blue and red paths to P are found to be safe to use to reach X and Y, then we conclude that we can use either the red or the blue path to P.

This simple reasoning gives the correct answer in most of the cases. However, additional logic is needed when either A or B (but not both A and B) is unordered with respect to S. This applies to cases 4.1.3, 4.2.3, 4.3.1, and 4.3.2. Looking at case 4.1.3 in more detail, A is ordered less than S, but B is unordered with respect to S. In the discussion of case 4.1.3 above, we saw that `Select_Proxy_Node_NHs()` chose the red path to reach Y and the red path to reach X. We also saw that the red path to reach Y will follow the `INCREASING` next-hops towards the localroot until it reaches some node (L) which is ordered greater than B, after which it will take `DECREASING` next-hops to B. The problem is that the red path to reach P (the one that goes through Y) won't necessarily be the same as the red path to reach Y. This is because the next-hop

that node L computes for its red next-hop to reach P may be different from the next-hop it computes for its red next-hop to reach Y. This is because B is ordered lower than L, so L applies case 4.1.2 of `Select_Proxy_Node_NHs()` in order to determine its next-hops to reach P. If $A.topo_order < B.topo_order$ (case 4.1.2.1), then L will choose DECREASING next-hops directly to B, which is the same result that L computes in `Compute_MRT_NextHops()` to reach Y. However, if $A.topo_order > B.topo_order$ (case 4.1.2.2), then L will choose INCREASING next-hops to reach B, which is different from what L computes in `Compute_MRT_NextHops()` to reach Y. So testing the safety of the path for S to reach Y on failure of F as a surrogate for the safety of using the red path to reach P is not reliable in this case. It is possible to construct topologies where the red path to P hits F even though the red path to Y does not hit F.

Fortunately there is enough information in the order relationships that we have already computed to still figure out which alternate to choose in these four cases. The basic idea is to always choose the path involving the ordered node, unless that path would hit F. Returning to case 4.1.3, we see that since A is ordered lower than S, the only way for S to hit F using a simple DECREASING path to A is for F to lie between A and S on the GADAG. This scenario is covered by requiring that F be lower than S (but not also higher than S) and that $F.topo_order > A.topo_order$ in case 4.1.3.1.

We just need to confirm that it is safe to use the path involving B in this scenario. In case 4.1.3.1, either F is between A and S on the GADAG, or F is unordered with respect to A and lies on the DECREASING path from S to the localroot. When F is between A and S on the GADAG, then the path through B chosen to avoid A in `Select_Proxy_Node_NHs()` will also avoid F. When F is unordered with respect to A and lies on the DECREASING path from S to the localroot, then we consider two cases. Either $F.topo_order < B.topo_order$ or $F.topo_order > B.topo_order$. In the first case, since $F.topo_order < B.topo_order$ and $F.topo_order > A.topo_order$, it must be the case that $A.topo_order < B.topo_order$. Therefore, L will choose DECREASING next-hops directly to B (case 4.1.2.1), which cannot hit F since $F.topo_order < B.topo_order$. In the second case, where $F.topo_order > B.topo_order$, the only way for the path involving B to hit F is if it DECREASES from L to B through F, ie. it must be that $L >> F >> B$. However, since $S >> F$, this would imply that $S >> B$. However, we know that S is unordered with respect to B, so the second case cannot occur. So we have demonstrated that the red path to P (which goes via B and Y) is safe to use under the conditions of 4.1.3.1. Similar reasoning can be applied to the other three special cases where either A or B is unordered with respect to S.

6. MRT Lowpoint Algorithm: Next-hop conformance

This specification defines the MRT Lowpoint Algorithm, which include the construction of a common GADAG and the computation of MRT-Red and MRT-Blue next-hops to each node in the graph. An implementation MAY select any subset of next-hops for MRT-Red and MRT-Blue that respect the available nodes that are described in Section 5.7 for each of the MRT-Red and MRT-Blue and the selected next-hops are further along in the interval of allowed nodes towards the destination.

For example, the MRT-Blue next-hops used when the destination $Y \gg X$, the computing router, MUST be one or more nodes, T, whose `topo_order` is in the interval $[X.topo_order, Y.topo_order]$ and where $Y \gg T$ or Y is T. Similarly, the MRT-Red next-hops MUST be have a `topo_order` in the interval $[R-small.topo_order, X.topo_order]$ or $[Y.topo_order, R-big.topo_order]$.

Implementations SHOULD implement the `Select_Alternates()` function to pick an MRT-FRR alternate.

7. Broadcast interfaces

When broadcast interfaces are used to connect nodes, the broadcast network MUST be represented as a pseudonode, where each real node connects to the pseudonode. The interface metric in the direction from real node to pseudonode is the non-zero interface metric, while the interface metric in the direction from the pseudonode to the real node is set to zero. This is consistent with the way that broadcast interfaces are represented as pseudonodes in IS-IS and OSPF.

Pseudonodes MUST be treated as equivalent to real nodes in the network graph used in the MRT algorithm with a few exceptions detailed below.

The pseudonodes MUST be included in the computation of the GADAG. The neighbors of the pseudonode need to know the `mrt_node_id` of the pseudonode in order to consistently order interfaces, which is needed to compute the GADAG. The `mrt_node_id` for IS-IS is the 7 octet neighbor system ID and pseudonode number in TLV #22 or TLV#222. The `mrt_node_id` for OSPFv2 is the 4 octet interface address of the Designated Router found in the Link ID field for the link type 2 (transit network) in the Router-LSA. The `mrt_node_id` for OSPFv3 is the 4 octet interface address of the Designated Router found in the Neighbor Interface ID field for the link type 2 (transit network) in the Router-LSA. pseudonodes MUST NOT be considered as candidates for GADAG root selection. Note that this is different from the Neighbor Router ID field used for the `mrt_node_id` for point-to-point links in OSPFv3 Router-LSAs given in Figure 15.

Pseudonodes MUST NOT be considered as candidates for selection as GADAG root. This rule is intended to result in a more stable network- wide selection of GADAG root by removing the possibility that the change of Designated Router or Designated Intermediate System on a broadcast network can result in a change of GADAG root.

7.1. Computing MRT next-hops on broadcast networks

The pseudonode does not correspond to an real node, so it is not actually involved in forwarding. A real node on a broadcast network cannot simply forward traffic to the broadcast network. It must specify another real node on the broadcast network as the next-hop. On a network graph where a broadcast network is represented by a pseudonode, this means that if a real node determines that the next-hop to reach a given destination is a pseudonode, it must also determine the next-next-hop for that destination in the network graph, which corresponds to a real node attached to the broadcast network.

It is interesting to note that this issue is not unique to the MRT algorithm, but is also encountered in normal SPF computations for IGP. Section 16.1.1 of [RFC2328] describes how this is done for OSPF. As OSPF runs Dijkstra's algorithm, whenever a shorter path is found reach a real destination node, and the shorter path is one hop from the computing routing, and that one hop is a pseudonode, then the next-hop for that destination is taken from the interface IP address in the Router-LSA correspond to the link to the real destination node

For IS-IS, in the example pseudo-code implementation of Dijkstra's algorithm in Annex C of [ISO10589-Second-Edition] whenever the algorithm encounters an adjacency from a real node to a pseudonode, it gets converted to a set of adjacencies from the real node to the neighbors of the pseudonode. In this way, the computed next-hops point all the way to the real node, and not the pseudonode.

We could avoid the problem of determining next-hops across pseudonodes in MRT by converting the pseudonode representation of broadcast networks to a full mesh of links between real nodes on the same network. However, if we make that conversion before computing the GADAG, we lose information about which links actually correspond to a single physical interface into the broadcast network. This could result computing red and blue next-hops that use the same broadcast interface, in which case neither the red nor the blue next-hop would be usable as an alternate on failure of the broadcast interface.

Instead, we take the following approach, which maintains the property that either the red and blue next-hop will avoid the broadcast network, if topologically allowed. We run the MRT algorithm treating the pseudonodes as equivalent to real nodes in the network graph, with the exceptions noted above. In addition to running the MRT algorithm from the point of view of itself, a computing router connected to a pseudonode MUST also run the MRT algorithm from the point of view of each of its pseudonode neighbors. For example, if a computing router S determines that its MRT red next-hop to reach a destination D is a pseudonode P, S looks at its MRT algorithm computation from P's point of view to determine P's red next-hop to reach D, say interface 1 on node X. S now knows that its real red next-hop to reach D is interface 1 on node X on the broadcast network represented by P, and can install the corresponding entry in its FIB.

7.2. Using MRT next-hops as alternates in the event of failures on broadcast networks

In the previous section, we specified how to compute MRT next-hops when broadcast networks are involved. In this section, we discuss how a PLR can use those MRT next-hops in the event of failures involving broadcast networks.

A PLR attached to a broadcast network running only OSPF or IS-IS with large Hello intervals has limited ability to quickly detect failures on a broadcast network. The only failure mode that can be quickly detected is the failure of the physical interface connecting the PLR to the broadcast network. For the failure of the interface connecting the PLR to the broadcast network, the alternate that avoids the broadcast network can be computed by using the broadcast network pseudonode as F, the primary next-hop node, in `Select_Alternates()`. This will choose an alternate path that avoids the broadcast network. However, the alternate path will not necessarily avoid all of the real nodes connected to the broadcast network. This is because we have used the pseudonode to represent the broadcast network. And we have enforced the node-protecting property of MRT on the pseudonode to provide protection against failure of the broadcast network, not the real next-hop nodes on the broadcast network. This is the best that we can hope to do if failure of the broadcast interface is the only failure mode that the PLR can respond to.

We can improve on this if the PLR also has the ability to quickly detect a lack of connectivity across the broadcast network to a given IP-layer node. This can be accomplished by running BFD between all pairs of IGP neighbors on the broadcast network. Note that in the case of OSPF, this would require establishing BFD sessions between all pairs of neighbors in the 2-WAY state. When the PLR can quickly

detect the failure of a particular next-hop across a broadcast network, then the PLR can be more selective in its choice of alternates. For example, when the PLR observes that connectivity to an IP-layer node on a broadcast network has failed, the PLR may choose to still use the broadcast network to reach other IP-layer nodes which are still reachable. Or if the PLR observes that connectivity has failed to several IP-layer nodes on the same broadcast network, it may choose to treat the entire broadcast network as failed. The choice of MRT alternates by a PLR for a particular set of failure conditions is a local decision, since it does not require coordination with other nodes.

8. Evaluation of Alternative Methods for Constructing GADAGs

This document specifies the MRT Lowpoint algorithm. One component of the algorithm involves constructing a common GADAG based on the network topology. The MRT Lowpoint algorithm computes the GADAG using the method described in Section 5.5. This method aims to minimize the amount of computation required to compute the GADAG. In the process of developing the MRT Lowpoint algorithm, two alternative methods for constructing GADAGs were also considered. These alternative methods are described in Appendix B and Appendix C. In general, these other two methods require more computation to compute the GADAG. The analysis below was performed to determine if the alternative GADAG construction methods produce shorter MRT alternate paths in real network topologies, and if so, to what extent.

Figure 30 compares results obtained using the three different methods for constructing GADAGs on five different service provider network topologies. MRT_LOWPOINT indicates the method specified in Section 5.5, while MRT_SPF and MRT_HYBRID indicate the methods specified in Appendix B and Appendix C, respectively. The columns on the right present the distribution of alternate path lengths for each GADAG construction method. Each MRT computation was performed using a same GADAG root chosen based on centrality.

For three of the topologies analyzed (T201, T206, and T211), the use of MRT_SPF or MRT_HYBRID methods does not appear to provide a significantly shorter alternate path lengths compared to the MRT_LOWPOINT method. However, for two of the topologies (T216 and T219), the use of the MRT_SPF method resulted in noticeably shorter alternate path lengths than the use of the MRT_LOWPOINT or MRT_HYBRID methods.

It was decided to use the MRT_LOWPOINT method to construct the GADAG in the algorithm specified in this draft, in order to initially offer an algorithm with lower computational requirements. These results indicate that in the future it may be useful to evaluate and

potentially specify other MRT algorithm variants that use different GADAG construction methods.

Topology name GADAG construction method	percentage of failure scenarios protected by an alternate N hops longer than the primary path								
	0-1	2-3	4-5	6-7	8-9	10-11	12-13	14-15	no alt <16
T201(avg primary hops=3.5)									
MRT_HYBRID	33	26	23	6	3				
MRT_SPF	33	36	23	6	3				
MRT_LOWPOINT	33	36	23	6	3				
T206(avg primary hops=3.7)									
MRT_HYBRID	50	35	13	2					
MRT_SPF	50	35	13	2					
MRT_LOWPOINT	55	32	13						
T211(avg primary hops=3.3)									
MRT_HYBRID	86	14							
MRT_SPF	86	14							
MRT_LOWPOINT	85	15	1						
T216(avg primary hops=5.2)									
MRT_HYBRID	23	22	18	13	10	7	4	2	2
MRT_SPF	35	32	19	9	3	1			
MRT_LOWPOINT	28	25	18	11	7	6	3	2	1
T219(avg primary hops=7.7)									
MRT_HYBRID	20	16	13	10	7	5	5	5	3
MRT_SPF	31	23	19	12	7	4	2	1	
MRT_LOWPOINT	19	14	15	12	10	8	7	6	10

Figure 30

9. Implementation Status

[RFC Editor: please remove this section prior to publication.]

Please see [I-D.ietf-rtgwg-mrt-frr-architecture] for details on implementation status.

10. Operational Considerations

This section discusses operational considerations related to the the MRT Lowpoint algorithm and other potential MRT algorithm variants. For a discussion of operational considerations related to MRT-FRR in general, see the Operational Considerations section of [I-D.ietf-rtgwg-mrt-frr-architecture].

10.1. GADAG Root Selection

The Default MRT Profile uses the GADAG Root Selection Priority advertised by routers as the primary criterion for selecting the GADAG root. It is RECOMMENDED that an operator designate a set of routers as good choices for selection as GADAG root by setting the GADAG Root Selection Priority for that set of routers to lower (more preferred) numerical values. Criteria for making this designation are discussed below.

Analysis has shown that the centrality of a router can have a significant impact on the lengths of the alternate paths computed. Therefore, it is RECOMMENDED that off-line analysis that considers the centrality of a router be used to help determine how good a choice a particular router is for the role of GADAG root.

If the router currently selected as GADAG root becomes unreachable in the IGP topology, then a new GADAG root will be selected. Changing the GADAG root can change the overall structure of the GADAG as well the paths of the red and blue MRT trees built using that GADAG. In order to minimize change in the associated red and blue MRT forwarding entries that can result from changing the GADAG root, it is RECOMMENDED that operators prioritize for selection as GADAG root those routers that are expected to consistently remain part of the IGP topology.

10.2. Destination-rooted GADAGs

The MRT Lowpoint algorithm constructs a single GADAG rooted at a single node selected as the GADAG root. It is also possible to construct a different GADAG for each destination, with the GADAG rooted at the destination. A router can compute the MRT-Red and MRT-Blue next-hops for that destination based on the GADAG rooted at that destination. Building a different GADAG for each destination is computationally more expensive, but it may give somewhat shorter alternate paths. Using destination-rooted GADAGs would require a new MRT profile to be created with a new MRT algorithm specification, since all routers in the MRT Island would need to use destination-rooted GADAGs.

11. Acknowledgements

The authors would like to thank Shraddha Hegde, Eric Wu, Janos Farkas, Stewart Bryant, and Alvaro Retana for their suggestions and review. We would also like to thank Anil Kumar SN for his assistance in clarifying the algorithm description and pseudo-code.

12. IANA Considerations

This document includes no request to IANA.

13. Security Considerations

The algorithm described in this document does not introduce new security concerns beyond those already discussed in the document describing the MRT FRR architecture [I-D.ietf-rtgwg-mrt-frr-architecture].

14. References

14.1. Normative References

- [I-D.ietf-rtgwg-mrt-frr-architecture]
Atlas, A., Kebler, R., Bowers, C., Envedi, G., Csaszar, A., Tantsura, J., and R. White, "An Architecture for IP/LDP Fast-Reroute Using Maximally Redundant Trees", draft-ietf-rtgwg-mrt-frr-architecture-07 (work in progress), October 2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

14.2. Informative References

- [EnvediThesis]
Envedi, G., "Novel Algorithms for IP Fast Reroute", Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics Ph.D. Thesis, February 2011, <http://www.omikk.bme.hu/collection/s/phd/Villamosmernoki_es_Informatikai_Kar/2011/Envedi_Gabor/ertekezes.pdf>.

- [IEEE8021Qca]
IEEE 802.1, "IEEE 802.1Qca Bridges and Bridged Networks - Amendment: Path Control and Reservation - Draft 2.1", (work in progress), June 24, 2015, <<http://www.ieee802.org/1/pages/802.1ca.html>>.
- [ISO10589-Second-Edition]
International Organization for Standardization, "Intermediate system to Intermediate system intra-domain routeing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode Network Service (ISO 8473)", ISO/IEC 10589:2002, Second Edition, Nov. 2002.
- [Kahn_1962_topo_sort]
Kahn, A., "Topological sorting of large networks", Communications of the ACM, Volume 5, Issue 11, Nov 1962, <<http://dl.acm.org/citation.cfm?doid=368996.369025>>.
- [MRTLlinear]
Enyedi, G., Retvari, G., and A. Csaszar, "On Finding Maximally Redundant Trees in Strictly Linear Time", IEEE Symposium on Computers and Communications (ISCC), 2009, <<http://opti.tmit.bme.hu/~enyedi/ipfrr/distMaxRedTree.pdf>>.
- [RFC2328] Moy, J., "OSPF Version 2", STD 54, RFC 2328, DOI 10.17487/RFC2328, April 1998, <<http://www.rfc-editor.org/info/rfc2328>>.
- [RFC5120] Przygienda, T., Shen, N., and N. Sheth, "M-ISIS: Multi Topology (MT) Routing in Intermediate System to Intermediate Systems (IS-ISs)", RFC 5120, DOI 10.17487/RFC5120, February 2008, <<http://www.rfc-editor.org/info/rfc5120>>.
- [RFC7490] Bryant, S., Filsfils, C., Previdi, S., Shand, M., and N. So, "Remote Loop-Free Alternate (LFA) Fast Reroute (FRR)", RFC 7490, DOI 10.17487/RFC7490, April 2015, <<http://www.rfc-editor.org/info/rfc7490>>.

Appendix A. Python Implementation of MRT Lowpoint Algorithm

Below is Python code implementing the MRT Lowpoint algorithm specified in this document. In order to avoid the page breaks in the .txt version of the draft, one can cut and paste the Python code from the .xml version. The code is also posted on Github.

While this Python code is believed to correctly implement the pseudo-code description of the algorithm, in the event of a difference, the pseudo-code description should be considered normative.

<CODE BEGINS>

```
# This program has been tested to run on Python 2.6 and 2.7
# (specifically Python 2.6.6 and 2.7.8 were tested).
# The program has known incompatibilities with Python 3.X.

# When executed, this program will generate a text file describing
# an example topology. It then reads that text file back in as input
# to create the example topology, and runs the MRT algorithm. This
# was done to simplify the inclusion of the program as a single text
# file that can be extracted from the IETF draft.

# The output of the program is four text files containing a description
# of the GADAG, the blue and red MRTs for all destinations, and the
# MRT alternates for all failures.
```

```
import random
import os.path
import heapq
```

```
# simple Class definitions allow structure-like dot notation for
# variables and a convenient place to initialize those variables.
```

```
class Topology:
```

```
    def __init__(self):
        self.gadag_root = None
        self.node_list = []
        self.node_dict = {}
        self.test_gr = None
        self.island_node_list_for_test_gr = []
        self.stored_named_proxy_dict = {}
        self.init_new_computing_router()
    def init_new_computing_router(self):
        self.island_node_list = []
        self.named_proxy_dict = {}
```

```
class Node:
```

```
    def __init__(self):
        self.node_id = None
        self.intf_list = []
        self.profile_id_list = [0]
        self.GR_sel_priority = 128
        self.blue_next_hops_dict = {}
        self.red_next_hops_dict = {}
        self.blue_to_green_nh_dict = {}
        self.red_to_green_nh_dict = {}
```

```
        self.prefix_cost_dict = {}
        self.pnh_dict = {}
        self.alt_dict = {}
        self.init_new_computing_router()
    def init_new_computing_router(self):
        self.island_intf_list = []
        self.IN_MRT_ISLAND = False
        self.IN_GADAG = False
        self.dfs_number = None
        self.dfs_parent = None
        self.dfs_parent_intf = None
        self.dfs_child_list = []
        self.lowpoint_number = None
        self.lowpoint_parent = None
        self.lowpoint_parent_intf = None
        self.localroot = None
        self.block_id = None
        self.IS_CUT_VERTEX = False
        self.blue_next_hops = []
        self.red_next_hops = []
        self.primary_next_hops = []
        self.alt_list = []

class Interface:
    def __init__(self):
        self.metric = None
        self.area = None
        self.MRT_INELIGIBLE = False
        self.IGP_EXCLUDED = False
        self.SIMULATION_OUTGOING = False
        self.init_new_computing_router()
    def init_new_computing_router(self):
        self.UNDIRECTED = True
        self.INCOMING = False
        self.OUTGOING = False
        self.INCOMING_STORED = False
        self.OUTGOING_STORED = False
        self.IN_MRT_ISLAND = False
        self.PROCESSED = False

class Bundle:
    def __init__(self):
        self.UNDIRECTED = True
        self.OUTGOING = False
        self.INCOMING = False

class Alternate:
    def __init__(self):
```

```
        self.failed_intf = None
        self.red_or_blue = None
        self.nh_list = []
        self.fec = 'NO_ALTERNATE'
        self.prot = 'NO_PROTECTION'
        self.info = 'NONE'

class Proxy_Node_Attachment_Router:
    def __init__(self):
        self.prefix = None
        self.node = None
        self.named_proxy_cost = None
        self.min_lfin = None
        self.nh_intf_list = []

class Named_Proxy_Node:
    def __init__(self):
        self.node_id = None #this is the prefix_id
        self.node_prefix_cost_list = []
        self.lfin_list = []
        self.pnar1 = None
        self.pnar2 = None
        self.pnar_X = None
        self.pnar_Y = None
        self.blue_next_hops = []
        self.red_next_hops = []
        self.primary_next_hops = []
        self.blue_next_hops_dict = {}
        self.red_next_hops_dict = {}
        self.pnh_dict = {}
        self.alt_dict = {}

def Interface_Compare(intf_a, intf_b):
    if intf_a.metric < intf_b.metric:
        return -1
    if intf_b.metric < intf_a.metric:
        return 1
    if intf_a.remote_node.node_id < intf_b.remote_node.node_id:
        return -1
    if intf_b.remote_node.node_id < intf_a.remote_node.node_id:
        return 1
    return 0

def Sort_Interfaces(topo):
    for node in topo.island_node_list:
        node.island_intf_list.sort(Interface_Compare)

def Reset_Computed_Node_and_Intf_Values(topo):
```

```
    topo.init_new_computing_router()
    for node in topo.node_list:
        node.init_new_computing_router()
        for intf in node.intf_list:
            intf.init_new_computing_router()

# This function takes a file with links represented by 2-digit
# numbers in the format:
# 01,05,10
# 05,02,30
# 02,01,15
# which represents a triangle topology with nodes 01, 05, and 02
# and symmetric metrics of 10, 30, and 15.

# Inclusion of a fourth column makes the metrics for the link
# asymmetric. An entry of:
# 02,07,10,15
# creates a link from node 02 to 07 with metrics 10 and 15.
def Create_Topology_From_File(filename):
    topo = Topology()
    node_id_set= set()
    cols_list = []
    # on first pass just create nodes
    with open(filename + '.csv') as topo_file:
        for line in topo_file:
            line = line.rstrip('\r\n')
            cols=line.split(',')
            cols_list.append(cols)
            nodea_node_id = int(cols[0])
            nodeb_node_id = int(cols[1])
            if (nodea_node_id > 999 or nodeb_node_id > 999):
                print("node_id must be between 0 and 999.")
                print("exiting.")
                exit()
            node_id_set.add(nodea_node_id)
            node_id_set.add(nodeb_node_id)
    for node_id in node_id_set:
        node = Node()
        node.node_id = node_id
        topo.node_list.append(node)
        topo.node_dict[node_id] = node
    # on second pass create interfaces
    for cols in cols_list:
        nodea_node_id = int(cols[0])
        nodeb_node_id = int(cols[1])
        metric = int(cols[2])
        reverse_metric = int(cols[2])
        if len(cols) > 3:
```

```

        reverse_metric=int(cols[3])
        nodea = topo.node_dict[nodea_node_id]
        nodeb = topo.node_dict[nodeb_node_id]
        nodea_intf = Interface()
        nodea_intf.metric = metric
        nodea_intf.area = 0
        nodeb_intf = Interface()
        nodeb_intf.metric = reverse_metric
        nodeb_intf.area = 0
        nodea_intf.remote_intf = nodeb_intf
        nodeb_intf.remote_intf = nodea_intf
        nodea_intf.remote_node = nodeb
        nodeb_intf.remote_node = nodea
        nodea_intf.local_node = nodea
        nodeb_intf.local_node = nodeb
        nodea_intf.link_data = len(nodea_intf_list)
        nodeb_intf.link_data = len(nodeb_intf_list)
        nodea_intf_list.append(nodea_intf)
        nodeb_intf_list.append(nodeb_intf)
    return topo

def MRT_Island_Identification(topo, computing_rtr, profile_id, area):
    if profile_id in computing_rtr.profile_id_list:
        computing_rtr.IN_MRT_ISLAND = True
        explore_list = [computing_rtr]
    else:
        return
    while explore_list != []:
        next_rtr = explore_list.pop()
        for intf in next_rtr.intf_list:
            if ( (not intf.MRT_INELIGIBLE)
                and (not intf.remote_intf.MRT_INELIGIBLE)
                and (not intf.IGP_EXCLUDED) and intf.area == area
                and (profile_id in intf.remote_node.profile_id_list)):
                intf.IN_MRT_ISLAND = True
                intf.remote_intf.IN_MRT_ISLAND = True
                if (not intf.remote_node.IN_MRT_ISLAND):
                    intf.remote_node.IN_MRT_ISLAND = True
                    explore_list.append(intf.remote_node)

def Compute_Island_Node_List_For_Test_GR(topo, test_gr):
    Reset_Computed_Node_and_Intf_Values(topo)
    topo.test_gr = topo.node_dict[test_gr]
    MRT_Island_Identification(topo, topo.test_gr, 0, 0)
    for node in topo.node_list:
        if node.IN_MRT_ISLAND:
            topo.island_node_list_for_test_gr.append(node)

```

```

def Set_Island_Intf_and_Node_Lists(topo):
    for node in topo.node_list:
        if node.IN_MRT_ISLAND:
            topo.island_node_list.append(node)
            for intf in node.intf_list:
                if intf.IN_MRT_ISLAND:
                    node.island_intf_list.append(intf)

global_dfs_number = None

def Lowpoint_Visit(x, parent, intf_p_to_x):
    global global_dfs_number
    x.dfs_number = global_dfs_number
    x.lowpoint_number = x.dfs_number
    global_dfs_number += 1
    x.dfs_parent = parent
    if intf_p_to_x == None:
        x.dfs_parent_intf = None
    else:
        x.dfs_parent_intf = intf_p_to_x.remote_intf
    x.lowpoint_parent = None
    if parent != None:
        parent.dfs_child_list.append(x)
    for intf in x.island_intf_list:
        if intf.remote_node.dfs_number == None:
            Lowpoint_Visit(intf.remote_node, x, intf)
            if intf.remote_node.lowpoint_number < x.lowpoint_number:
                x.lowpoint_number = intf.remote_node.lowpoint_number
                x.lowpoint_parent = intf.remote_node
                x.lowpoint_parent_intf = intf
        else:
            if intf.remote_node is not parent:
                if intf.remote_node.dfs_number < x.lowpoint_number:
                    x.lowpoint_number = intf.remote_node.dfs_number
                    x.lowpoint_parent = intf.remote_node
                    x.lowpoint_parent_intf = intf

def Run_Lowpoint(topo):
    global global_dfs_number
    global_dfs_number = 0
    Lowpoint_Visit(topo.gadag_root, None, None)

max_block_id = None

def Assign_Block_ID(x, cur_block_id):
    global max_block_id
    x.block_id = cur_block_id
    for c in x.dfs_child_list:

```

```

        if (c.localroot is x):
            max_block_id += 1
            Assign_Block_ID(c, max_block_id)
        else:
            Assign_Block_ID(c, cur_block_id)

def Run_Assign_Block_ID(topo):
    global max_block_id
    max_block_id = 0
    Assign_Block_ID(topo.gadag_root, max_block_id)

def Construct_Ear(x, stack, intf, ear_type):
    ear_list = []
    cur_intf = intf
    not_done = True
    while not_done:
        cur_intf.UNDIRECTED = False
        cur_intf.OUTGOING = True
        cur_intf.remote_intf.UNDIRECTED = False
        cur_intf.remote_intf.INCOMING = True
        if cur_intf.remote_node.IN_GADAG == False:
            cur_intf.remote_node.IN_GADAG = True
            ear_list.append(cur_intf.remote_node)
            if ear_type == 'CHILD':
                cur_intf = cur_intf.remote_node.lowpoint_parent_intf
            else:
                assert ear_type == 'NEIGHBOR'
                cur_intf = cur_intf.remote_node.dfs_parent_intf
        else:
            not_done = False

    if ear_type == 'CHILD' and cur_intf.remote_node is x:
        # x is a cut-vertex and the local root for the block
        # in which the ear is computed
        x.IS_CUT_VERTEX = True
        localroot = x
    else:
        # inherit local root from the end of the ear
        localroot = cur_intf.remote_node.localroot

    while ear_list != []:
        y = ear_list.pop()
        y.localroot = localroot
        stack.append(y)

def Construct_GADAG_via_Lowpoint(topo):
    gadag_root = topo.gadag_root
    gadag_root.IN_GADAG = True

```

```

gadag_root.localroot = None
stack = []
stack.append(gadag_root)
while stack != []:
    x = stack.pop()
    for intf in x.island_intf_list:
        if ( intf.remote_node.IN_GADAG == False
            and intf.remote_node.dfs_parent is x ):
            Construct_Ear(x, stack, intf, 'CHILD' )
    for intf in x.island_intf_list:
        if (intf.remote_node.IN_GADAG == False
            and intf.remote_node.dfs_parent is not x):
            Construct_Ear(x, stack, intf, 'NEIGHBOR')

def Assign_Remaining_Lowpoint_Parents(topo):
    for node in topo.island_node_list:
        if ( node is not topo.gadag_root
            and node.lowpoint_parent == None ):
            node.lowpoint_parent = node.dfs_parent
            node.lowpoint_parent_intf = node.dfs_parent_intf
            node.lowpoint_number = node.dfs_parent.dfs_number

def Add_Undirected_Block_Root_Links(topo):
    for node in topo.island_node_list:
        if node.IS_CUT_VERTEX or node is topo.gadag_root:
            for intf in node.island_intf_list:
                if ( intf.remote_node.localroot is not node
                    or intf.PROCESSED ):
                    continue
            bundle_list = []
            bundle = Bundle()
            for intf2 in node.island_intf_list:
                if intf2.remote_node is intf.remote_node:
                    bundle_list.append(intf2)
                    if not intf2.UNDIRECTED:
                        bundle.UNDIRECTED = False
                        if intf2.INCOMING:
                            bundle.INCOMING = True
                        if intf2.OUTGOING:
                            bundle.OUTGOING = True
            if bundle.UNDIRECTED:
                for intf3 in bundle_list:
                    intf3.UNDIRECTED = False
                    intf3.remote_intf.UNDIRECTED = False
                    intf3.PROCESSED = True
                    intf3.remote_intf.PROCESSED = True
                    intf3.OUTGOING = True
                    intf3.remote_intf.INCOMING = True

```



```
    else:
        if (bundle.OUTGOING and bundle.INCOMING):
            for intf3 in bundle_list:
                intf3.UNDIRECTED = False
                intf3.remote_intf.UNDIRECTED = False
                intf3.PROCESSED = True
                intf3.remote_intf.PROCESSED = True
                intf3.OUTGOING = True
                intf3.INCOMING = True
                intf3.remote_intf.INCOMING = True
                intf3.remote_intf.OUTGOING = True
            elif bundle.OUTGOING:
                for intf3 in bundle_list:
                    intf3.UNDIRECTED = False
                    intf3.remote_intf.UNDIRECTED = False
                    intf3.PROCESSED = True
                    intf3.remote_intf.PROCESSED = True
                    intf3.OUTGOING = True
                    intf3.remote_intf.INCOMING = True
            elif bundle.INCOMING:
                for intf3 in bundle_list:
                    intf3.UNDIRECTED = False
                    intf3.remote_intf.UNDIRECTED = False
                    intf3.PROCESSED = True
                    intf3.remote_intf.PROCESSED = True
                    intf3.INCOMING = True
                    intf3.remote_intf.OUTGOING = True

def Modify_Block_Root_Incoming_Links(topo):
    for node in topo.island_node_list:
        if ( node.IS_CUT_VERTEX == True or node is topo.gadag_root ):
            for intf in node.island_intf_list:
                if intf.remote_node.localroot is node:
                    if intf.INCOMING:
                        intf.INCOMING = False
                        intf.INCOMING_STORED = True
                        intf.remote_intf.OUTGOING = False
                        intf.remote_intf.OUTGOING_STORED = True

def Revert_Block_Root_Incoming_Links(topo):
    for node in topo.island_node_list:
        if ( node.IS_CUT_VERTEX == True or node is topo.gadag_root ):
            for intf in node.island_intf_list:
                if intf.remote_node.localroot is node:
                    if intf.INCOMING_STORED:
                        intf.INCOMING = True
                        intf.remote_intf.OUTGOING = True
                        intf.INCOMING_STORED = False
```

```

        intf.remote_intf.OUTGOING_STORED = False

def Run_Topological_Sort_GADAG(topo):
    Modify_Block_Root_Incoming_Links(topo)
    for node in topo.island_node_list:
        node.unvisited = 0
        for intf in node.island_intf_list:
            if (intf.INCOMING == True):
                node.unvisited += 1
    working_list = []
    topo_order_list = []
    working_list.append(topo.gadag_root)
    while working_list != []:
        y = working_list.pop(0)
        topo_order_list.append(y)
        for intf in y.island_intf_list:
            if ( intf.OUTGOING == True):
                intf.remote_node.unvisited -= 1
                if intf.remote_node.unvisited == 0:
                    working_list.append(intf.remote_node)
    next_topo_order = 1
    while topo_order_list != []:
        y = topo_order_list.pop(0)
        y.topo_order = next_topo_order
        next_topo_order += 1
    Revert_Block_Root_Incoming_Links(topo)

def Set_Other_Undirected_Links_Based_On_Topo_Order(topo):
    for node in topo.island_node_list:
        for intf in node.island_intf_list:
            if intf.UNDIRECTED:
                if node.topo_order < intf.remote_node.topo_order:
                    intf.OUTGOING = True
                    intf.UNDIRECTED = False
                    intf.remote_intf.INCOMING = True
                    intf.remote_intf.UNDIRECTED = False
                else:
                    intf.INCOMING = True
                    intf.UNDIRECTED = False
                    intf.remote_intf.OUTGOING = True
                    intf.remote_intf.UNDIRECTED = False

def Initialize_Temporary_Interface_Flags(topo):
    for node in topo.island_node_list:
        for intf in node.island_intf_list:
            intf.PROCESSED = False
            intf.INCOMING_STORED = False
            intf.OUTGOING_STORED = False

```

```

def Add_Undirected_Links(topo):
    Initialize_Temporary_Interface_Flags(topo)
    Add_Undirected_Block_Root_Links(topo)
    Run_Topological_Sort_GADAG(topo)
    Set_Other_Undirected_Links_Based_On_Topo_Order(topo)

def In_Common_Block(x,y):
    if ( (x.block_id == y.block_id)
        or ( x is y.localroot) or (y is x.localroot) ):
        return True
    return False

def Copy_List_Items(target_list, source_list):
    del target_list[:] # Python idiom to remove all elements of a list
    for element in source_list:
        target_list.append(element)

def Add_Item_To_List_If_New(target_list, item):
    if item not in target_list:
        target_list.append(item)

def Store_Results(y, direction):
    if direction == 'INCREASING':
        y.HIGHER = True
        Copy_List_Items(y.blue_next_hops, y.next_hops)
    if direction == 'DECREASING':
        y.LOWER = True
        Copy_List_Items(y.red_next_hops, y.next_hops)
    if direction == 'NORMAL_SPF':
        y.primary_spf_metric = y.spf_metric
        Copy_List_Items(y.primary_next_hops, y.next_hops)
    if direction == 'MRT_ISLAND_SPF':
        Copy_List_Items(y.mrt_island_next_hops, y.next_hops)
    if direction == 'COLLAPSED_SPF':
        y.collapsed_metric = y.spf_metric
        Copy_List_Items(y.collapsed_next_hops, y.next_hops)

# Note that the Python heapq fucntion allows for duplicate items,
# so we use the 'spf_visited' property to only consider a node
# as min_node the first time it gets removed from the heap.
def SPF_No_Traverse_Block_Root(topo, spf_root, block_root, direction):
    spf_heap = []
    for y in topo.island_node_list:
        y.spf_metric = 2147483647 # 2^31-1
        y.next_hops = []
        y.spf_visited = False
    spf_root.spf_metric = 0
    heapq.heappush(spf_heap,

```

```

        (spf_root.spf_metric, spf_root.node_id, spf_root) )
while spf_heap != []:
    #extract third element of tuple popped from heap
    min_node = heapq.heappop(spf_heap)[2]
    if min_node.spf_visited:
        continue
    min_node.spf_visited = True
    Store_Results(min_node, direction)
    if ( (min_node is spf_root) or (min_node is not block_root) ):
        for intf in min_node.island_intf_list:
            if ( (direction == 'INCREASING' and intf.OUTGOING )
                or (direction == 'DECREASING' and intf.INCOMING ) )
                and In_Common_Block(spf_root, intf.remote_node) ):
                path_metric = min_node.spf_metric + intf.metric
                if path_metric < intf.remote_node.spf_metric:
                    intf.remote_node.spf_metric = path_metric
                    if min_node is spf_root:
                        intf.remote_node.next_hops = [intf]
                    else:
                        Copy_List_Items(intf.remote_node.next_hops,
                                       min_node.next_hops)
                heapq.heappush(spf_heap,
                              ( intf.remote_node.spf_metric,
                                intf.remote_node.node_id,
                                intf.remote_node ) )
            elif path_metric == intf.remote_node.spf_metric:
                if min_node is spf_root:
                    Add_Item_To_List_If_New(
                        intf.remote_node.next_hops,intf)
                else:
                    for nh_intf in min_node.next_hops:
                        Add_Item_To_List_If_New(
                            intf.remote_node.next_hops,nh_intf)

def Normal_SPF(topo, spf_root):
    spf_heap = []
    for y in topo.node_list:
        y.spf_metric = 2147483647 # 2^31-1 as max metric
        y.next_hops = []
        y.primary_spf_metric = 2147483647
        y.primary_next_hops = []
        y.spf_visited = False
    spf_root.spf_metric = 0
    heapq.heappush(spf_heap,
                   (spf_root.spf_metric,spf_root.node_id,spf_root) )
    while spf_heap != []:
        #extract third element of tuple popped from heap
        min_node = heapq.heappop(spf_heap)[2]

```

```

    if min_node.spf_visited:
        continue
    min_node.spf_visited = True
    Store_Results(min_node, 'NORMAL_SPF')
    for intf in min_node.intf_list:
        path_metric = min_node.spf_metric + intf.metric
        if path_metric < intf.remote_node.spf_metric:
            intf.remote_node.spf_metric = path_metric
            if min_node is spf_root:
                intf.remote_node.next_hops = [intf]
            else:
                Copy_List_Items(intf.remote_node.next_hops,
                               min_node.next_hops)
            heapq.heappush(spf_heap,
                          ( intf.remote_node.spf_metric,
                            intf.remote_node.node_id,
                            intf.remote_node ) )
        elif path_metric == intf.remote_node.spf_metric:
            if min_node is spf_root:
                Add_Item_To_List_If_New(
                    intf.remote_node.next_hops,intf)
            else:
                for nh_intf in min_node.next_hops:
                    Add_Item_To_List_If_New(
                        intf.remote_node.next_hops,nh_intf)

def Set_Edge(y):
    if (y.blue_next_hops == [] and y.red_next_hops == []):
        Set_Edge(y.localroot)
        Copy_List_Items(y.blue_next_hops,y.localroot.blue_next_hops)
        Copy_List_Items(y.red_next_hops ,y.localroot.red_next_hops)
        y.order_proxy = y.localroot.order_proxy

def Compute_MRT_NH_For_One_Src_To_Island_Dests(topo,x):
    for y in topo.island_node_list:
        y.HIGHER = False
        y.LOWER = False
        y.red_next_hops = []
        y.blue_next_hops = []
        y.order_proxy = y
    SPF_No_Traverse_Block_Root(topo, x, x.localroot, 'INCREASING')
    SPF_No_Traverse_Block_Root(topo, x, x.localroot, 'DECREASING')
    for y in topo.island_node_list:
        if ( y is not x and (y.block_id == x.block_id) ):
            assert (not ( y is x.localroot or x is y.localroot) )
            assert(not (y.HIGHER and y.LOWER) )
            if y.HIGHER == True:
                Copy_List_Items(y.red_next_hops,

```

```

        x.localroot.red_next_hops)
    elif y.LOWER == True:
        Copy_List_Items(y.blue_next_hops,
                        x.localroot.blue_next_hops)
    else:
        Copy_List_Items(y.blue_next_hops,
                        x.localroot.red_next_hops)
        Copy_List_Items(y.red_next_hops,
                        x.localroot.blue_next_hops)

# Inherit x's MRT next-hops to reach the GADAG root
# from x's MRT next-hops to reach its local root,
# but first check if x is the gadag_root (in which case
# x does not have a local root) or if x's local root
# is the gadag root (in which case we already have the
# x's MRT next-hops to reach the gadag root)
if x is not topo.gadag_root and x.localroot is not topo.gadag_root:
    Copy_List_Items(topo.gadag_root.blue_next_hops,
                    x.localroot.blue_next_hops)
    Copy_List_Items(topo.gadag_root.red_next_hops,
                    x.localroot.red_next_hops)
    topo.gadag_root.order_proxy = x.localroot

# Inherit next-hops and order_proxies to other blocks
for y in topo.island_node_list:
    if (y is not topo.gadag_root and y is not x ):
        Set_Edge(y)

def Store_MRT_Nexthops_For_One_Src_To_Island_Dests(topo,x):
    for y in topo.island_node_list:
        if y is x:
            continue
        x.blue_next_hops_dict[y.node_id] = []
        x.red_next_hops_dict[y.node_id] = []
        Copy_List_Items(x.blue_next_hops_dict[y.node_id],
                        y.blue_next_hops)
        Copy_List_Items(x.red_next_hops_dict[y.node_id],
                        y.red_next_hops)

def Store_Primary_and_Alts_For_One_Src_To_Island_Dests(topo,x):
    for y in topo.island_node_list:
        x.pnh_dict[y.node_id] = []
        Copy_List_Items(x.pnh_dict[y.node_id], y.primary_next_hops)
        x.alt_dict[y.node_id] = []
        Copy_List_Items(x.alt_dict[y.node_id], y.alt_list)

def Store_Primary_NHs_For_One_Source_To_Nodes(topo,x):
    for y in topo.node_list:

```

```

        x.pnh_dict[y.node_id] = []
        Copy_List_Items(x.pnh_dict[y.node_id], y.primary_next_hops)

def Store_MRT_NHs_For_One_Src_To_Named_Proxy_Nodes(topo,x):
    for prefix in topo.named_proxy_dict:
        P = topo.named_proxy_dict[prefix]
        x.blue_next_hops_dict[P.node_id] = []
        x.red_next_hops_dict[P.node_id] = []
        Copy_List_Items(x.blue_next_hops_dict[P.node_id],
                        P.blue_next_hops)
        Copy_List_Items(x.red_next_hops_dict[P.node_id],
                        P.red_next_hops)

def Store_Alts_For_One_Src_To_Named_Proxy_Nodes(topo,x):
    for prefix in topo.named_proxy_dict:
        P = topo.named_proxy_dict[prefix]
        x.alt_dict[P.node_id] = []
        Copy_List_Items(x.alt_dict[P.node_id],
                        P.alt_list)

def Store_Primary_NHs_For_One_Src_To_Named_Proxy_Nodes(topo,x):
    for prefix in topo.named_proxy_dict:
        P = topo.named_proxy_dict[prefix]
        x.pnh_dict[P.node_id] = []
        Copy_List_Items(x.pnh_dict[P.node_id],
                        P.primary_next_hops)

def Select_Alternates_Internal(D, F, primary_intf,
                               D_lower, D_higher, D_topo_order):
    if D_higher and D_lower:
        if F.HIGHER and F.LOWER:
            if F.topo_order > D_topo_order:
                return 'USE_BLUE'
            else:
                return 'USE_RED'
        if F.HIGHER:
            return 'USE_RED'
        if F.LOWER:
            return 'USE_BLUE'
        assert(primary_intf.MRT_INELIGIBLE
               or primary_intf.remote_intf.MRT_INELIGIBLE)
        return 'USE_RED_OR_BLUE'
    if D_higher:
        if F.HIGHER and F.LOWER:
            return 'USE_BLUE'
        if F.LOWER:
            return 'USE_BLUE'
        if F.HIGHER:

```

```
        if (F.topo_order > D.topo_order):
            return 'USE_BLUE'
        if (F.topo_order < D.topo_order):
            return 'USE_RED'
        assert(False)
    assert(primary_intf.MRT_INELIGIBLE
           or primary_intf.remote_intf.MRT_INELIGIBLE)
    return 'USE_RED_OR_BLUE'
if D_lower:
    if F.HIGHER and F.LOWER:
        return 'USE_RED'
    if F.HIGHER:
        return 'USE_RED'
    if F.LOWER:
        if F.topo_order > D.topo_order:
            return 'USE_BLUE'
        if F.topo_order < D.topo_order:
            return 'USE_RED'
        assert(False)
    assert(primary_intf.MRT_INELIGIBLE
           or primary_intf.remote_intf.MRT_INELIGIBLE)
    return 'USE_RED_OR_BLUE'
else: # D is unordered wrt S
    if F.HIGHER and F.LOWER:
        if primary_intf.OUTGOING and primary_intf.INCOMING:
            # This can happen when F and D are in different blocks
            return 'USE_RED_OR_BLUE'
        if primary_intf.OUTGOING:
            return 'USE_BLUE'
        if primary_intf.INCOMING:
            return 'USE_RED'
        #This can occur when primary_intf is MRT_INELIGIBLE.
        #This appears to be a case where the special
        #construction of the GADAG allows us to choose red,
        #whereas with an arbitrary GADAG, neither red nor blue
        #is guaranteed to work.
        assert(primary_intf.MRT_INELIGIBLE
               or primary_intf.remote_intf.MRT_INELIGIBLE)
        return 'USE_RED'
    if F.LOWER:
        return 'USE_RED'
    if F.HIGHER:
        return 'USE_BLUE'
    assert(primary_intf.MRT_INELIGIBLE
           or primary_intf.remote_intf.MRT_INELIGIBLE)
    if F.topo_order > D.topo_order:
        return 'USE_BLUE'
    else:
```



```

        return 'USE_RED'

def Select_Alternates(D, F, primary_intf):
    S = primary_intf.local_node
    if not In_Common_Block(F, S):
        return 'PRIM_NH_IN_DIFFERENT_BLOCK'
    if (D is F) or (D.order_proxy is F):
        return 'PRIM_NH_IS_D_OR_OP_FOR_D'
    D_lower = D.order_proxy.LOWER
    D_higher = D.order_proxy.HIGHER
    D_topo_order = D.order_proxy.topo_order
    return Select_Alternates_Internal(D, F, primary_intf,
                                     D_lower, D_higher, D_topo_order)

def Is_Remote_Node_In_NH_List(node, intf_list):
    for intf in intf_list:
        if node is intf.remote_node:
            return True
    return False

def Select_Alts_For_One_Src_To_Island_Dests(topo,x):
    Normal_SPF(topo, x)
    for D in topo.island_node_list:
        D.alt_list = []
        if D is x:
            continue
        for failed_intf in D.primary_next_hops:
            alt = Alternate()
            alt.failed_intf = failed_intf
            cand_alt_list = []
            F = failed_intf.remote_node
            #We need to test if F is in the island, as opposed
            #to just testing if failed_intf is in island_intf_list,
            #because failed_intf could be marked as MRT_INELIGIBLE.
            if F in topo.island_node_list:
                alt.info = Select_Alternates(D, F, failed_intf)
            else:
                #The primary next-hop is not in the MRT Island.
                #Either red or blue will avoid the primary next-hop,
                #because the primary next-hop is not even in the
                #GADAG.
                alt.info = 'USE_RED_OR_BLUE'

            if (alt.info == 'USE_RED_OR_BLUE'):
                alt.red_or_blue = random.choice(['USE_RED', 'USE_BLUE'])
            if (alt.info == 'USE_BLUE'
                or alt.red_or_blue == 'USE_BLUE'):

```

```
Copy_List_Items(alt.nh_list, D.blue_next_hops)
alt.fec = 'BLUE'
alt.prot = 'NODE_PROTECTION'
if (alt.info == 'USE_RED' or alt.red_or_blue == 'USE_RED'):
    Copy_List_Items(alt.nh_list, D.red_next_hops)
    alt.fec = 'RED'
    alt.prot = 'NODE_PROTECTION'
if (alt.info == 'PRIM_NH_IN_DIFFERENT_BLOCK'):
    alt.fec = 'NO_ALTERNATE'
    alt.prot = 'NO_PROTECTION'
if (alt.info == 'PRIM_NH_IS_D_OR_OP_FOR_D'):
    if failed_intf.OUTGOING and failed_intf.INCOMING:
        # cut-link: if there are parallel cut links, use
        # the link(s) with lowest metric that are not
        # primary intf or None
        cand_alt_list = [None]
        min_metric = 2147483647
        for intf in x.island_intf_list:
            if ( intf is not failed_intf and
                (intf.remote_node is
                 failed_intf.remote_node)):
                if intf.metric < min_metric:
                    cand_alt_list = [intf]
                    min_metric = intf.metric
                elif intf.metric == min_metric:
                    cand_alt_list.append(intf)
        if cand_alt_list != [None]:
            alt.fec = 'GREEN'
            alt.prot = 'PARALLEL_CUTLINK'
        else:
            alt.fec = 'NO_ALTERNATE'
            alt.prot = 'NO_PROTECTION'
        Copy_List_Items(alt.nh_list, cand_alt_list)

# Is_Remote_Node_In_NH_List() is used, as opposed
# to just checking if failed_intf is in D.red_next_hops,
# because failed_intf could be marked as MRT_INELIGIBLE.
elif Is_Remote_Node_In_NH_List(F, D.red_next_hops):
    Copy_List_Items(alt.nh_list, D.blue_next_hops)
    alt.fec = 'BLUE'
    alt.prot = 'LINK_PROTECTION'
elif Is_Remote_Node_In_NH_List(F, D.blue_next_hops):
    Copy_List_Items(alt.nh_list, D.red_next_hops)
    alt.fec = 'RED'
    alt.prot = 'LINK_PROTECTION'
else:
    alt.fec = random.choice(['RED', 'BLUE'])
    alt.prot = 'LINK_PROTECTION'
```

```

        D.alt_list.append(alt)

def Write_GADAG_To_File(topo, file_prefix):
    gadag_edge_list = []
    for node in topo.node_list:
        for intf in node.intf_list:
            if intf.SIMULATION_OUTGOING:
                local_node = "%04d" % (intf.local_node.node_id)
                remote_node = "%04d" % (intf.remote_node.node_id)
                intf_data = "%03d" % (intf.link_data)
                edge_string=(local_node+', '+remote_node+', '+
                            intf_data+'\n')
                gadag_edge_list.append(edge_string)
    gadag_edge_list.sort();
    filename = file_prefix + '_gadag.csv'
    with open(filename, 'w') as gadag_file:
        gadag_file.write('local_node,'\
                        'remote_node,local_intf_link_data\n')
        for edge_string in gadag_edge_list:
            gadag_file.write(edge_string);

def Write_MRTs_For_All_Dests_To_File(topo, color, file_prefix):
    edge_list = []
    for node in topo.island_node_list_for_test_gr:
        if color == 'blue':
            node_next_hops_dict = node.blue_next_hops_dict
        elif color == 'red':
            node_next_hops_dict = node.red_next_hops_dict
        for dest_node_id in node_next_hops_dict:
            for intf in node_next_hops_dict[dest_node_id]:
                gadag_root = "%04d" % (topo.gadag_root.node_id)
                dest_node = "%04d" % (dest_node_id)
                local_node = "%04d" % (intf.local_node.node_id)
                remote_node = "%04d" % (intf.remote_node.node_id)
                intf_data = "%03d" % (intf.link_data)
                edge_string=(gadag_root+', '+dest_node+', '+local_node+
                            ', '+remote_node+', '+intf_data+'\n')
                edge_list.append(edge_string)
    edge_list.sort()
    filename = file_prefix + '_' + color + '_to_all.csv'
    with open(filename, 'w') as mrt_file:
        mrt_file.write('gadag_root,dest,'\
                        'local_node,remote_node,link_data\n')
        for edge_string in edge_list:
            mrt_file.write(edge_string);

def Write_Both_MRTs_For_All_Dests_To_File(topo, file_prefix):
    Write_MRTs_For_All_Dests_To_File(topo, 'blue', file_prefix)

```

```

Write_MRTs_For_All_Dests_To_File(topo, 'red', file_prefix)

def Write_Alternates_For_All_Dests_To_File(topo, file_prefix):
    edge_list = []
    for x in topo.island_node_list_for_test_gr:
        for dest_node_id in x.alt_dict:
            alt_list = x.alt_dict[dest_node_id]
            for alt in alt_list:
                for alt_intf in alt.nh_list:
                    gadag_root = "%04d" % (topo.gadag_root.node_id)
                    dest_node = "%04d" % (dest_node_id)
                    prim_local_node = \
                        "%04d" % (alt.failed_intf.local_node.node_id)
                    prim_remote_node = \
                        "%04d" % (alt.failed_intf.remote_node.node_id)
                    prim_intf_data = \
                        "%03d" % (alt.failed_intf.link_data)
                    if alt_intf == None:
                        alt_local_node = "None"
                        alt_remote_node = "None"
                        alt_intf_data = "None"
                    else:
                        alt_local_node = \
                            "%04d" % (alt_intf.local_node.node_id)
                        alt_remote_node = \
                            "%04d" % (alt_intf.remote_node.node_id)
                        alt_intf_data = \
                            "%03d" % (alt_intf.link_data)
                    edge_string = (gadag_root+', '+dest_node+', '+
                        prim_local_node+', '+prim_remote_node+', '+
                        prim_intf_data+', '+alt_local_node+', '+
                        alt_remote_node+', '+alt_intf_data+', '+
                        alt.fec + '\n')
                    edge_list.append(edge_string)
    edge_list.sort()
    filename = file_prefix + '_alts_to_all.csv'
    with open(filename, 'w') as alt_file:
        alt_file.write('gadag_root,dest,'\
            'prim_nh.local_node,prim_nh.remote_node,'\
            'prim_nh.link_data,alt_nh.local_node,'\
            'alt_nh.remote_node,alt_nh.link_data,'\
            'alt_nh.fec\n')
        for edge_string in edge_list:
            alt_file.write(edge_string);

def Raise_GADAG_Root_Selection_Priority(topo,node_id):
    node = topo.node_dict[node_id]
    node.GR_sel_priority = 255

```

```
def Lower_GADAG_Root_Selection_Priority(topo,node_id):
    node = topo.node_dict[node_id]
    node.GR_sel_priority = 128

def GADAG_Root_Compare(node_a, node_b):
    if (node_a.GR_sel_priority > node_b.GR_sel_priority):
        return 1
    elif (node_a.GR_sel_priority < node_b.GR_sel_priority):
        return -1
    else:
        if node_a.node_id > node_b.node_id:
            return 1
        elif node_a.node_id < node_b.node_id:
            return -1

def Set_GADAG_Root(topo,computing_router):
    gadag_root_list = []
    for node in topo.island_node_list:
        gadag_root_list.append(node)
    gadag_root_list.sort(GADAG_Root_Compare)
    topo.gadag_root = gadag_root_list.pop()

def Add_Prefix_Advertisements_From_File(topo, filename):
    prefix_filename = filename + '.prefix'
    cols_list = []
    if not os.path.exists(prefix_filename):
        return
    with open(prefix_filename) as prefix_file:
        for line in prefix_file:
            line = line.rstrip('\r\n')
            cols=line.split(',')
            cols_list.append(cols)
            prefix_id = int(cols[0])
            if prefix_id < 2000 or prefix_id >2999:
                print('skipping the following line of prefix file')
                print('prefix id should be between 2000 and 2999')
                print(line)
                continue
            prefix_node_id = int(cols[1])
            prefix_cost = int(cols[2])
            advertising_node = topo.node_dict[prefix_node_id]
            advertising_node.prefix_cost_dict[prefix_id] = prefix_cost

def Add_Prefixes_for_Non_Island_Nodes(topo):
    for node in topo.node_list:
        if node.IN_MRT_ISLAND:
            continue
        prefix_id = node.node_id + 1000
```

```

        node.prefix_cost_dict[prefix_id] = 0

def Add_Profile_IDs_from_File(topo, filename):
    profile_filename = filename + '.profile'
    for node in topo.node_list:
        node.profile_id_list = []
    cols_list = []
    if os.path.exists(profile_filename):
        with open(profile_filename) as profile_file:
            for line in profile_file:
                line = line.rstrip('\r\n')
                cols=line.split(',')
                cols_list.append(cols)
                node_id = int(cols[0])
                profile_id = int(cols[1])
                this_node = topo.node_dict[node_id]
                this_node.profile_id_list.append(profile_id)
    else:
        for node in topo.node_list:
            node.profile_id_list = [0]

def Island_Marking_SPF(topo,spf_root):
    spf_root.isl_marking_spf_dict = {}
    for y in topo.node_list:
        y.spf_metric = 2147483647 # 2^31-1 as max metric
        y.PATH_HITS_ISLAND = False
        y.next_hops = []
        y.spf_visited = False
    spf_root.spf_metric = 0
    spf_heap = []
    heapq.heappush(spf_heap,
                    (spf_root.spf_metric,spf_root.node_id,spf_root) )
    while spf_heap != []:
        #extract third element of tuple popped from heap
        min_node = heapq.heappop(spf_heap)[2]
        if min_node.spf_visited:
            continue
        min_node.spf_visited = True
        spf_root.isl_marking_spf_dict[min_node.node_id] = \
            (min_node.spf_metric, min_node.PATH_HITS_ISLAND)
        for intf in min_node.intf_list:
            path_metric = min_node.spf_metric + intf.metric
            if path_metric < intf.remote_node.spf_metric:
                intf.remote_node.spf_metric = path_metric
                if min_node is spf_root:
                    intf.remote_node.next_hops = [intf]
            else:
                Copy_List_Items(intf.remote_node.next_hops,

```

```

        min_node.next_hops)
    if (intf.remote_node.IN_MRT_ISLAND):
        intf.remote_node.PATH_HITS_ISLAND = True
    else:
        intf.remote_node.PATH_HITS_ISLAND = \
            min_node.PATH_HITS_ISLAND
    heapq.heappush(spf_heap,
        ( intf.remote_node.spf_metric,
          intf.remote_node.node_id,
          intf.remote_node ) )
elif path_metric == intf.remote_node.spf_metric:
    if min_node is spf_root:
        Add_Item_To_List_If_New(
            intf.remote_node.next_hops,intf)
    else:
        for nh_intf in min_node.next_hops:
            Add_Item_To_List_If_New(
                intf.remote_node.next_hops,nh_intf)
    if (intf.remote_node.IN_MRT_ISLAND):
        intf.remote_node.PATH_HITS_ISLAND = True
    else:
        if (intf.remote_node.PATH_HITS_ISLAND
            or min_node.PATH_HITS_ISLAND):
            intf.remote_node.PATH_HITS_ISLAND = True

def Create_Basic_Named_Proxy_Nodes(topo):
    for node in topo.node_list:
        for prefix in node.prefix_cost_dict:
            prefix_cost = node.prefix_cost_dict[prefix]
            if prefix in topo.named_proxy_dict:
                P = topo.named_proxy_dict[prefix]
                P.node_prefix_cost_list.append((node,prefix_cost))
            else:
                P = Named_Proxy_Node()
                topo.named_proxy_dict[prefix] = P
                P.node_id = prefix
                P.node_prefix_cost_list = [(node,prefix_cost)]

def Compute_Loop_Free_Island_Neighbors_For_Each_Prefix(topo):
    topo.island_nbr_set = set()
    topo.island_border_set = set()
    for node in topo.node_list:
        if node.IN_MRT_ISLAND:
            continue
        for intf in node.intf_list:
            if intf.remote_node.IN_MRT_ISLAND:
                topo.island_nbr_set.add(node)

```

```

        topo.island_border_set.add(intf.remote_node)

    for island_nbr in topo.island_nbr_set:
        Island_Marking_SPF(topo, island_nbr)

    for prefix in topo.named_proxy_dict:
        P = topo.named_proxy_dict[prefix]
        P.lfin_list = []
        for island_nbr in topo.island_nbr_set:
            min_isl_nbr_to_pref_cost = 2147483647
            for (adv_node, prefix_cost) in P.node_prefix_cost_list:
                (adv_node_cost, path_hits_island) = \
                    island_nbr.isl_marking_spf_dict[adv_node.node_id]
                isl_nbr_to_pref_cost = adv_node_cost + prefix_cost
                if isl_nbr_to_pref_cost < min_isl_nbr_to_pref_cost:
                    min_isl_nbr_to_pref_cost = isl_nbr_to_pref_cost
                    min_path_hits_island = path_hits_island
                elif isl_nbr_to_pref_cost == min_isl_nbr_to_pref_cost:
                    if min_path_hits_island or path_hits_island:
                        min_path_hits_island = True
            if not min_path_hits_island:
                P.lfin_list.append( (island_nbr,
                                    min_isl_nbr_to_pref_cost) )

def Compute_Island_Border_Router_LFIN_Pairs_For_Each_Prefix(topo):
    for ibr in topo.island_border_set:
        ibr.prefix_lfin_dict = {}
        ibr.min_intf_metric_dict = {}
        ibr.min_intf_list_dict = {}
        ibr.min_intf_list_dict[None] = None
        for intf in ibr.intf_list:
            if not intf.remote_node in topo.island_nbr_set:
                continue
            if not intf.remote_node in ibr.min_intf_metric_dict:
                ibr.min_intf_metric_dict[intf.remote_node] = \
                    intf.metric
                ibr.min_intf_list_dict[intf.remote_node] = [intf]
            else:
                if (intf.metric
                    < ibr.min_intf_metric_dict[intf.remote_node]):
                    ibr.min_intf_metric_dict[intf.remote_node] = \
                        intf.metric
                    ibr.min_intf_list_dict[intf.remote_node] = [intf]
                elif (intf.metric
                      < ibr.min_intf_metric_dict[intf.remote_node]):
                    ibr.min_intf_list_dict[intf.remote_node].\
                        append(intf)

```



```

for prefix in topo.named_proxy_dict:
    P = topo.named_proxy_dict[prefix]
    for ibr in topo.island_border_set:
        min_ibr_lfin_pref_cost = 2147483647
        min_lfin = None
        for (lfin, lfin_to_pref_cost) in P.lfin_list:
            if not lfin in ibr.min_intf_metric_dict:
                continue
            ibr_lfin_pref_cost = \
                ibr.min_intf_metric_dict[lfin] + lfin_to_pref_cost
            if ibr_lfin_pref_cost < min_ibr_lfin_pref_cost:
                min_ibr_lfin_pref_cost = ibr_lfin_pref_cost
                min_lfin = lfin
        ibr.prefix_lfin_dict[prefix] = (min_lfin,
            min_ibr_lfin_pref_cost,
            ibr.min_intf_list_dict[min_lfin])

def Proxy_Node_Att_Router_Compare(pnar_a, pnar_b):
    if pnar_a.named_proxy_cost < pnar_b.named_proxy_cost:
        return -1
    if pnar_b.named_proxy_cost < pnar_a.named_proxy_cost:
        return 1
    if pnar_a.node.node_id < pnar_b.node.node_id:
        return -1
    if pnar_b.node.node_id < pnar_a.node.node_id:
        return 1
    if pnar_a.min_lfin == None:
        return -1
    if pnar_b.min_lfin == None:
        return 1

def Choose_Proxy_Node_Attachment_Routers(topo):
    for prefix in topo.named_proxy_dict:
        P = topo.named_proxy_dict[prefix]
        pnar_candidate_list = []
        for (node, prefix_cost) in P.node_prefix_cost_list:
            if not node.IN_MRT_ISLAND:
                continue
            pnar = Proxy_Node_Attachment_Router()
            pnar.prefix = prefix
            pnar.named_proxy_cost = prefix_cost
            pnar.node = node
            pnar_candidate_list.append(pnar)
        for ibr in topo.island_border_set:
            (min_lfin, prefix_cost, min_intf_list) = \
                ibr.prefix_lfin_dict[prefix]
            if min_lfin == None:
                continue

```

```

        pnar = Proxy_Node_Attachment_Router()
        pnar.named_proxy_cost = prefix_cost
        pnar.node = ibr
        pnar.min_lfin = min_lfin
        pnar.nh_intf_list = min_intf_list
        pnar_candidate_list.append(pnar)
    pnar_candidate_list.sort(cmp=Proxy_Node_Att_Router_Compare)
    #pop first element from list
    first_pnar = pnar_candidate_list.pop(0)
    second_pnar = None
    for next_pnar in pnar_candidate_list:
        if next_pnar.node is first_pnar.node:
            continue
        second_pnar = next_pnar
        break

    P.pnar1 = first_pnar
    P.pnar2 = second_pnar

def Attach_Named_Proxy_Nodes(topo):
    Compute_Loop_Free_Island_Neighbors_For_Each_Prefix(topo)
    Compute_Island_Border_Router_LFIN_Pairs_For_Each_Prefix(topo)
    Choose_Proxy_Node_Attachment_Routers(topo)

def Select_Proxy_Node_NHs(P,S):
    if P.pnar1.node.node_id < P.pnar2.node.node_id:
        X = P.pnar1.node
        Y = P.pnar2.node
    else:
        X = P.pnar2.node
        Y = P.pnar1.node
    P.pnar_X = X
    P.pnar_Y = Y
    A = X.order_proxy
    B = Y.order_proxy
    if (A is S.localroot
        and B is S.localroot):
        #print("1.0")
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
    if (A is S.localroot
        and B is not S.localroot):
        #print("2.0")
        if B.LOWER:
            #print("2.1")
            Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
            Copy_List_Items(P.red_next_hops, Y.red_next_hops)

```

```

        return
    if B.HIGHER:
        #print("2.2")
        Copy_List_Items(P.blue_next_hops, X.red_next_hops)
        Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
        return
    else:
        #print("2.3")
        Copy_List_Items(P.blue_next_hops, X.red_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
if (A is not S.localroot
    and B is S.localroot):
    #print("3.0")
    if A.LOWER:
        #print("3.1")
        Copy_List_Items(P.blue_next_hops, X.red_next_hops)
        Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
        return
    if A.HIGHER:
        #print("3.2")
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
    else:
        #print("3.3")
        Copy_List_Items(P.blue_next_hops, X.red_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
if (A is not S.localroot
    and B is not S.localroot):
    #print("4.0")
    if (S is A.localroot or S is B.localroot):
        #print("4.05")
        if A.topo_order < B.topo_order:
            #print("4.05.1")
            Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
            Copy_List_Items(P.red_next_hops, Y.red_next_hops)
            return
        else:
            #print("4.05.2")
            Copy_List_Items(P.blue_next_hops, X.red_next_hops)
            Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
            return
    if A.LOWER:
        #print("4.1")
        if B.HIGHER:
            #print("4.1.1")

```

```
Copy_List_Items(P.blue_next_hops, X.red_next_hops)
Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
return
if B.LOWER:
    #print("4.1.2")
    if A.topo_order < B.topo_order:
        #print("4.1.2.1")
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
    else:
        #print("4.1.2.2")
        Copy_List_Items(P.blue_next_hops, X.red_next_hops)
        Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
        return
else:
    #print("4.1.3")
    Copy_List_Items(P.blue_next_hops, X.red_next_hops)
    Copy_List_Items(P.red_next_hops, Y.red_next_hops)
    return
if A.HIGHER:
    #print("4.2")
    if B.HIGHER:
        #print("4.2.1")
        if A.topo_order < B.topo_order:
            #print("4.2.1.1")
            Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
            Copy_List_Items(P.red_next_hops, Y.red_next_hops)
            return
        else:
            #print("4.2.1.2")
            Copy_List_Items(P.blue_next_hops, X.red_next_hops)
            Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
            return
    if B.LOWER:
        #print("4.2.2")
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
    else:
        #print("4.2.3")
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
        return
else:
    #print("4.3")
    if B.LOWER:
        #print("4.3.1")
```

```

        Copy_List_Items(P.blue_next_hops, X.red_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
    if B.HIGHER:
        #print("4.3.2")
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
        return
    else:
        #print("4.3.3")
        if A.topo_order < B.topo_order:
            #print("4.3.3.1")
            Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
            Copy_List_Items(P.red_next_hops, Y.red_next_hops)
            return
        else:
            #print("4.3.3.2")
            Copy_List_Items(P.blue_next_hops, X.red_next_hops)
            Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
            return
    assert(False)

def Compute_MRT_NHs_For_One_Src_To_Named_Proxy_Nodes(topo,S):
    for prefix in topo.named_proxy_dict:
        P = topo.named_proxy_dict[prefix]
        if P.pnar2 == None:
            if S is P.pnar1.node:
                # set the MRT next-hops for the PNAR to
                # reach the LFIN and change FEC to green
                Copy_List_Items(P.blue_next_hops,
                               P.pnar1.nh_intf_list)
                S.blue_to_green_nh_dict[P.node_id] = True
                Copy_List_Items(P.red_next_hops,
                               P.pnar1.nh_intf_list)
                S.red_to_green_nh_dict[P.node_id] = True
            else:
                # inherit MRT NHs for P from pnar1
                Copy_List_Items(P.blue_next_hops,
                               P.pnar1.node.blue_next_hops)
                Copy_List_Items(P.red_next_hops,
                               P.pnar1.node.red_next_hops)
        else:
            Select_Proxy_Node_NHs(P,S)
            # set the MRT next-hops for the PNAR to reach the LFIN
            # and change FEC to green rely on the red or blue
            # next-hops being empty to figure out which one needs
            # to point to the LFIN.
            if S is P.pnar1.node:

```

```

        this_pnar = P.pnar1
    elif S is P.pnar2.node:
        this_pnar = P.pnar2
    else:
        continue
    if P.blue_next_hops == []:
        Copy_List_Items(P.blue_next_hops,
            this_pnar.nh_intf_list)
        S.blue_to_green_nh_dict[P.node_id] = True
    if P.red_next_hops == []:
        Copy_List_Items(P.red_next_hops,
            this_pnar.nh_intf_list)
        S.red_to_green_nh_dict[P.node_id] = True

def Select_Alternates_Proxy_Node(P,F,primary_intf):
    S = primary_intf.local_node
    X = P.pnar_X
    Y = P.pnar_Y
    A = X.order_proxy
    B = Y.order_proxy
    if F is A and F is B:
        return 'PRIM_NH_IS_OP_FOR_BOTH_X_AND_Y'
    if F is A:
        return 'USE_RED'
    if F is B:
        return 'USE_BLUE'

    if not In_Common_Block(A, B):
        if In_Common_Block(F, A):
            return 'USE_RED'
        elif In_Common_Block(F, B):
            return 'USE_BLUE'
        else:
            return 'USE_RED_OR_BLUE'
    if (not In_Common_Block(F, A)
        and not In_Common_Block(F, B)):
        return 'USE_RED_OR_BLUE'

    alt_to_X = Select_Alternates(X, F, primary_intf)
    alt_to_Y = Select_Alternates(Y, F, primary_intf)

    if (alt_to_X == 'USE_RED_OR_BLUE'
        and alt_to_Y == 'USE_RED_OR_BLUE'):
        return 'USE_RED_OR_BLUE'
    if alt_to_X == 'USE_RED_OR_BLUE':
        return 'USE_BLUE'
    if alt_to_Y == 'USE_RED_OR_BLUE':
        return 'USE_RED'

```

```
if (A is S.localroot
    and B is S.localroot):
    #print("1.0")
    if (alt_to_X == 'USE_BLUE' and alt_to_Y == 'USE_RED'):
        return 'USE_RED_OR_BLUE'
    if alt_to_X == 'USE_BLUE':
        return 'USE_BLUE'
    if alt_to_Y == 'USE_RED':
        return 'USE_RED'
    assert(False)
if (A is S.localroot
    and B is not S.localroot):
    #print("2.0")
    if B.LOWER:
        #print("2.1")
        if (alt_to_X == 'USE_BLUE' and alt_to_Y == 'USE_RED'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_BLUE':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_RED':
            return 'USE_RED'
        assert(False)
    if B.HIGHER:
        #print("2.2")
        if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_BLUE'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_RED':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_BLUE':
            return 'USE_RED'
        assert(False)
    else:
        #print("2.3")
        if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_RED'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_RED':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_RED':
            return 'USE_RED'
        assert(False)
if (A is not S.localroot
    and B is S.localroot):
    #print("3.0")
    if A.LOWER:
        #print("3.1")
        if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_BLUE'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_RED':
```

```

        return 'USE_BLUE'
    if alt_to_Y == 'USE_BLUE':
        return 'USE_RED'
    assert(False)
if A.HIGHER:
    #print("3.2")
    if (alt_to_X == 'USE_BLUE' and alt_to_Y == 'USE_RED'):
        return 'USE_RED_OR_BLUE'
    if alt_to_X == 'USE_BLUE':
        return 'USE_BLUE'
    if alt_to_Y == 'USE_RED':
        return 'USE_RED'
    assert(False)
else:
    #print("3.3")
    if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_RED'):
        return 'USE_RED_OR_BLUE'
    if alt_to_X == 'USE_RED':
        return 'USE_BLUE'
    if alt_to_Y == 'USE_RED':
        return 'USE_RED'
    assert(False)
if (A is not S.localroot
    and B is not S.localroot):
    #print("4.0")
    if (S is A.localroot or S is B.localroot):
        #print("4.05")
        if A.topo_order < B.topo_order:
            #print("4.05.1")
            if (alt_to_X == 'USE_BLUE' and alt_to_Y == 'USE_RED'):
                return 'USE_RED_OR_BLUE'
            if alt_to_X == 'USE_BLUE':
                return 'USE_BLUE'
            if alt_to_Y == 'USE_RED':
                return 'USE_RED'
            assert(False)
        else:
            #print("4.05.2")
            if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_BLUE'):
                return 'USE_RED_OR_BLUE'
            if alt_to_X == 'USE_RED':
                return 'USE_BLUE'
            if alt_to_Y == 'USE_BLUE':
                return 'USE_RED'
            assert(False)
    if A.LOWER:
        #print("4.1")
        if B.HIGHER:

```



```
#print("4.1.1")
if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_BLUE'):
    return 'USE_RED_OR_BLUE'
if alt_to_X == 'USE_RED':
    return 'USE_BLUE'
if alt_to_Y == 'USE_BLUE':
    return 'USE_RED'
assert(False)
if B.LOWER:
    #print("4.1.2")
    if A.topo_order < B.topo_order:
        #print("4.1.2.1")
        if (alt_to_X == 'USE_BLUE'
            and alt_to_Y == 'USE_RED'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_BLUE':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_RED':
            return 'USE_RED'
        assert(False)
    else:
        #print("4.1.2.2")
        if (alt_to_X == 'USE_RED'
            and alt_to_Y == 'USE_BLUE'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_RED':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_BLUE':
            return 'USE_RED'
        assert(False)
else:
    #print("4.1.3")
    if (F.LOWER and not F.HIGHER
        and F.topo_order > A.topo_order):
        #print("4.1.3.1")
        return 'USE_RED'
    else:
        #print("4.1.3.2")
        return 'USE_BLUE'
if A.HIGHER:
    #print("4.2")
    if B.HIGHER:
        #print("4.2.1")
        if A.topo_order < B.topo_order:
            #print("4.2.1.1")
            if (alt_to_X == 'USE_BLUE'
                and alt_to_Y == 'USE_RED'):
                return 'USE_RED_OR_BLUE'
```

```

        if alt_to_X == 'USE_BLUE':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_RED':
            return 'USE_RED'
        assert(False)
    else:
        #print("4.2.1.2")
        if (alt_to_X == 'USE_RED'
            and alt_to_Y == 'USE_BLUE'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_RED':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_BLUE':
            return 'USE_RED'
        assert(False)
if B.LOWER:
    #print("4.2.2")
    if (alt_to_X == 'USE_BLUE'
        and alt_to_Y == 'USE_RED'):
        return 'USE_RED_OR_BLUE'
    if alt_to_X == 'USE_BLUE':
        return 'USE_BLUE'
    if alt_to_Y == 'USE_RED':
        return 'USE_RED'
    assert(False)
else:
    #print("4.2.3")
    if (F.HIGHER and not F.LOWER
        and F.topo_order < A.topo_order):
        return 'USE_RED'
    else:
        return 'USE_BLUE'
else:
    #print("4.3")
    if B.LOWER:
        #print("4.3.1")
        if (F.LOWER and not F.HIGHER
            and F.topo_order > B.topo_order):
            return 'USE_BLUE'
        else:
            return 'USE_RED'
    if B.HIGHER:
        #print("4.3.2")
        if (F.HIGHER and not F.LOWER
            and F.topo_order < B.topo_order):
            return 'USE_BLUE'
        else:
            return 'USE_RED'

```

```
    else:
        #print("4.3.3")
        if A.topo_order < B.topo_order:
            #print("4.3.3.1")
            if (alt_to_X == 'USE_BLUE'
                and alt_to_Y == 'USE_RED'):
                return 'USE_RED_OR_BLUE'
            if alt_to_X == 'USE_BLUE':
                return 'USE_BLUE'
            if alt_to_Y == 'USE_RED':
                return 'USE_RED'
            assert(False)
        else:
            #print("4.3.3.2")
            if (alt_to_X == 'USE_RED'
                and alt_to_Y == 'USE_BLUE'):
                return 'USE_RED_OR_BLUE'
            if alt_to_X == 'USE_RED':
                return 'USE_BLUE'
            if alt_to_Y == 'USE_BLUE':
                return 'USE_RED'
            assert(False)
    assert(False)

def Compute_Primary_NHs_For_One_Src_To_Named_Proxy_Nodes(topo,src):
    for prefix in topo.named_proxy_dict:
        P = topo.named_proxy_dict[prefix]
        min_total_pref_cost = 2147483647
        for (adv_node, prefix_cost) in P.node_prefix_cost_list:
            total_pref_cost = (adv_node.primary_spf_metric
                               + prefix_cost)
            if total_pref_cost < min_total_pref_cost:
                min_total_pref_cost = total_pref_cost
                Copy_List_Items(P.primary_next_hops,
                               adv_node.primary_next_hops)
            elif total_pref_cost == min_total_pref_cost:
                for nh_intf in adv_node.primary_next_hops:
                    Add_Item_To_List_If_New(P.primary_next_hops,
                                             nh_intf)

def Select_Alts_For_One_Src_To_Named_Proxy_Nodes(topo,src):
    for prefix in topo.named_proxy_dict:
        P = topo.named_proxy_dict[prefix]
        P.alt_list = []
        for failed_intf in P.primary_next_hops:
            alt = Alternate()
            alt.failed_intf = failed_intf
            if failed_intf not in src.island_intf_list:
```

```
        alt.info = 'PRIM_NH_FOR_PROXY_NODE_NOT_IN_ISLAND'
    elif P.pnar1 is None:
        alt.info = 'NO_PNARS_EXIST_FOR_THIS_PREFIX'
    elif src is P.pnar1.node:
        alt.info = 'SRC_IS_PNAR'
    elif P.pnar2 is not None and src is P.pnar2.node:
        alt.info = 'SRC_IS_PNAR'
    elif P.pnar2 is None:
        #inherit alternates from the only pnar.
        alt.info = Select_Alternates(P.pnar1.node,
                                     failed_intf.remote_node, failed_intf)
    elif failed_intf in src.island_intf_list:
        alt.info = Select_Alternates_Proxy_Node(P,
                                                  failed_intf.remote_node, failed_intf)

    if alt.info == 'USE_RED_OR_BLUE':
        alt.red_or_blue = \
            random.choice(['USE_RED', 'USE_BLUE'])
    if (alt.info == 'USE_BLUE'
        or alt.red_or_blue == 'USE_BLUE'):
        Copy_List_Items(alt.nh_list, P.blue_next_hops)
        alt.fec = 'BLUE'
        alt.prot = 'NODE_PROTECTION'
    elif (alt.info == 'USE_RED'
          or alt.red_or_blue == 'USE_RED'):
        Copy_List_Items(alt.nh_list, P.red_next_hops)
        alt.fec = 'RED'
        alt.prot = 'NODE_PROTECTION'
    elif (alt.info == 'PRIM_NH_IS_D_OR_OP_FOR_D'
          or alt.info == 'PRIM_NH_IS_OP_FOR_BOTH_X_AND_Y'):
        if failed_intf.OUTGOING and failed_intf.INCOMING:
            # cut-link: if there are parallel cut links, use
            # the link(s) with lowest metric that are not
            # primary intf or None
            cand_alt_list = [None]
            min_metric = 2147483647
            for intf in src.island_intf_list:
                if ( intf is not failed_intf and
                    (intf.remote_node is
                     failed_intf.remote_node)):
                    if intf.metric < min_metric:
                        cand_alt_list = [intf]
                        min_metric = intf.metric
                    elif intf.metric == min_metric:
                        cand_alt_list.append(intf)
            if cand_alt_list != [None]:
                alt.fec = 'GREEN'
                alt.prot = 'PARALLEL_CUTLINK'
```

```
        else:
            alt.fec = 'NO_ALTERNATE'
            alt.prot = 'NO_PROTECTION'
            Copy_List_Items(alt.nh_list, cand_alt_list)
    else:
        # set Z as the node to inherit blue next-hops from
        if alt.info == 'PRIM_NH_IS_D_OR_OP_FOR_D':
            Z = P.pnar1.node
        else:
            Z = P
        if failed_intf in Z.red_next_hops:
            Copy_List_Items(alt.nh_list, Z.blue_next_hops)
            alt.fec = 'BLUE'
            alt.prot = 'LINK_PROTECTION'
        else:
            assert(failed_intf in Z.blue_next_hops)
            Copy_List_Items(alt.nh_list, Z.red_next_hops)
            alt.fec = 'RED'
            alt.prot = 'LINK_PROTECTION'

    elif alt.info == 'PRIM_NH_FOR_PROXY_NODE_NOT_IN_ISLAND':
        if (P.pnar2 == None and src is P.pnar1.node):
            #MRT Island is singly connected to non-island dest
            alt.fec = 'NO_ALTERNATE'
            alt.prot = 'NO_PROTECTION'
        elif P.node_id in src.blue_to_green_nh_dict:
            # blue to P goes to failed LFIN so use red to P
            Copy_List_Items(alt.nh_list, P.red_next_hops)
            alt.fec = 'RED'
            alt.prot = 'LINK_PROTECTION'
        elif P.node_id in src.red_to_green_nh_dict:
            # red to P goes to failed LFIN so use blue to P
            Copy_List_Items(alt.nh_list, P.blue_next_hops)
            alt.fec = 'BLUE'
            alt.prot = 'LINK_PROTECTION'
        else:
            Copy_List_Items(alt.nh_list, P.blue_next_hops)
            alt.fec = 'BLUE'
            alt.prot = 'LINK_PROTECTION'
    elif alt.info == 'TEMP_NO_ALTERNATE':
        alt.fec = 'NO_ALTERNATE'
        alt.prot = 'NO_PROTECTION'

    P.alt_list.append(alt)

def Run_Basic_MRT_for_One_Source(topo, src):
    MRT_Island_Identification(topo, src, 0, 0)
    Set_Island_Intf_and_Node_Lists(topo)
```

```
Set_GADAG_Root(topo,src)
Sort_Interfaces(topo)
Run_Lowpoint(topo)
Assign_Remaining_Lowpoint_Parents(topo)
Construct_GADAG_via_Lowpoint(topo)
Run_Assign_Block_ID(topo)
Add_Undirected_Links(topo)
Compute_MRT_NH_For_One_Src_To_Island_Dests(topo,src)
Store_MRT_Nexthops_For_One_Src_To_Island_Dests(topo,src)
Select_Alts_For_One_Src_To_Island_Dests(topo,src)
Store_Primary_and_Alts_For_One_Src_To_Island_Dests(topo,src)

def Store_GADAG_and_Named_Proxies_Once(topo):
    for node in topo.node_list:
        for intf in node.intf_list:
            if intf.OUTGOING:
                intf.SIMULATION_OUTGOING = True
            else:
                intf.SIMULATION_OUTGOING = False
    for prefix in topo.named_proxy_dict:
        P = topo.named_proxy_dict[prefix]
        topo.stored_named_proxy_dict[prefix] = P

def Run_Basic_MRT_for_All_Sources(topo):
    for src in topo.node_list:
        Reset_Computed_Node_and_Intf_Values(topo)
        Run_Basic_MRT_for_One_Source(topo,src)
        if src is topo.gadag_root:
            Store_GADAG_and_Named_Proxies_Once(topo)

def Run_MRT_for_One_Source(topo, src):
    MRT_Island_Identification(topo, src, 0, 0)
    Set_Island_Intf_and_Node_Lists(topo)
    Set_GADAG_Root(topo,src)
    Sort_Interfaces(topo)
    Run_Lowpoint(topo)
    Assign_Remaining_Lowpoint_Parents(topo)
    Construct_GADAG_via_Lowpoint(topo)
    Run_Assign_Block_ID(topo)
    Add_Undirected_Links(topo)
    Compute_MRT_NH_For_One_Src_To_Island_Dests(topo,src)
    Store_MRT_Nexthops_For_One_Src_To_Island_Dests(topo,src)
    Select_Alts_For_One_Src_To_Island_Dests(topo,src)
    Store_Primary_and_Alts_For_One_Src_To_Island_Dests(topo,src)
    Create_Basic_Named_Proxy_Nodes(topo)
    Attach_Named_Proxy_Nodes(topo)
    Compute_MRT_NHs_For_One_Src_To_Named_Proxy_Nodes(topo,src)
    Store_MRT_NHs_For_One_Src_To_Named_Proxy_Nodes(topo,src)
```

```

    Compute_Primary_NHs_For_One_Src_To_Named_Proxy_Nodes(topo,src)
    Store_Primary_NHs_For_One_Src_To_Named_Proxy_Nodes(topo,src)
    Select_Alts_For_One_Src_To_Named_Proxy_Nodes(topo,src)
    Store_Alts_For_One_Src_To_Named_Proxy_Nodes(topo,src)

def Run_Prim_SPF_for_One_Source(topo,src):
    Normal_SPF(topo, src)
    Store_Primary_NHs_For_One_Source_To_Nodes(topo,src)
    Create_Basic_Named_Proxy_Nodes(topo)
    Compute_Primary_NHs_For_One_Src_To_Named_Proxy_Nodes(topo,src)
    Store_Primary_NHs_For_One_Src_To_Named_Proxy_Nodes(topo,src)

def Run_MRT_for_All_Sources(topo):
    for src in topo.node_list:
        Reset_Computed_Node_and_Intf_Values(topo)
        if src in topo.island_node_list_for_test_gr:
            # src runs MRT if it is in same MRT island as test_gr
            Run_MRT_for_One_Source(topo,src)
            if src is topo.gadag_root:
                Store_GADAG_and_Named_Proxies_Once(topo)
        else:
            # src still runs SPF if not in MRT island
            Run_Prim_SPF_for_One_Source(topo,src)

def Write_Output_To_Files(topo,file_prefix):
    Write_GADAG_To_File(topo,file_prefix)
    Write_Both_MRTs_For_All_Dests_To_File(topo,file_prefix)
    Write_Alternates_For_All_Dests_To_File(topo,file_prefix)

def Create_Basic_Topology_Input_File(filename):
    data = [[01,02,10],[02,03,10],[03,04,11],[04,05,10,20],[05,06,10],
            [06,07,10],[06,07,10],[06,07,15],[07,01,10],[07,51,10],
            [51,52,10],[52,53,10],[53,03,10],[01,55,10],[55,06,10],
            [04,12,10],[12,13,10],[13,14,10],[14,15,10],[15,16,10],
            [16,17,10],[17,04,10],[05,76,10],[76,77,10],[77,78,10],
            [78,79,10],[79,77,10]]
    with open(filename + '.csv', 'w') as topo_file:
        for item in data:
            if len(item) > 3:
                line = (str(item[0])+','+','+str(item[1])+','+','+
                        str(item[2])+','+','+str(item[3])+'\n')
            else:
                line = (str(item[0])+','+','+str(item[1])+','+','+
                        str(item[2])+'\n')
            topo_file.write(line)

def Create_Complex_Topology_Input_File(filename):
    data = [[01,02,10],[02,03,10],[03,04,11],[04,05,10,20],[05,06,10],

```

```

        [06,07,10],[06,07,10],[06,07,15],[07,01,10],[07,51,10],
        [51,52,10],[52,53,10],[53,03,10],[01,55,10],[55,06,10],
        [04,12,10],[12,13,10],[13,14,10],[14,15,10],[15,16,10],
        [16,17,10],[17,04,10],[05,76,10],[76,77,10],[77,78,10],
        [78,79,10],[79,77,10]]
    with open(filename + '.csv', 'w') as topo_file:
        for item in data:
            if len(item) > 3:
                line = (str(item[0])+',' +str(item[1])+',' +
                        str(item[2])+',' +str(item[3])+'\n')
            else:
                line = (str(item[0])+',' +str(item[1])+',' +
                        str(item[2])+'\n')
            topo_file.write(line)

    data = [[01,0],[02,0],[03,0],[04,0],[05,0],
            [06,0],[07,0],
            [51,0],[55,0],
            [12,0],[13,0],[14,0],[15,0],
            [16,0],[17,0],[76,0],[77,0],
            [78,0],[79,0]]
    with open(filename + '.profile', 'w') as topo_file:
        for item in data:
            line = (str(item[0])+',' +str(item[1])+'\n')
            topo_file.write(line)

    data = [[2001,05,100],[2001,07,120],[2001,03,130],
            [2002,13,100],[2002,15,110],
            [2003,52,100],[2003,78,100]]
    with open(filename + '.prefix', 'w') as topo_file:
        for item in data:
            line = (str(item[0])+',' +str(item[1])+',' +
                    str(item[2])+'\n')
            topo_file.write(line)

def Generate_Basic_Topology_and_Run_MRT():
    this_gadag_root = 3
    Create_Basic_Topology_Input_File('basic_topo_input')
    topo = Create_Topology_From_File('basic_topo_input')
    res_file_base = 'basic_topo'
    Compute_Island_Node_List_For_Test_GR(topo, this_gadag_root)
    Raise_GADAG_Root_Selection_Priority(topo,this_gadag_root)
    Run_Basic_MRT_for_All_Sources(topo)
    Write_Output_To_Files(topo, res_file_base)

def Generate_Complex_Topology_and_Run_MRT():
    this_gadag_root = 3
    Create_Complex_Topology_Input_File('complex_topo_input')

```



```
topo = Create_Topology_From_File('complex_topo_input')
Add_Profile_IDs_from_File(topo,'complex_topo_input')
Add_Prefix_Advertisements_From_File(topo,'complex_topo_input')
Compute_Island_Node_List_For_Test_GR(topo, this_gadag_root)
Add_Prefixes_for_Non_Island_Nodes(topo)
res_file_base = 'complex_topo'
Raise_GADAG_Root_Selection_Priority(topo,this_gadag_root)
Run_MRT_for_All_Sources(topo)
Write_Output_To_Files(topo, res_file_base)

Generate_Basic_Topology_and_Run_MRT()

Generate_Complex_Topology_and_Run_MRT()

<CODE ENDS>
```

Appendix B. Constructing a GADAG using SPF's

The basic idea in this method for constructing a GADAG is to use slightly-modified SPF computations to find ears. In every block, an SPF computation is first done to find a cycle from the local root and then SPF computations in that block find ears until there are no more interfaces to be explored. The used result from the SPF computation is the path of interfaces indicated by following the previous hops from the minimized IN_GADAG node back to the SPF root.

To do this, first all cut-vertices must be identified and local-roots assigned as specified in Figure 12.

The slight modifications to the SPF are as follows. The root of the block is referred to as the block-root; it is either the GADAG root or a cut-vertex.

- a. The SPF is rooted at a neighbor *x* of an IN_GADAG node *y*. All links between *y* and *x* are marked as TEMP_UNUSABLE. They should not be used during the SPF computation.
- b. If *y* is not the block-root, then it is marked TEMP_UNUSABLE. It should not be used during the SPF computation. This prevents ears from starting and ending at the same node and avoids cycles; the exception is because cycles to/from the block-root are acceptable and expected.
- c. Do not explore links to nodes whose local-root is not the block-root. This keeps the SPF confined to the particular block.
- d. Terminate when the first IN_GADAG node *z* is minimized.

- e. Respect the existing directions (e.g. INCOMING, OUTGOING, UNDIRECTED) already specified for each interface.

```

Mod_SPF(spf_root, block_root)
  Initialize spf_heap to empty
  Initialize nodes' spf_metric to infinity
  spf_root.spf_metric = 0
  insert(spf_heap, spf_root)
  found_in_gadag = false
  while (spf_heap is not empty) and (found_in_gadag is false)
    min_node = remove_lowest(spf_heap)
    if min_node.IN_GADAG
      found_in_gadag = true
    else
      foreach interface intf of min_node
        if ((intf.OUTGOING or intf.UNDIRECTED) and
            ((intf.remote_node.localroot is block_root) or
             (intf.remote_node is block_root)) and
            (intf.remote_node is not TEMP_UNUSABLE) and
            (intf is not TEMP_UNUSABLE))
          path_metric = min_node.spf_metric + intf.metric
          if path_metric < intf.remote_node.spf_metric
            intf.remote_node.spf_metric = path_metric
            intf.remote_node.spf_prev_intf = intf
            insert_or_update(spf_heap, intf.remote_node)
  return min_node

SPF_for_Ear(cand_intf.local_node, cand_intf.remote_node, block_root,
            method)
  Mark all interfaces between cand_intf.remote_node
    and cand_intf.local_node as TEMP_UNUSABLE
  if cand_intf.local_node is not block_root
    Mark cand_intf.local_node as TEMP_UNUSABLE
  Initialize ear_list to empty
  end_ear = Mod_SPF(spf_root, block_root)
  y = end_ear.spf_prev_hop
  while y.local_node is not spf_root
    add_to_list_start(ear_list, y)
    y.local_node.IN_GADAG = true
    y = y.local_node.spf_prev_intf
  if(method is not hybrid)
    Set_Ear_Direction(ear_list, cand_intf.local_node,
                      end_ear, block_root)
  Clear TEMP_UNUSABLE from all interfaces between
    cand_intf.remote_node and cand_intf.local_node

```

```
    Clear TEMP_UNUSABLE from cand_intf.local_node  
    return end_ear
```

Figure 31: Modified SPF for GADAG construction

Assume that an ear is found by going from y to x and then running an SPF that terminates by minimizing z (e.g. $y \leftarrow x \dots q \leftarrow z$). Now it is necessary to determine the direction of the ear; if $y \ll z$, then the path should be $y \rightarrow x \dots q \rightarrow z$ but if $y \gg z$, then the path should be $y \leftarrow x \dots q \leftarrow z$. In Section 5.5, the same problem was handled by finding all ears that started at a node before looking at ears starting at nodes higher in the partial order. In this GADAG construction method, using that approach could mean that new ears aren't added in order of their total cost since all ears connected to a node would need to be found before additional nodes could be found.

The alternative is to track the order relationship of each node with respect to every other node. This can be accomplished by maintaining two sets of nodes at each node. The first set, `Higher_Nodes`, contains all nodes that are known to be ordered above the node. The second set, `Lower_Nodes`, contains all nodes that are known to be ordered below the node. This is the approach used in this GADAG construction method.

```

Set_Ear_Direction(ear_list, end_a, end_b, block_root)
// Default of A_TO_B for the following cases:
// (a) end_a and end_b are the same (root)
// or (b) end_a is in end_b's Lower_Nodes
// or (c) end_a and end_b were unordered with respect to each
//      other
direction = A_TO_B
if (end_b is block_root) and (end_a is not end_b)
    direction = B_TO_A
else if end_a is in end_b.Higher_Nodes
    direction = B_TO_A
if direction is B_TO_A
    foreach interface i in ear_list
        i.UNDIRECTED = false
        i.INCOMING = true
        i.remote_intf.UNDIRECTED = false
        i.remote_intf.OUTGOING = true
else
    foreach interface i in ear_list
        i.UNDIRECTED = false
        i.OUTGOING = true
        i.remote_intf.UNDIRECTED = false
        i.remote_intf.INCOMING = true
if end_a is end_b
    return
// Next, update all nodes' Lower_Nodes and Higher_Nodes
if (end_a is in end_b.Higher_Nodes)
    foreach node x where x.localroot is block_root
        if end_a is in x.Lower_Nodes
            foreach interface i in ear_list
                add i.remote_node to x.Lower_Nodes
        if end_b is in x.Higher_Nodes
            foreach interface i in ear_list
                add i.local_node to x.Higher_Nodes
else
    foreach node x where x.localroot is block_root
        if end_b is in x.Lower_Nodes
            foreach interface i in ear_list
                add i.local_node to x.Lower_Nodes
        if end_a is in x.Higher_Nodes
            foreach interface i in ear_list
                add i.remote_node to x.Higher_Nodes

```

Figure 32: Algorithm to assign links of an ear direction

A goal of this GADAG construction method is to find the shortest cycles and ears. An ear is started by going to a neighbor *x* of an IN_GADAG node *y*. The path from *x* to an IN_GADAG node is minimal,

since it is computed via SPF. Since a shortest path is made of shortest paths, to find the shortest ears requires reaching from the set of IN_GADAG nodes to the closest node that isn't IN_GADAG. Therefore, an ordered tree is maintained of interfaces that could be explored from the IN_GADAG nodes. The interfaces are ordered by their characteristics of metric, local loopback address, remote loopback address, and ifindex, based on the Interface_Compare function defined in Figure 14.

This GADAG construction method ignores interfaces picked from the ordered list that belong to the block root if the block in which the interface is present already has an ear that has been computed. This is necessary since we allow at most one incoming interface to a block root in each block. This requirement stems from the way next-hops are computed as was seen in Section 5.7. After any ear gets computed, we traverse the newly added nodes to the GADAG and insert interfaces whose far end is not yet on the GADAG to the ordered tree for later processing.

Finally, cut-links are a special case because there is no point in doing an SPF on a block of 2 nodes. The algorithm identifies cut-links simply as links where both ends of the link are cut-vertices. Cut-links can simply be added to the GADAG with both OUTGOING and INCOMING specified on their interfaces.

```

add_eligible_interfaces_of_node(ordered_intfs_tree,node)
  for each interface of node
    if intf.remote_node.IN_GADAG is false
      insert(intf,ordered_intfs_tree)

check_if_block_has_ear(x,block_id)
  block_has_ear = false
  for all interfaces of x
    if ( (intf.remote_node.block_id == block_id) &&
        intf.remote_node.IN_GADAG )
      block_has_ear = true
  return block_has_ear

Construct_GADAG_via_SPF(topology, root)
  Compute_Localroot (root,root)
  Assign_Block_ID(root,0)
  root.IN_GADAG = true
  add_eligible_interfaces_of_node(ordered_intfs_tree,root)
  while ordered_intfs_tree is not empty
    cand_intf = remove_lowest(ordered_intfs_tree)
    if cand_intf.remote_node.IN_GADAG is false
      if L(cand_intf.remote_node) == D(cand_intf.remote_node)
        // Special case for cut-links

```

```

        cand_intf.UNDIRECTED = false
        cand_intf.remote_intf.UNDIRECTED = false
        cand_intf.OUTGOING = true
        cand_intf.INCOMING = true
        cand_intf.remote_intf.OUTGOING = true
        cand_intf.remote_intf.INCOMING = true
        cand_intf.remote_node.IN_GADAG = true
        add_eligible_interfaces_of_node(
            ordered_intfs_tree, cand_intf.remote_node)
    else
        if (cand_intf.remote_node.local_root ==
            cand_intf.local_node) &&
            check_if_block_has_ear(cand_intf.local_node,
                                   cand_intf.remote_node.block_id))
            /* Skip the interface since the block root
               already has an incoming interface in the
               block */
        else
            ear_end = SPF_for_Ear(cand_intf.local_node,
                                   cand_intf.remote_node,
                                   cand_intf.remote_node.localroot,
                                   SPF method)
            y = ear_end.spf_prev_hop
            while y.local_node is not cand_intf.local_node
                add_eligible_interfaces_of_node(
                    ordered_intfs_tree, y.local_node)
                y = y.local_node.spf_prev_intf

```

Figure 33: SPF-based method for GADAG construction

Appendix C. Constructing a GADAG using a hybrid method

The idea of this method is to combine the salient features of the lowpoint inheritance and SPF methods. To this end, we process nodes as they get added to the GADAG just like in the lowpoint inheritance by maintaining a stack of nodes. This ensures that we do not need to maintain lower and higher sets at each node to ascertain ear directions since the ears will always be directed from the node being processed towards the end of the ear. To compute the ear however, we resort to an SPF to have the possibility of better ears (path lengths) thus giving more flexibility than the restricted use of lowpoint/dfs parents.

Regarding ears involving a block root, unlike the SPF method which ignored interfaces of the block root after the first ear, in the hybrid method we would have to process all interfaces of the block root before moving on to other nodes in the block since the direction

of an ear is pre-determined. Thus, whenever the block already has an ear computed, and we are processing an interface of the block root, we mark the block root as unusable before the SPF run that computes the ear. This ensures that the SPF terminates at some node other than the block-root. This in turn guarantees that the block-root has only one incoming interface in each block, which is necessary for correctly computing the next-hops on the GADAG.

As in the SPF gadag, bridge ears are handled as a special case.

The entire algorithm is shown below in Figure 34

```

find_spf_stack_ear(stack, x, y, xy_intf, block_root)
  if L(y) == D(y)
    // Special case for cut-links
    xy_intf.UNDIRECTED = false
    xy_intf.remote_intf.UNDIRECTED = false
    xy_intf.OUTGOING = true
    xy_intf.INCOMING = true
    xy_intf.remote_intf.OUTGOING = true
    xy_intf.remote_intf.INCOMING = true
    xy_intf.remote_node.IN_GADAG = true
    push y onto stack
    return
  else
    if (y.local_root == x) &&
      check_if_block_has_ear(x,y.block_id)
      //Avoid the block root during the SPF
      Mark x as TEMP_UNUSABLE
    end_ear = SPF_for_Ear(x,y,block_root,hybrid)
    If x was set as TEMP_UNUSABLE, clear it
    cur = end_ear
    while (cur != y)
      intf = cur.spf_prev_hop
      prev = intf.local_node
      intf.UNDIRECTED = false
      intf.remote_intf.UNDIRECTED = false
      intf.OUTGOING = true
      intf.remote_intf.INCOMING = true
      push prev onto stack
      cur = prev
    xy_intf.UNDIRECTED = false
    xy_intf.remote_intf.UNDIRECTED = false
    xy_intf.OUTGOING = true
    xy_intf.remote_intf.INCOMING = true
    return

Construct_GADAG_via_hybrid(topology,root)

```

```
Compute_Localroot (root,root)
Assign_Block_ID(root,0)
root.IN_GADAG = true
Initialize Stack to empty
push root onto Stack
while (Stack is not empty)
    x = pop(Stack)
    for each interface intf of x
        y = intf.remote_node
        if y.IN_GADAG is false
            find_spf_stack_ear(stack, x, y, intf, y.block_root)
```

Figure 34: Hybrid GADAG construction method

Authors' Addresses

Gabor Sandor Enyedi
Ericsson
Konyves Kalman krt 11
Budapest 1097
Hungary

Email: Gabor.Sandor.Enyedi@ericsson.com

Andras Csaszar
Ericsson
Konyves Kalman krt 11
Budapest 1097
Hungary

Email: Andras.Csaszar@ericsson.com

Alia Atlas
Juniper Networks
10 Technology Park Drive
Westford, MA 01886
USA

Email: akatlas@juniper.net

Chris Bowers
Juniper Networks
1194 N. Mathilda Ave.
Sunnyvale, CA 94089
USA

Email: cbowers@juniper.net

Abishek Gopalan
University of Arizona
1230 E Speedway Blvd.
Tucson, AZ 85721
USA

Email: abishek@ece.arizona.edu

Routing Area Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 8, 2016

A. Atlas
C. Bowers
Juniper Networks
G. Enyedi
Ericsson
February 5, 2016

An Architecture for IP/LDP Fast-Reroute Using Maximally Redundant Trees
draft-ietf-rtgwg-mrt-frr-architecture-10

Abstract

This document defines the architecture for IP and LDP Fast-Reroute using Maximally Redundant Trees (MRT-FRR). MRT-FRR is a technology that gives link-protection and node-protection with 100% coverage in any network topology that is still connected after the failure.

Status of This Memo

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

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

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

This Internet-Draft will expire on August 8, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Importance of 100% Coverage	4
1.2. Partial Deployment and Backwards Compatibility	5
2. Requirements Language	5
3. Terminology	5
4. Maximally Redundant Trees (MRT)	7
5. Maximally Redundant Trees (MRT) and Fast-Reroute	9
6. Unicast Forwarding with MRT Fast-Reroute	9
6.1. Introduction to MRT Forwarding Options	10
6.1.1. MRT LDP labels	10
6.1.1.1. Topology-scoped FEC encoded using a single label (Option 1A)	10
6.1.1.2. Topology and FEC encoded using a two label stack (Option 1B)	11
6.1.1.3. Compatibility of MRT LDP Label Options 1A and 1B	12
6.1.1.4. Required support for MRT LDP Label options	12
6.1.2. MRT IP tunnels (Options 2A and 2B)	12
6.2. Forwarding LDP Unicast Traffic over MRT Paths	13
6.2.1. Forwarding LDP traffic using MRT LDP Label Option 1A	13
6.2.2. Forwarding LDP traffic using MRT LDP Label Option 1B	14
6.2.3. Other considerations for forwarding LDP traffic using MRT LDP Labels	14
6.2.4. Required support for LDP traffic	14
6.3. Forwarding IP Unicast Traffic over MRT Paths	14
6.3.1. Tunneling IP traffic using MRT LDP Labels	15
6.3.1.1. Tunneling IP traffic using MRT LDP Label Option 1A	15
6.3.1.2. Tunneling IP traffic using MRT LDP Label Option 1B	15
6.3.2. Tunneling IP traffic using MRT IP Tunnels	16
6.3.3. Required support for IP traffic	16
7. MRT Island Formation	16
7.1. IGP Area or Level	17
7.2. Support for a specific MRT profile	17
7.3. Excluding additional routers and interfaces from the MRT Island	18
7.3.1. Existing IGP exclusion mechanisms	18
7.3.2. MRT-specific exclusion mechanism	19
7.4. Connectivity	19
7.5. Algorithm for MRT Island Identification	19
8. MRT Profile	19
8.1. MRT Profile Options	19
8.2. Router-specific MRT paramaters	21

8.3. Default MRT profile	21
9. LDP signaling extensions and considerations	22
10. Inter-area Forwarding Behavior	22
10.1. ABR Forwarding Behavior with MRT LDP Label Option 1A . .	23
10.1.1. Motivation for Creating the Rainbow-FEC	24
10.2. ABR Forwarding Behavior with IP Tunneling (option 2) . .	24
10.3. ABR Forwarding Behavior with MRT LDP Label option 1B . .	25
11. Prefixes Multiply Attached to the MRT Island	26
11.1. Protecting Multi-Homed Prefixes using Tunnel Endpoint Selection	28
11.2. Protecting Multi-Homed Prefixes using Named Proxy-Nodes	29
11.3. MRT Alternates for Destinations Outside the MRT Island .	31
12. Network Convergence and Preparing for the Next Failure . . .	31
12.1. Micro-loop prevention and MRTs	32
12.2. MRT Recalculation for the Default MRT Profile	33
13. Implementation Status	34
14. Operational Considerations	35
14.1. Verifying Forwarding on MRT Paths	35
14.2. Traffic Capacity on Backup Paths	36
14.3. MRT IP Tunnel Loopback Address Management	38
14.4. MRT-FRR in a Network with Degraded Connectivity	38
14.5. Partial Deployment of MRT-FRR in a Network	38
15. Acknowledgements	39
16. IANA Considerations	39
17. Security Considerations	40
18. Contributors	40
19. References	41
19.1. Normative References	41
19.2. Informative References	42
Appendix A. Inter-level Forwarding Behavior for IS-IS	43
Appendix B. General Issues with Area Abstraction	44
Authors' Addresses	45

1. Introduction

This document describes a solution for IP/LDP fast-reroute [RFC5714]. MRT-FRR creates two alternate forwarding trees which are distinct from the primary next-hop forwarding used during stable operation. These two trees are maximally diverse from each other, providing link and node protection for 100% of paths and failures as long as the failure does not cut the network into multiple pieces. This document defines the architecture for IP/LDP fast-reroute with MRT.

[I-D.ietf-rtgwg-mrt-frr-algorithm] describes how to compute maximally redundant trees using a specific algorithm, the MRT Lowpoint algorithm. The MRT Lowpoint algorithm is used by a router that supports the Default MRT Profile, as specified in this document.

IP/LDP Fast-Reroute with MRT (MRT-FRR) uses two maximally diverse forwarding topologies to provide alternates. A primary next-hop should be on only one of the diverse forwarding topologies; thus, the other can be used to provide an alternate. Once traffic has been moved to one of the MRTs by one point of local repair (PLR), that traffic is not subject to further repair actions by another PLR, even in the event of multiple simultaneous failures. Therefore, traffic repaired by MRT-FRR will not loop between different PLRs responding to different simultaneous failures.

While MRT provides 100% protection for a single link or node failure, it may not protect traffic in the event of multiple simultaneous failures, nor does take into account Shared Risk Link Groups (SRLGs). Also, while the MRT Lowpoint algorithm is computationally efficient, it is also new. In order for MRT-FRR to function properly, all of the other nodes in the network that support MRT must correctly compute next-hops based on the same algorithm, and install the corresponding forwarding state. This is in contrast to other FRR methods where the calculation of backup paths generally involves repeated application of the simpler and widely-deployed shortest path first (SPF) algorithm, and backup paths themselves re-use the forwarding state used for shortest path forwarding of normal traffic. Section 14 provides operational guidance related to verification of MRT forwarding paths.

In addition to supporting IP and LDP unicast fast-reroute, the diverse forwarding topologies and guarantee of 100% coverage permit fast-reroute technology to be applied to multicast traffic as described in [I-D.atlas-rtgwg-mrt-mc-arch]. However, the current document does not address the multicast applications of MRTs.

1.1. Importance of 100% Coverage

Fast-reroute is based upon the single failure assumption - that the time between single failures is long enough for a network to reconverge and start forwarding on the new shortest paths. That does not imply that the network will only experience one failure or change.

It is straightforward to analyze a particular network topology for coverage. However, a real network does not always have the same topology. For instance, maintenance events will take links or nodes out of use. Simply costing out a link can have a significant effect on what loop-free alternates (LFAs) are available. Similarly, after a single failure has happened, the topology is changed and its associated coverage. Finally, many networks have new routers or links added and removed; each of those changes can have an effect on the coverage for topology-sensitive methods such as LFA and Remote

LFA. If fast-reroute is important for the network services provided, then a method that guarantees 100% coverage is important to accommodate natural network topology changes.

When a network needs to use Ordered FIB[RFC6976] or Nearside Tunneling[RFC5715] as a micro-loop prevention mechanism [RFC5715], then the whole IGP area needs to have alternates available. This allows the micro-loop prevention mechanism, which requires slower network convergence, to take the necessary time without adversely impacting traffic. Without complete coverage, traffic to the unprotected destinations will be dropped for significantly longer than with current convergence - where routers individually converge as fast as possible. See Section 12.1 for more discussion of micro-loop prevention and MRTs.

1.2. Partial Deployment and Backwards Compatibility

MRT-FRR supports partial deployment. Routers advertise their ability to support MRT. Inside the MRT-capable connected group of routers (referred to as an MRT Island), the MRTs are computed. Alternates to destinations outside the MRT Island are computed and depend upon the existence of a loop-free neighbor of the MRT Island for that destination. MRT Islands are discussed in detail in Section 7, and partial deployment is discussed in more detail in Section 14.5.

2. Requirements Language

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

3. Terminology

network graph: A graph that reflects the network topology where all links connect exactly two nodes and broadcast links have been transformed into the standard pseudo-node representation.

cut-link: A link whose removal partitions the network. A cut-link by definition must be connected between two cut-vertices. If there are multiple parallel links, then they are referred to as cut-links in this document if removing the set of parallel links would partition the network graph.

cut-vertex: A vertex whose removal partitions the network graph.

2-connected: A graph that has no cut-vertices. This is a graph that requires two nodes to be removed before the network is partitioned.

2-connected cluster: A maximal set of nodes that are 2-connected.

block: Either a 2-connected cluster, a cut-edge, or an isolated vertex.

Redundant Trees (RT): A pair of trees where the path from any node X to the root R along the first tree is node-disjoint with the path from the same node X to the root along the second tree. Redundant trees can always be computed in 2-connected graphs.

Maximally Redundant Trees (MRT): A pair of trees where the path from any node X to the root R along the first tree and the path from the same node X to the root along the second tree share the minimum number of nodes and the minimum number of links. Each such shared node is a cut-vertex. Any shared links are cut-links. In graphs that are not 2-connected, it is not possible to compute RTs. However, it is possible to compute MRTs. MRTs are maximally redundant in the sense that they are as redundant as possible given the constraints of the network graph.

Directed Acyclic Graph (DAG): A graph where all links are directed and there are no cycles in it.

Almost Directed Acyclic Graph (ADAG): A graph with one node designated as the root. The graph has the property that if all links incoming to the root were removed, then resulting graph would be a DAG.

Generalized ADAG (GADAG): A graph that is the combination of the ADAGs of all blocks.

MRT-Red: MRT-Red is used to describe one of the two MRTs; it is used to describe the associated forwarding topology and MPLS multi-topology identifier (MT-ID). Specifically, MRT-Red is the decreasing MRT where links in the GADAG are taken in the direction from a higher topologically ordered node to a lower one.

MRT-Blue: MRT-Blue is used to describe one of the two MRTs; it is used to describe the associated forwarding topology and MPLS MT-ID. Specifically, MRT-Blue is the increasing MRT where links in the GADAG are taken in the direction from a lower topologically ordered node to a higher one.

Rainbow MRT: It is useful to have an MPLS MT-ID that refers to the multiple MRT forwarding topologies and to the default forwarding topology. This is referred to as the Rainbow MRT MPLS MT-ID and is used by LDP to reduce signaling and permit the same label to always be advertised to all peers for the same (MT-ID, Prefix).

MRT Island: The set of routers that support a particular MRT profile and the links connecting them that support MRT.

Island Border Router (IBR): A router in the MRT Island that is connected to a router not in the MRT Island and both routers are in a common area or level.

Island Neighbor (IN): A router that is not in the MRT Island but is adjacent to an IBR and in the same area/level as the IBR.

named proxy-node: A proxy-node can represent a destination prefix that can be attached to the MRT Island via at least two routers. It is named if there is a way that traffic can be encapsulated to reach specifically that proxy node; this could be because there is an LDP FEC (Forwarding Equivalence Class) for the associated prefix or because MRT-Red and MRT-Blue IP addresses are advertised in an undefined fashion for that proxy-node.

4. Maximally Redundant Trees (MRT)

A pair of Maximally Redundant Trees is a pair of directed spanning trees that provides maximally disjoint paths towards their common root. Only links or nodes whose failure would partition the network (i.e. cut-links and cut-vertices) are shared between the trees. The MRT Lowpoint algorithm is given in [I-D.ietf-rtgwg-mrt-frr-algorithm]. This algorithm can be computed in $O(e + n \log n)$; it is less than three SPF's. This document describes how the MRTs can be used and not how to compute them.

MRT provides destination-based trees for each destination. Each router stores its normal primary next-hop(s) as well as MRT-Blue next-hop(s) and MRT-Red next-hop(s) toward each destination. The alternate will be selected between the MRT-Blue and MRT-Red.

The most important thing to understand about MRTs is that for each pair of destination-routed MRTs, there is a path from every node X to the destination D on the Blue MRT that is as disjoint as possible from the path on the Red MRT.

For example, in Figure 1, there is a network graph that is 2-connected in (a) and associated MRTs in (b) and (c). One can consider the paths from B to R; on the Blue MRT, the paths are B->F->D->E->R or B->C->D->E->R. On the Red MRT, the path is B->A->R. These are clearly link and node-disjoint. These MRTs are redundant trees because the paths are disjoint.

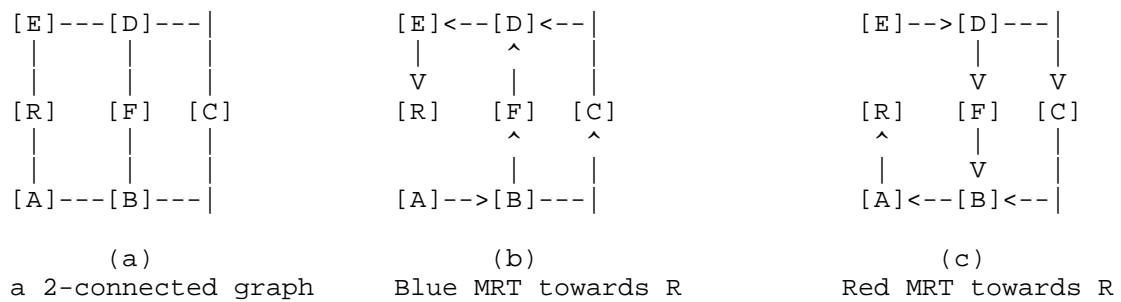


Figure 1: A 2-connected Network

By contrast, in Figure 2, the network in (a) is not 2-connected. If F, G or the link F \leftrightarrow G failed, then the network would be partitioned. It is clearly impossible to have two link-disjoint or node-disjoint paths from G, I or J to R. The MRTs given in (b) and (c) offer paths that are as disjoint as possible. For instance, the paths from B to R are the same as in Figure 1 and the path from G to R on the Blue MRT is G \rightarrow F \rightarrow D \rightarrow E \rightarrow R and on the Red MRT is G \rightarrow F \rightarrow B \rightarrow A \rightarrow R.

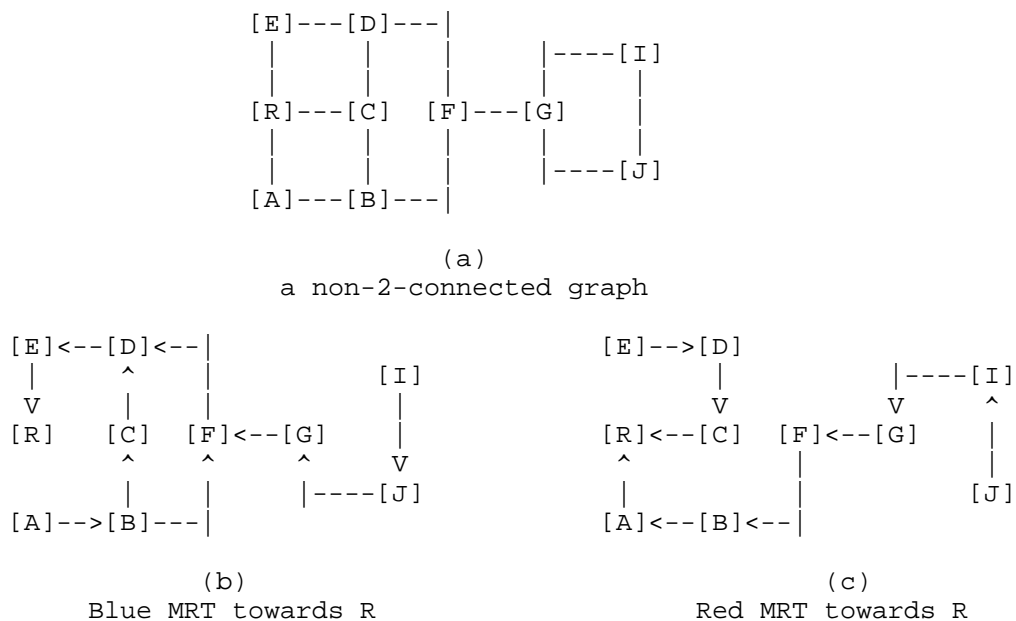


Figure 2: A non-2-connected network

5. Maximally Redundant Trees (MRT) and Fast-Reroute

In normal IGP routing, each router has its shortest path tree (SPT) to all destinations. From the perspective of a particular destination, D, this looks like a reverse SPT. To use maximally redundant trees, in addition, each destination D has two MRTs associated with it; by convention these will be called the MRT-Blue and MRT-Red. MRT-FRR is realized by using multi-topology forwarding. There is a MRT-Blue forwarding topology and a MRT-Red forwarding topology.

Any IP/LDP fast-reroute technique beyond LFA requires an additional dataplane procedure, such as an additional forwarding mechanism. The well-known options are multi-topology forwarding (used by MRT-FRR), tunneling (e.g. [RFC6981] or [RFC7490]), and per-interface forwarding (e.g. Loop-Free Failure Insensitive Routing in [EnyediThesis]).

When there is a link or node failure affecting, but not partitioning, the network, each node will still have at least one path via one of the MRTs to reach the destination D. For example, in Figure 2, C would normally forward traffic to R across the C->R link. If that C->R link fails, then C could use the Blue MRT path C->D->E->R.

As is always the case with fast-reroute technologies, forwarding does not change until a local failure is detected. Packets are forwarded along the shortest path. The appropriate alternate to use is pre-computed. [I-D.ietf-rtgwg-mrt-frr-algorithm] describes exactly how to determine whether the MRT-Blue next-hops or the MRT-Red next-hops should be the MRT alternate next-hops for a particular primary next-hop to a particular destination.

MRT alternates are always available to use. It is a local decision whether to use an MRT alternate, a Loop-Free Alternate or some other type of alternate.

As described in [RFC5286], when a worse failure than is anticipated happens, using LFAs that are not downstream neighbors can cause looping among alternates. Section 1.1 of [RFC5286] gives an example of link-protecting alternates causing a loop on node failure. Even if a worse failure than anticipated happens, the use of MRT alternates will not cause looping.

6. Unicast Forwarding with MRT Fast-Reroute

There are three possible types of routers involved in forwarding a packet along an MRT path. At the MRT ingress router, the packet leaves the shortest path to the destination and follows an MRT path

to the destination. In an FRR application, the MRT ingress router is the PLR. An MRT transit router takes a packet that arrives already associated with the particular MRT, and forwards it on that same MRT. In some situations (to be discussed later), the packet will need to leave the MRT path and return to the shortest path. This takes place at the MRT egress router. The MRT ingress and egress functionality may depend on the underlying type of packet being forwarded (LDP or IP). The MRT transit functionality is independent of the type of packet being forwarded. We first consider several MRT transit forwarding mechanisms. Then we look at how these forwarding mechanisms can be applied to carrying LDP and IP traffic.

6.1. Introduction to MRT Forwarding Options

The following options for MRT forwarding mechanisms are considered.

1. MRT LDP Labels

- A. Topology-scoped FEC encoded using a single label
- B. Topology and FEC encoded using a two label stack

2. MRT IP Tunnels

- A. MRT IPv4 Tunnels
- B. MRT IPv6 Tunnels

6.1.1. MRT LDP labels

We consider two options for the MRT forwarding mechanisms using MRT LDP labels.

6.1.1.1. Topology-scoped FEC encoded using a single label (Option 1A)

[RFC7307] provides a mechanism to distribute FEC-Label bindings scoped to a given MPLS topology (represented by MPLS MT-ID). To use multi-topology LDP to create MRT forwarding topologies, we associate two MPLS MT-IDs with the MRT-Red and MRT-Blue forwarding topologies, in addition to the default shortest path forwarding topology with MT-ID=0.

With this forwarding mechanism, a single label is distributed for each topology-scoped FEC. For a given FEC in the default topology (call it default-FEC-A), two additional topology-scoped FECs would be created, corresponding to the Red and Blue MRT forwarding topologies (call them red-FEC-A and blue-FEC-A). A router supporting this MRT transit forwarding mechanism advertises a different FEC-label binding

for each of the three topology-scoped FECs. When a packet is received with a label corresponding to red-FEC-A (for example), an MRT transit router will determine the next-hop for the MRT-Red forwarding topology for that FEC, swap the incoming label with the outgoing label corresponding to red-FEC-A learned from the MRT-Red next-hop router, and forward the packet.

This forwarding mechanism has the useful property that the FEC associated with the packet is maintained in the labels at each hop along the MRT. We will take advantage of this property when specifying how to carry LDP traffic on MRT paths using multi-topology LDP labels.

This approach is very simple for hardware to support. However, it reduces the label space for other uses, and it increases the memory needed to store the labels and the communication required by LDP to distribute FEC-label bindings. In general, this approach will also increase the time needed to install the FRR entries in the Forwarding Information Base (FIB) and hence the time needed before the next failure can be protected.

This forwarding option uses the LDP signaling extensions described in [RFC7307]. The MRT-specific LDP extensions required to support this option will be described elsewhere.

6.1.1.2. Topology and FEC encoded using a two label stack (Option 1B)

With this forwarding mechanism, a two label stack is used to encode the topology and the FEC of the packet. The top label (topology-id label) identifies the MRT forwarding topology, while the second label (FEC label) identifies the FEC. The top label would be a new FEC type with two values corresponding to MRT Red and Blue topologies.

When an MRT transit router receives a packet with a topology-id label, the router pops the top label and uses that it to guide the next-hop selection in combination with the next label in the stack (the FEC label). The router then swaps the FEC label, using the FEC-label bindings learned through normal LDP mechanisms. The router then pushes the topology-id label for the next-hop.

As with Option 1A, this forwarding mechanism also has the useful property that the FEC associated with the packet is maintained in the labels at each hop along the MRT.

This forwarding mechanism has minimal usage of additional labels, memory and LDP communication. It does increase the size of packets and the complexity of the required label operations and look-ups.

This forwarding option is consistent with context-specific label spaces, as described in [RFC5331]. However, the precise LDP behavior required to support this option for MRT has not been specified.

6.1.1.3. Compatibility of MRT LDP Label Options 1A and 1B

MRT transit forwarding based on MRT LDP Label options 1A and 1B can coexist in the same network, with a packet being forwarded along a single MRT path using the single label of option 1A for some hops and the two label stack of option 1B for other hops. However, to simplify the process of MRT Island formation we require that all routers in the MRT Island support at least one common forwarding mechanism. As an example, the Default MRT Profile requires support for the MRT LDP Label Option 1A forwarding mechanism. This ensures that the routers in an MRT island supporting the Default MRT Profile will be able to establish MRT forwarding paths based on MRT LDP Label Option 1A. However, an implementation supporting Option 1A may also support Option 1B. If the scaling or performance characteristics for the two options differ in this implementation, then it may be desirable for a pair of adjacent routers to use Option 1B labels instead of the Option 1A labels. If those routers successfully negotiate the use of Option 1B labels, they are free to use them. This can occur without any of the other routers in the MRT Island being made aware of it.

Note that this document only defines the Default MRT Profile which requires support for the MRT LDP Label Option 1A forwarding mechanism.

6.1.1.4. Required support for MRT LDP Label options

If a router supports a profile that includes the MRT LDP Label Option 1A for the MRT transit forwarding mechanism, then it **MUST** support option 1A, which encodes topology-scoped FECs using a single label. The router **MAY** also support option 1B.

If a router supports a profile that includes the MRT LDP Label Option 1B for the MRT transit forwarding mechanism, then it **MUST** support option 1B, which encodes the topology and FEC using a two label stack. The router **MAY** also support option 1A.

6.1.2. MRT IP tunnels (Options 2A and 2B)

IP tunneling can also be used as an MRT transit forwarding mechanism. Each router supporting this MRT transit forwarding mechanism announces two additional loopback addresses and their associated MRT color. Those addresses are used as destination addresses for MRT-blue and MRT-red IP tunnels respectively. The special loopback

addresses allow the transit nodes to identify the traffic as being forwarded along either the MRT-blue or MRT-red topology to reach the tunnel destination. For example, an MRT ingress router can cause a packet to be tunneled along the MRT-red path to router X by encapsulating the packet using the MRT-red loopback address advertised by router X. Upon receiving the packet, router X would remove the encapsulation header and forward the packet based on the original destination address.

Either IPv4 (option 2A) or IPv6 (option 2B) can be used as the tunneling mechanism.

Note that the two forwarding mechanisms using LDP Label options do not require additional loopbacks per router, as is required by the IP tunneling mechanism. This is because LDP labels are used on a hop-by-hop basis to identify MRT-blue and MRT-red forwarding topologies.

6.2. Forwarding LDP Unicast Traffic over MRT Paths

In the previous section, we examined several options for providing MRT transit forwarding functionality, which is independent of the type of traffic being carried. We now look at the MRT ingress functionality, which will depend on the type of traffic being carried (IP or LDP). We start by considering LDP traffic.

We also simplify the initial discussion by assuming that the network consists of a single IGP area, and that all routers in the network participate in MRT. Other deployment scenarios that require MRT egress functionality are considered later in this document.

In principle, it is possible to carry LDP traffic in MRT IP tunnels. However, for LDP traffic, it is desirable to avoid tunneling. Tunneling LDP traffic to a remote node requires knowledge of remote FEC-label bindings so that the LDP traffic can continue to be forwarded properly when it leaves the tunnel. This requires targeted LDP sessions which can add management complexity. As described below, the two MRT forwarding mechanisms that use LDP labels do not require targeted LDP sessions.

6.2.1. Forwarding LDP traffic using MRT LDP Label Option 1A

The MRT LDP Label option 1A forwarding mechanism uses topology-scoped FECs encoded using a single label as described in section Section 6.1.1.1. When a PLR receives an LDP packet that needs to be forwarded on the Red MRT (for example), it does a label swap operation, replacing the usual LDP label for the FEC with the Red MRT label for that FEC received from the next-hop router in the Red MRT computed by the PLR. When the next-hop router in the Red MRT

receives the packet with the Red MRT label for the FEC, the MRT transit forwarding functionality continues as described in Section 6.1.1.1. In this way the original FEC associated with the packet is maintained at each hop along the MRT.

6.2.2. Forwarding LDP traffic using MRT LDP Label Option 1B

The MRT LDP Label option 1B forwarding mechanism encodes the topology and the FEC using a two label stack as described in Section 6.1.1.2. When a PLR receives an LDP packet that needs to be forwarded on the Red MRT, it first does a normal LDP label swap operation, replacing the incoming normal LDP label associated with a given FEC with the outgoing normal LDP label for that FEC learned from the next-hop on the Red MRT. In addition, the PLR pushes the topology-identification label associated with the Red MRT, and forward the packet to the appropriate next-hop on the Red MRT. When the next-hop router in the Red MRT receives the packet with the Red MRT label for the FEC, the MRT transit forwarding functionality continues as described in Section 6.1.1.2. As with option 1A, the original FEC associated with the packet is maintained at each hop along the MRT.

6.2.3. Other considerations for forwarding LDP traffic using MRT LDP Labels

Note that forwarding LDP traffic using MRT LDP Labels can be done without the use of targeted LDP sessions when an MRT path to the destination FEC is used. The alternates selected in [I-D.ietf-rtgwg-mrt-frr-algorithm] use the MRT path to the destination FEC, so targeted LDP sessions are not needed. If instead one found it desirable to have the PLR use an MRT to reach the primary next-next-hop for the FEC, and then continue forwarding the LDP packet along the shortest path tree from the primary next-next-hop, this would require tunneling to the primary next-next-hop and a targeted LDP session for the PLR to learn the FEC-label binding for primary next-next-hop to correctly forward the packet.

6.2.4. Required support for LDP traffic

For greatest hardware compatibility, routers implementing MRT fast-reroute of LDP traffic MUST support Option 1A of encoding the MT-ID in the labels (See Section 9).

6.3. Forwarding IP Unicast Traffic over MRT Paths

For IPv4 traffic, there is no currently practical alternative except tunneling to gain the bits needed to indicate the MRT-Blue or MRT-Red forwarding topology. For IPv6 traffic, in principle one could define bits in the IPv6 options header to indicate the MRT-Blue or MRT-Red

forwarding topology. However, in this document, we have chosen not to define a solution that would work for IPv6 traffic but not for IPv4 traffic.

The choice of tunnel egress is flexible since any router closer to the destination than the next-hop can work. This architecture assumes that the original destination in the area is selected (see Section 11 for handling of multi-homed prefixes); another possible choice is the next-next-hop towards the destination. As discussed in the previous section, for LDP traffic, using the MRT to the original destination simplifies MRT-FRR by avoiding the need for targeted LDP sessions to the next-next-hop. For IP, that consideration doesn't apply.

Some situations require tunneling IP traffic along an MRT to a tunnel endpoint that is not the destination of the IP traffic. These situations will be discussed in detail later. We note here that an IP packet with a destination in a different IGP area/level from the PLR should be tunneled on the MRT to the Area Border Router (ABR) or Level Border Router (LBR) on the shortest path to the destination. For a destination outside of the PLR's MRT Island, the packet should be tunneled on the MRT to a non-proxy-node immediately before the named proxy-node on that particular color MRT.

6.3.1. Tunneling IP traffic using MRT LDP Labels

An IP packet can be tunneled along an MRT path by pushing the appropriate MRT LDP label(s). Tunneling using LDP labels, as opposed to IP headers, has the advantage that more installed routers can do line-rate encapsulation and decapsulation using LDP than using IP. Also, no additional IP addresses would need to be allocated or signaled.

6.3.1.1. Tunneling IP traffic using MRT LDP Label Option 1A

The MRT LDP Label option 1A forwarding mechanism uses topology-scoped FECs encoded using a single label as described in section Section 6.1.1.1. When a PLR receives an IP packet that needs to be forwarded on the Red MRT to a particular tunnel endpoint, it does a label push operation. The label pushed is the Red MRT label for a FEC originated by the tunnel endpoint, learned from the next-hop on the Red MRT.

6.3.1.2. Tunneling IP traffic using MRT LDP Label Option 1B

The MRT LDP Label option 1B forwarding mechanism encodes the topology and the FEC using a two label stack as described in Section 6.1.1.2. When a PLR receives an IP packet that needs to be forwarded on the

Red MRT to a particular tunnel endpoint, the PLR pushes two labels on the IP packet. The first (inner) label is the normal LDP label learned from the next-hop on the Red MRT, associated with a FEC originated by the tunnel endpoint. The second (outer) label is the topology-identification label associated with the Red MRT.

For completeness, we note here a potential variation that uses a single label as opposed to two labels. In order to tunnel an IP packet over an MRT to the destination of the IP packet (as opposed to an arbitrary tunnel endpoint), then we could just push a topology-identification label directly onto the packet. An MRT transit router would need to pop the topology-id label, do an IP route lookup in the context of that topology-id, and push the topology-id label.

6.3.2. Tunneling IP traffic using MRT IP Tunnels

In order to tunnel over the MRT to a particular tunnel endpoint, the PLR encapsulates the original IP packet with an additional IP header using the MRT-Blue or MRT-Red loopback address of the tunnel endpoint.

6.3.3. Required support for IP traffic

For greatest hardware compatibility and ease in removing the MRT-topology marking at area/level boundaries, routers that support MPLS and implement IP MRT fast-reroute MUST support tunneling of IP traffic using MRT LDP Label Option 1A (topology-scoped FEC encoded using a single label).

7. MRT Island Formation

The purpose of communicating support for MRT is to indicate that the MRT-Blue and MRT-Red forwarding topologies are created for transit traffic. The MRT architecture allows for different, potentially incompatible options. In order to create consistent MRT forwarding topologies, the routers participating in a particular MRT Island need to use the same set of options. These options are grouped into MRT profiles. In addition, the routers in an MRT Island all need to use the same set of nodes and links within the Island when computing the MRT forwarding topologies. This section describes the information used by a router to determine the nodes and links to include in a particular MRT Island. Some information already exists in the IGP and can be used by MRT in Island formation, subject to the interpretation defined here.

Other information needs to be communicated between routers for which there do not currently exist protocol extensions. This new information needs to be shared among all routers in an IGP area, so

defining extensions to existing IGPs to carry this information makes sense. These new protocol extensions will be defined elsewhere.

Deployment scenarios using multi-topology OSPF or IS-IS, or running both IS-IS and OSPF on the same routers is out of scope for this specification. As with LFA, it is expected that OSPF Virtual Links will not be supported.

At a high level, an MRT Island is defined as the set of routers supporting the same MRT profile, in the same IGP area/level and the bi-directional links interconnecting those routers. More detailed descriptions of these criteria are given below.

7.1. IGP Area or Level

All links in an MRT Island are bidirectional and belong to the same IGP area or level. For IS-IS, a link belonging to both level 1 and level 2 would qualify to be in multiple MRT Islands. A given ABR or LBR can belong to multiple MRT Islands, corresponding to the areas or levels in which it participates. Inter-area forwarding behavior is discussed in Section 10.

7.2. Support for a specific MRT profile

All routers in an MRT Island support the same MRT profile. A router advertises support for a given MRT profile using an 8-bit MRT Profile ID value. The registry for the MRT Profile ID is defined in this document. The protocol extensions for advertising the MRT Profile ID value will be defined elsewhere. A given router can support multiple MRT profiles and participate in multiple MRT Islands. The options that make up an MRT profile, as well as the default MRT profile, are defined in Section 8.

The process of MRT Island formation takes place independently for each MRT profile advertised by a given router. For example, consider a network with 40 connected routers in the same area advertising support for MRT Profile A and MRT Profile B. Two distinct MRT Islands will be formed corresponding to Profile A and Profile B, with each island containing all 40 routers. A complete set of maximally redundant trees will be computed for each island following the rules defined for each profile. If we add a third MRT Profile to this example, with Profile C being advertised by a connected subset of 30 routers, there will be a third MRT Island formed corresponding to those 30 routers, and a third set of maximally redundant trees will be computed. In this example, 40 routers would compute and install two sets of MRT transit forwarding entries corresponding to Profiles A and B, while 30 routers would compute and install three sets of MRT transit forwarding entries corresponding to Profiles A, B, and C.

7.3. Excluding additional routers and interfaces from the MRT Island

MRT takes into account existing IGP mechanisms for discouraging traffic from using particular links and routers, and it introduces an MRT-specific exclusion mechanism for links.

7.3.1. Existing IGP exclusion mechanisms

Mechanisms for discouraging traffic from using particular links already exist in IS-IS and OSPF. In IS-IS, an interface configured with a metric of $2^{24}-2$ (0xFFFFFE) will only be used as a last resort. (An interface configured with a metric of $2^{24}-1$ (0xFFFFF) will not be advertised into the topology.) In OSPF, an interface configured with a metric of $2^{16}-1$ (0xFFFF) will only be used as a last resort. These metrics can be configured manually to enforce administrative policy, or they can be set in an automated manner as with LDP IGP synchronization [RFC5443].

Mechanisms also already exist in IS-IS and OSPF to discourage or prevent transit traffic from using a particular router. In IS-IS, the overload bit is prevents transit traffic from using a router.

For OSPFv2 and OSPFv3, [RFC6987] specifies setting all outgoing interface metrics to 0xFFFF to discourage transit traffic from using a router. ([RFC6987] defines the metric value 0xFFFF as MaxLinkMetric, a fixed architectural value for OSPF.) For OSPFv3, [RFC5340] specifies that a router be excluded from the intra-area shortest path tree computation if the V6-bit or R-bit of the LSA options is not set in the Router LSA.

The following rules for MRT Island formation ensure that MRT FRR protection traffic does not use a link or router that is discouraged or prevented from carrying traffic by existing IGP mechanisms.

1. A bidirectional link MUST be excluded from an MRT Island if either the forward or reverse cost on the link is 0xFFFFFE (for IS-IS) or 0xFFFF for OSPF.
2. A router MUST be excluded from an MRT Island if it is advertised with the overload bit set (for IS-IS), or it is advertised with metric values of 0xFFFF on all of its outgoing interfaces (for OSPFv2 and OSPFv3).
3. A router MUST be excluded from an MRT Island if it is advertised with either the V6-bit or R-bit of the LSA options not set in the Router LSA.

7.3.2. MRT-specific exclusion mechanism

This architecture also defines a means of excluding an otherwise usable link from MRT Islands. The protocol extensions for advertising that a link is MRT-Ineligible will be defined elsewhere. A link with either interface advertised as MRT-Ineligible MUST be excluded from an MRT Island. Note that an interface advertised as MRT-Ineligible by a router is ineligible with respect to all profiles advertised by that router.

7.4. Connectivity

All of the routers in an MRT Island MUST be connected by bidirectional links with other routers in the MRT Island. Disconnected MRT Islands will operate independently of one another.

7.5. Algorithm for MRT Island Identification

An algorithm that allows a computing router to identify the routers and links in the local MRT Island satisfying the above rules is given in section 5.2 of [I-D.ietf-rtgwg-mrt-frr-algorithm].

8. MRT Profile

An MRT Profile is a set of values and options related to MRT behavior. The complete set of options is designated by the corresponding 8-bit Profile ID value.

This document specifies the values and options that correspond to the Default MRT Profile (Profile ID = 0). Future documents may define other MRT Profiles by specifying the MRT Profile Options below.

8.1. MRT Profile Options

Below is a description of the values and options that define an MRT Profile.

MRT Algorithm: This identifies the particular algorithm for computing maximally redundant trees used by the router for this profile.

MRT-Red MT-ID: This specifies the MPLS MT-ID to be associated with the MRT-Red forwarding topology. It is allocated from the MPLS Multi-Topology Identifiers Registry.

MRT-Blue MT-ID: This specifies the MPLS MT-ID to be associated with the MRT-Blue forwarding topology. It is allocated from the MPLS Multi-Topology Identifiers Registry.

GADAG Root Selection Policy: This specifies the manner in which the GADAG root is selected. All routers in the MRT island need to use the same GADAG root in the calculations used to construct the MRTs. A valid GADAG Root Selection Policy **MUST** be such that each router in the MRT island chooses the same GADAG root based on information available to all routers in the MRT island. GADAG Root Selection Priority values, advertised as router-specific MRT parameters, **MAY** be used in a GADAG Root Selection Policy.

MRT Forwarding Mechanism: This specifies which forwarding mechanism the router uses to carry transit traffic along MRT paths. A router which supports a specific MRT forwarding mechanism must program appropriate next-hops into the forwarding plane. The current options are MRT LDP Label Option 1A, MRT LDP Label Option 1B, IPv4 Tunneling, IPv6 Tunneling, and None. If IPv4 is supported, then both MRT-Red and MRT-Blue IPv4 Loopback Addresses **SHOULD** be specified. If IPv6 is supported, both MRT-Red and MRT-Blue IPv6 Loopback Addresses **SHOULD** be specified.

Recalculation: Recalculation specifies the process and timing by which new MRTs are computed after the topology has been modified.

Area/Level Border Behavior: This specifies how traffic traveling on the MRT-Blue or MRT-Red in one area should be treated when it passes into another area.

Other Profile-Specific Behavior: Depending upon the use-case for the profile, there may be additional profile-specific behavior.

When a new MRT Profile is defined, new and unique values should be allocated from the MPLS Multi-Topology Identifiers Registry, corresponding to the MRT-Red and MRT-Blue MT-ID values for the new MRT Profile .

If a router advertises support for multiple MRT profiles, then it **MUST** create the transit forwarding topologies for each of those, unless the profile specifies the None option for MRT Forwarding Mechanism.

The ability of MRT-FRR to support transit forwarding entries for multiple profiles can be used to facilitate a smooth transition from an existing deployed MRT Profile to a new MRT Profile. The new profile can be activated in parallel with the existing profile, installing the transit forwarding entries for the new profile without affecting the transit forwarding entries for the existing profile. Once the new transit forwarding state has been verified, the router can be configured to use the alternates computed by the new profile in the event of a failure.

8.2. Router-specific MRT parameters

For some profiles, additional router-specific MRT parameters may need to be advertised. While the set of options indicated by the MRT Profile ID must be identical for all routers in an MRT Island, these router-specific MRT parameters may differ between routers in the same MRT island. Several such parameters are described below.

GADAG Root Selection Priority: A GADAG Root Selection Policy MAY rely on the GADAG Root Selection Priority values advertised by each router in the MRT island. A GADAG Root Selection Policy may use the GADAG Root Selection Priority to allow network operators to configure a parameter to ensure that the GADAG root is selected from a particular subset of routers. An example of this use of the GADAG Root Selection Priority value by the GADAG Root Selection Policy is given in the Default MRT profile below.

MRT-Red Loopback Address: This provides the router's loopback address to reach the router via the MRT-Red forwarding topology. It can be specified for either IPv4 or IPv6. Note that this parameter is not needed to support the Default MRT profile.

MRT-Blue Loopback Address: This provides the router's loopback address to reach the router via the MRT-Blue forwarding topology. It can be specified for either IPv4 and IPv6. Note that this parameter is not needed to support the Default MRT profile.

Protocol extensions for advertising a router's GADAG Root Selection Priority value will be defined in other documents. Protocol extensions for the advertising a router's MRT-Red and MRT-Blue Loopback Addresses will be defined elsewhere.

8.3. Default MRT profile

The following set of options defines the default MRT Profile. The default MRT profile is indicated by the MRT Profile ID value of 0.

MRT Algorithm: MRT Lowpoint algorithm defined in [I-D.ietf-rtgwg-mrt-frr-algorithm].

MRT-Red MPLS MT-ID: This value will be allocated from the MPLS Multi-Topology Identifiers Registry. The IANA request for this allocation will be in another document.

MRT-Blue MPLS MT-ID: This value will be allocated from the MPLS Multi-Topology Identifiers Registry. The IANA request for this allocation will be in another document.

GADAG Root Selection Policy: Among the routers in the MRT Island with the lowest numerical value advertised for GADAG Root Selection Priority, an implementation MUST pick the router with the highest Router ID to be the GADAG root. Note that a lower numerical value for GADAG Root Selection Priority indicates a higher preference for selection.

Forwarding Mechanisms: MRT LDP Label Option 1A

Recalculation: Recalculation of MRTs SHOULD occur as described in Section 12.2. This allows the MRT forwarding topologies to support IP/LDP fast-reroute traffic.

Area/Level Border Behavior: As described in Section 10, ABRs/LBRs SHOULD ensure that traffic leaving the area also exits the MRT-Red or MRT-Blue forwarding topology.

9. LDP signaling extensions and considerations

The protocol extensions for LDP will be defined in another document. A router must indicate that it has the ability to support MRT; having this explicit allows the use of MRT-specific processing, such as special handling of FECs sent with the Rainbow MRT MT-ID.

A FEC sent with the Rainbow MRT MT-ID indicates that the FEC applies to all the MRT-Blue and MRT-Red MT-IDs in supported MRT profiles. The FEC-label bindings for the default shortest-path based MT-ID 0 MUST still be sent (even though it could be inferred from the Rainbow FEC-label bindings) to ensure continuous operation of normal LDP forwarding. The Rainbow MRT MT-ID is defined to provide an easy way to handle the special signaling that is needed at ABRs or LBRs. It avoids the problem of needing to signal different MPLS labels to different LDP neighbors for the same FEC. Because the Rainbow MRT MT-ID is used only by ABRs/LBRs or an LDP egress router, it is not MRT profile specific.

The value of the Rainbow MRT MPLS MT-ID will be allocated from the MPLS Multi-Topology Identifiers Registry. The IANA request for this allocation will be in another document.

10. Inter-area Forwarding Behavior

An ABR/LBR has two forwarding roles. First, it forwards traffic within areas. Second, it forwards traffic from one area into another. These same two roles apply for MRT transit traffic. Traffic on MRT-Red or MRT-Blue destined inside the area needs to stay on MRT-Red or MRT-Blue in that area. However, it is desirable for

traffic leaving the area to also exit MRT-Red or MRT-Blue and return to shortest path forwarding.

For unicast MRT-FRR, the need to stay on an MRT forwarding topology terminates at the ABR/LBR whose best route is via a different area/level. It is highly desirable to go back to the default forwarding topology when leaving an area/level. There are three basic reasons for this. First, the default topology uses shortest paths; the packet will thus take the shortest possible route to the destination. Second, this allows a single router failure that manifests itself in multiple areas (as would be the case with an ABR/LBR failure) to be separately identified and repaired around. Third, the packet can be fast-rerouted again, if necessary, due to a second distinct failure in a different area.

In OSPF, an ABR that receives a packet on MRT-Red or MRT-Blue towards destination Z should continue to forward the packet along MRT-Red or MRT-Blue only if the best route to Z is in the same OSPF area as the interface that the packet was received on. Otherwise, the packet should be removed from MRT-Red or MRT-Blue and forwarded on the shortest-path default forwarding topology.

The above description applies to OSPF. The same essential behavior also applies to IS-IS if one substitutes IS-IS level for OSPF area. However, the analogy with OSPF is not exact. An interface in OSPF can only be in one area, whereas an interface in IS-IS can be in both Level-1 and Level-2. Therefore, to avoid confusion and address this difference, we explicitly describe the behavior for IS-IS in Appendix A. In the following sections only the OSPF terminology is used.

10.1. ABR Forwarding Behavior with MRT LDP Label Option 1A

For LDP forwarding where a single label specifies (MT-ID, FEC), the ABR is responsible for advertising the proper label to each neighbor. Assume that an ABR has allocated three labels for a particular destination; those labels are *L_primary*, *L_blue*, and *L_red*. To those routers in the same area as the best route to the destination, the ABR advertises the following FEC-label bindings: *L_primary* for the default topology, *L_blue* for the MRT-Blue MT-ID and *L_red* for the MRT-Red MT-ID, as expected. However, to routers in other areas, the ABR advertises the following FEC-label bindings: *L_primary* for the default topology, and *L_primary* for the Rainbow MRT MT-ID. Associating *L_primary* with the Rainbow MRT MT-ID causes the receiving routers to use *L_primary* for the MRT-Blue MT-ID and for the MRT-Red MT-ID.

The ABR installs all next-hops for the best area: primary next-hops for `L_primary`, MRT-Blue next-hops for `L_blue`, and MRT-Red next-hops for `L_red`. Because the ABR advertised (Rainbow MRT MT-ID, FEC) with `L_primary` to neighbors not in the best area, packets from those neighbors will arrive at the ABR with a label `L_primary` and will be forwarded into the best area along the default topology. By controlling what labels are advertised, the ABR can thus enforce that packets exiting the area do so on the shortest-path default topology.

10.1.1. Motivation for Creating the Rainbow-FEC

The desired forwarding behavior could be achieved in the above example without using the Rainbow-FEC. This could be done by having the ABR advertise the following FEC-label bindings to neighbors not in the best area: `L1_primary` for the default topology, `L1_primary` for the MRT-Blue MT-ID, and `L1_primary` for the MRT-Red MT-ID. Doing this would require machinery to spoof the labels used in FEC-label binding advertisements on a per-neighbor basis. Such label-spoofing machinery does not currently exist in most LDP implementations and doesn't have other obvious uses.

Many existing LDP implementations do however have the ability to filter FEC-label binding advertisements on a per-neighbor basis. The Rainbow-FEC allows us to re-use the existing per-neighbor FEC filtering machinery to achieve the desired result. By introducing the Rainbow FEC, we can use per-neighbor FEC-filtering machinery to advertise the FEC-label binding for the Rainbow-FEC (and filter those for MRT-Blue and MRT-Red) to non-best-area neighbors of the ABR.

An ABR may choose to either advertise the Rainbow-FEC or advertise separate MRT-Blue and MRT-Red advertisements. This is a local choice. A router that supports the MRT LDP Label Option 1A Forwarding Mechanism MUST be able to receive and correctly interpret the Rainbow-FEC.

10.2. ABR Forwarding Behavior with IP Tunneling (option 2)

If IP tunneling is used, then the ABR behavior is dependent upon the outermost IP address. If the outermost IP address is an MRT loopback address of the ABR, then the packet is decapsulated and forwarded based upon the inner IP address, which should go on the default SPT topology. If the outermost IP address is not an MRT loopback address of the ABR, then the packet is simply forwarded along the associated forwarding topology. A PLR sending traffic to a destination outside its local area/level will pick the MRT and use the associated MRT loopback address of the selected ABR advertising the lowest cost to the external destination.

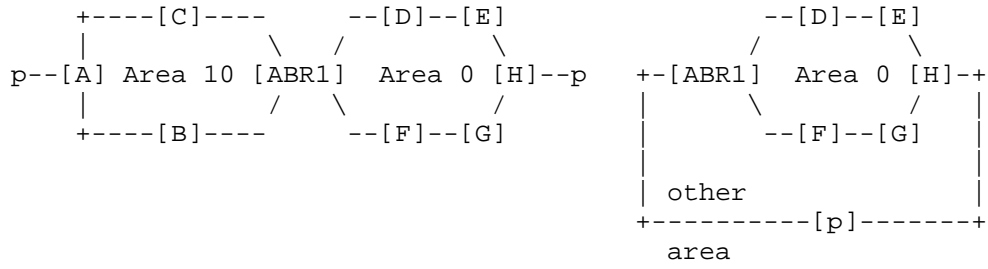
Thus, for these two MRT Forwarding Mechanisms (MRT LDP Label option 1A and IP tunneling option 2), there is no need for additional computation or per-area forwarding state.

10.3. ABR Forwarding Behavior with MRT LDP Label option 1B

The other MRT forwarding mechanism described in Section 6 uses two labels, a topology-id label, and a FEC-label. This mechanism would require that any router whose MRT-Red or MRT-Blue next-hop is an ABR would need to determine whether the ABR would forward the packet out of the area/level. If so, then that router should pop off the topology-identification label before forwarding the packet to the ABR.

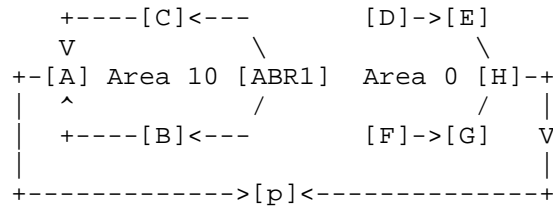
For example, in Figure 3, if node H fails, node E has to put traffic towards prefix p onto MRT-Red. But since node D knows that ABR1 will use a best route from another area, it is safe for D to pop the Topology-Identification Label and just forward the packet to ABR1 along the MRT-Red next-hop. ABR1 will use the shortest path in Area 10.

In all cases for IS-IS and most cases for OSPF, the penultimate router can determine what decision the adjacent ABR will make. The one case where it can't be determined is when two ASBRs are in different non-backbone areas attached to the same ABR, then the ASBR's Area ID may be needed for tie-breaking (prefer the route with the largest OPSF area ID) and the Area ID isn't announced as part of the ASBR link-state advertisement (LSA). In this one case, suboptimal forwarding along the MRT in the other area would happen. If that becomes a realistic deployment scenario, protocol extensions could be developed to address this issue.

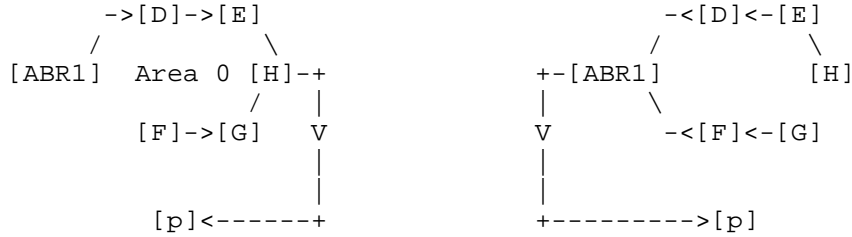


(a) Example topology

(b) Proxy node view in Area 0 nodes



(c) rSPT towards destination p



(d) Blue MRT in Area 0

(e) Red MRT in Area 0

Figure 3: ABR Forwarding Behavior and MRTs

11. Prefixes Multiply Attached to the MRT Island

How a computing router S determines its local MRT Island for each supported MRT profile is already discussed in Section 7.

There are two types of prefixes or FECs that may be multiply attached to an MRT Island. The first type are multi-homed prefixes that usually connect at a domain or protocol boundary. The second type represent routers that do not support the profile for the MRT Island.

The key difference is whether the traffic, once out of the MRT Island, might re-enter the MRT Island if a loop-free exit point is not selected.

FRR using LFA has the useful property that it is able to protect multi-homed prefixes against ABR failure. For instance, if a prefix from the backbone is available via both ABR A and ABR B, if A fails, then the traffic should be redirected to B. This can be accomplished with MRT FRR as well.

If ASBR protection is desired, this has additional complexities if the ASBRs are in different areas. Similarly, protecting labeled BGP traffic in the event of an ASBR failure has additional complexities due to the per-ASBR label spaces involved.

As discussed in [RFC5286], a multi-homed prefix could be:

- o An out-of-area prefix announced by more than one ABR,
- o An AS-External route announced by 2 or more ASBRs,
- o A prefix with iBGP multipath to different ASBRs,
- o etc.

See Appendix B for a discussion of a general issue with multi-homed prefixes connected in two different areas.

There are also two different approaches to protection. The first is tunnel endpoint selection where the PLR picks a router to tunnel to where that router is loop-free with respect to the failure-point. Conceptually, the set of candidate routers to provide LFAs expands to all routers that can be reached via an MRT alternate, attached to the prefix.

The second is to use a proxy-node, that can be named via MPLS label or IP address, and pick the appropriate label or IP address to reach it on either MRT-Blue or MRT-Red as appropriate to avoid the failure point. A proxy-node can represent a destination prefix that can be attached to the MRT Island via at least two routers. It is termed a named proxy-node if there is a way that traffic can be encapsulated to reach specifically that proxy-node; this could be because there is an LDP FEC for the associated prefix or because MRT-Red and MRT-Blue IP addresses are advertised (in an as-yet undefined fashion) for that proxy-node. Traffic to a named proxy-node may take a different path than traffic to the attaching router; traffic is also explicitly forwarded from the attaching router along a predetermined interface towards the relevant prefixes.

For IP traffic, multi-homed prefixes can use tunnel endpoint selection. For IP traffic that is destined to a router outside the MRT Island, if that router is the egress for a FEC advertised into the MRT Island, then the named proxy-node approach can be used.

For LDP traffic, there is always a FEC advertised into the MRT Island. The named proxy-node approach should be used, unless the computing router S knows the label for the FEC at the selected tunnel endpoint.

If a FEC is advertised from outside the MRT Island into the MRT Island and the forwarding mechanism specified in the profile includes LDP, then the routers learning that FEC MUST also advertise labels for (MRT-Red, FEC) and (MRT-Blue, FEC) to neighbors inside the MRT Island. Any router receiving a FEC corresponding to a router outside the MRT Island or to a multi-homed prefix MUST compute and install the transit MRT-Blue and MRT-Red next-hops for that FEC. The FEC-label bindings for the topology-scoped FECs ((MT-ID 0, FEC), (MRT-Red, FEC), and (MRT-Blue, FEC)) MUST also be provided via LDP to neighbors inside the MRT Island.

11.1. Protecting Multi-Homed Prefixes using Tunnel Endpoint Selection

Tunnel endpoint selection is a local matter for a router in the MRT Island since it pertains to selecting and using an alternate and does not affect the transit MRT-Red and MRT-Blue forwarding topologies.

Let the computing router be S and the next-hop F be the node whose failure is to be avoided. Let the destination be prefix p. Have A be the router to which the prefix p is attached for S's shortest path to p.

The candidates for tunnel endpoint selection are those to which the destination prefix is attached in the area/level. For a particular candidate B, it is necessary to determine if B is loop-free to reach p with respect to S and F for node-protection or at least with respect to S and the link (S, F) for link-protection. If B will always prefer to send traffic to p via a different area/level, then this is definitional. Otherwise, distance-based computations are necessary and an SPF from B's perspective may be necessary. The following equations give the checks needed; the rationale is similar to that given in [RFC5286]. In the inequalities below, $D_{\text{opt}}(X,Y)$ means the shortest distance from node X to node Y, and $D_{\text{opt}}(X,p)$ means the shortest distance from node X to prefix p.

Loop-Free for S: $D_{\text{opt}}(B, p) < D_{\text{opt}}(B, S) + D_{\text{opt}}(S, p)$

Loop-Free for F: $D_{\text{opt}}(B, p) < D_{\text{opt}}(B, F) + D_{\text{opt}}(F, p)$

The latter is equivalent to the following, which avoids the need to compute the shortest path from F to p.

Loop-Free for F: $D_{\text{opt}}(B, p) < D_{\text{opt}}(B, F) + D_{\text{opt}}(S, p) - D_{\text{opt}}(S, F)$

Finally, the rules for Endpoint selection are given below. The basic idea is to repair to the prefix-advertising router selected for the shortest-path and only to select and tunnel to a different endpoint if necessary (e.g. $A=F$ or F is a cut-vertex or the link (S,F) is a cut-link).

1. Does S have a node-protecting alternate to A? If so, select that. Tunnel the packet to A along that alternate. For example, if LDP is the forwarding mechanism, then push the label (MRT-Red, A) or (MRT-Blue, A) onto the packet.
2. If not, then is there a router B that is loop-free to reach p while avoiding both F and S? If so, select B as the end-point. Determine the MRT alternate to reach B while avoiding F. Tunnel the packet to B along that alternate. For example, with LDP, push the label (MRT-Red, B) or (MRT-Blue, B) onto the packet.
3. If not, then does S have a link-protecting alternate to A? If so, select that.
4. If not, then is there a router B that is loop-free to reach p while avoiding S and the link from S to F? If so, select B as the endpoint and the MRT alternate for reaching B from S that avoid the link (S,F).

The tunnel endpoint selected will receive a packet destined to itself and, being the egress, will pop that MPLS label (or have signaled Implicit Null) and forward based on what is underneath. This suffices for IP traffic since the tunnel endpoint can use the IP header of the original packet to continue forwarding the packet. However, tunnelling of LDP traffic requires targeted LDP sessions for learning the FEC-label binding at the tunnel endpoint.

11.2. Protecting Multi-Homed Prefixes using Named Proxy-Nodes

Instead, the named proxy-node method works with LDP traffic without the need for targeted LDP sessions. It also has a clear advantage over tunnel endpoint selection, in that it is possible to explicitly forward from the MRT Island along an interface to a loop-free island neighbor when that interface may not be a primary next-hop.

A named proxy-node represents one or more destinations and, for LDP forwarding, has a FEC associated with it that is signalled into the MRT Island. Therefore, it is possible to explicitly label packets to go to (MRT-Red, FEC) or (MRT-Blue, FEC); at the border of the MRT Island, the label will swap to meaning (MT-ID 0, FEC). It would be possible to have named proxy-nodes for IP forwarding, but this would require extensions to signal two IP addresses to be associated with MRT-Red and MRT-Blue for the proxy-node. A named proxy-node can be uniquely represented by the two routers in the MRT Island to which it is connected. The extensions to signal such IP addresses will be defined elsewhere. The details of what label-bindings must be originated will be described in another document.

Computing the MRT next-hops to a named proxy-node and the MRT alternate for the computing router S to avoid a particular failure node F is straightforward. The details of the simple constant-time functions, `Select_Proxy_Node_NHs()` and `Select_Alternates_Proxy_Node()`, are given in [I-D.ietf-rtgwg-mrt-frr-algorithm]. A key point is that computing these MRT next-hops and alternates can be done as new named proxy-nodes are added or removed without requiring a new MRT computation or impacting other existing MRT paths. This maps very well to, for example, how OSPFv2 (see [RFC2328] Section 16.5) does incremental updates for new summary-LSAs.

The remaining question is how to attach the named proxy-node to the MRT Island; all the routers in the MRT Island MUST do this consistently. No more than 2 routers in the MRT Island can be selected; one should only be selected if there are no others that meet the necessary criteria. The named proxy-node is logically part of the area/level.

There are two sources for candidate routers in the MRT Island to connect to the named proxy-node. The first set are those routers in the MRT Island that are advertising the prefix; the named-proxy-cost assigned to each prefix-advertising router is the announced cost to the prefix. The second set are those routers in the MRT Island that are connected to routers not in the MRT Island but in the same area/level; such routers will be defined as Island Border Routers (IBRs). The routers connected to the IBRs that are not in the MRT Island and are in the same area/level as the MRT island are Island Neighbors (INs).

Since packets sent to the named proxy-node along MRT-Red or MRT-Blue may come from any router inside the MRT Island, it is necessary that whatever router to which an IBR forwards the packet be loop-free with respect to the whole MRT Island for the destination. Thus, an IBR is a candidate router only if it possesses at least one IN whose

shortest path to the prefix does not enter the MRT Island. A method for identifying loop-free Island Neighbors(LFINs) is given in [I-D.ietf-rtgwg-mrt-frr-algorithm]. The named-proxy-cost assigned to each (IBR, IN) pair is $\text{cost}(\text{IBR}, \text{IN}) + D_{\text{opt}}(\text{IN}, \text{prefix})$.

From the set of prefix-advertising routers and the set of IBRs with at least one LFIN, the two routers with the lowest named-proxy-cost are selected. Ties are broken based upon the lowest Router ID. For ease of discussion, the two selected routers will be referred to as proxy-node attachment routers.

A proxy-node attachment router has a special forwarding role. When a packet is received destined to (MRT-Red, prefix) or (MRT-Blue, prefix), if the proxy-node attachment router is an IBR, it MUST swap to the shortest path forwarding topology (e.g. swap to the label for (MT-ID 0, prefix) or remove the outer IP encapsulation) and forward the packet to the IN whose cost was used in the selection. If the proxy-node attachment router is not an IBR, then the packet MUST be removed from the MRT forwarding topology and sent along the interface(s) that caused the router to advertise the prefix; this interface might be out of the area/level/AS.

11.3. MRT Alternates for Destinations Outside the MRT Island

A natural concern with new functionality is how to have it be useful when it is not deployed across an entire IGP area. In the case of MRT FRR, where it provides alternates when appropriate LFAs aren't available, there are also deployment scenarios where it may make sense to only enable some routers in an area with MRT FRR. A simple example of such a scenario would be a ring of 6 or more routers that is connected via two routers to the rest of the area.

Destinations inside the local island can obviously use MRT alternates. Destinations outside the local island can be treated like a multi-homed prefix and either Endpoint Selection or Named Proxy-Nodes can be used. Named Proxy-Nodes MUST be supported when LDP forwarding is supported and a label-binding for the destination is sent to an IBR.

Naturally, there are more complicated options to improve coverage, such as connecting multiple MRT islands across tunnels, but the need for the additional complexity has not been justified.

12. Network Convergence and Preparing for the Next Failure

After a failure, MRT detours ensure that packets reach their intended destination while the IGP has not reconverged onto the new topology. As link-state updates reach the routers, the IGP process calculates

the new shortest paths. Two things need attention: micro-loop prevention and MRT re-calculation.

12.1. Micro-loop prevention and MRTs

A micro-loop is a transient packet forwarding loop among two or more routers that can occur during convergence of IGP forwarding state. [RFC5715] discusses several techniques for preventing micro-loops. This section discusses how MRT-FRR relates to two of the micro-loop prevention techniques discussed in [RFC5715], Nearside Tunneling and Farside Tunneling.

In Nearside Tunneling, a router (PLR) adjacent to a failure perform local repair and inform remote routers of the failure. The remote routers initially tunnel affected traffic to the nearest PLR, using tunnels which are unaffected by the failure. Once the forwarding state for normal shortest path routing has converged, the remote routers return the traffic to shortest path forwarding. MRT-FRR is relevant for Nearside Tunneling for the following reason. The process of tunneling traffic to the PLRs and waiting a sufficient amount of time for IGP forwarding state convergence with Nearside Tunneling means that traffic will generally be relying on the local repair at the PLR for longer than it would in the absence of Nearside Tunneling. Since MRT-FRR provides 100% coverage for single link and node failure, it may be an attractive option to provide the local repair paths when Nearside Tunneling is deployed.

MRT-FRR is also relevant for the Farside Tunneling micro-loop prevention technique. In Farside Tunneling, remote routers tunnel traffic affected by a failure to a node downstream of the failure with respect to traffic destination. This node can be viewed as being on the farside of the failure with respect to the node initiating the tunnel. Note that the discussion of Farside Tunneling in [RFC5715] focuses on the case where the farside node is immediately adjacent to a failed link or node. However, the farside node may be any node downstream of the failure with respect to traffic destination, including the destination itself. The tunneling mechanism used to reach the farside node must be unaffected by the failure. The alternative forwarding paths created by MRT-FRR have the potential to be used to forward traffic from the remote routers upstream of the failure all the way to the destination. In the event of failure, either the MRT-Red or MRT-Blue path from the remote upstream router to the destination is guaranteed to avoid a link failure or inferred node failure. The MRT forwarding paths are also guaranteed to not be subject to micro-loops because they are locked to the topology before the failure.

We note that the computations in [I-D.ietf-rtgwg-mrt-frr-algorithm] address the case of a PLR adjacent to a failure determining which choice of MRT-Red or MRT-Blue will avoid a failed link or node. More computation may be required for an arbitrary remote upstream router to determine whether to choose MRT-Red or MRT-Blue for a given destination and failure.

12.2. MRT Recalculation for the Default MRT Profile

This section describes how the MRT recalculation SHOULD be performed for the Default MRT Profile. This is intended to support FRR applications. Other approaches are possible, but they are not specified in this document.

When a failure event happens, traffic is put by the PLRs onto the MRT topologies. After that, each router recomputes its shortest path tree (SPT) and moves traffic over to that. Only after all the PLRs have switched to using their SPTs and traffic has drained from the MRT topologies should each router install the recomputed MRTs into the FIBs.

At each router, therefore, the sequence is as follows:

1. Receive failure notification
2. Recompute SPT.
3. Install the new SPT in the FIB.
4. If the network was stable before the failure occurred, wait a configured (or advertised) period for all routers to be using their SPTs and traffic to drain from the MRTs.
5. Recompute MRTs.
6. Install new MRTs in the FIB.

While the recomputed MRTs are not installed in the FIB, protection coverage is lowered. Therefore, it is important to recalculate the MRTs and install them quickly.

New protocol extensions for advertising the time needed to recompute shortest path routes and install them in the FIB will be defined elsewhere.

13. Implementation Status

[RFC Editor: please remove this section prior to publication.]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC6982]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC6982], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

Juniper Networks Implementation

- o Organization responsible for the implementation: Juniper Networks
- o Implementation name: MRT-FRR
- o Implementation description: MRT-FRR using OSPF as the IGP has been implemented and verified.
- o The implementation's level of maturity: prototype
- o Protocol coverage: This implementation of the MRT-FRR includes Island identification, GADAG root selection, MRT Lowpoint algorithm, augmentation of GADAG with additional links, and calculation of MRT transit next-hops alternate next-hops based on draft "draft-ietf-rtgwg-mrt-frr-algorithm-00". This implementation also includes the M-bit in OSPF based on "draft-atlas-ospf-mrt-01" as well as LDP MRT Capability based on "draft-atlas-mpls-ldp-mrt-00".
- o Licensing: proprietary

- o Implementation experience: Implementation was useful for verifying functionality and lack of gaps. It has also been useful for improving aspects of the algorithm.
- o Contact information: akatlas@juniper.net, shraddha@juniper.net, kishoret@juniper.net

Huawei Technology Implementation

- o Organization responsible for the implementation: Huawei Technology Co., Ltd.
- o Implementation name: MRT-FRR and IS-IS extensions for MRT.
- o Implementation description: The MRT-FRR using IS-IS extensions for MRT and LDP multi-topology have been implemented and verified.
- o The implementation's level of maturity: prototype
- o Protocol coverage: This implementation of the MRT algorithm includes Island identification, GADAG root selection, MRT Lowpoint algorithm, augmentation of GADAG with additional links, and calculation of MRT transit next-hops alternate next-hops based on draft "draft-enyedi-rtgwg-mrt-frr-algorithm-03". This implementation also includes IS-IS extension for MRT based on "draft-li-mrt-00".
- o Licensing: proprietary
- o Implementation experience: It is important produce a second implementation to verify the algorithm is implemented correctly without looping. It is important to verify the IS-IS extensions work for MRT-FRR.
- o Contact information: lizhenbin@huawei.com, eric.wu@huawei.com

14. Operational Considerations

The following aspects of MRT-FRR are useful to consider when deploying the technology in different operational environments and network topologies.

14.1. Verifying Forwarding on MRT Paths

The forwarding paths created by MRT-FRR are not used by normal (non-FRR) traffic. They are only used to carry FRR traffic for a short period of time after a failure has been detected. It is RECOMMENDED that an operator proactively monitor the MRT forwarding paths in

order to be certain that the paths will be able to carry FRR traffic when needed. Therefore, an implementation SHOULD provide an operator with the ability to test MRT paths with Operations, Administration, and Maintenance (OAM) traffic. For example, when MRT paths are realized using LDP labels distributed for topology-scoped FECs, an implementation can use the MPLS ping and traceroute as defined in [RFC4379] and extended in [RFC7307] for topology-scoped FECs.

14.2. Traffic Capacity on Backup Paths

During a fast-reroute event initiated by a PLR in response to a network failure, the flow of traffic in the network will generally not be identical to the flow of traffic after the IGP forwarding state has converged, taking the failure into account. Therefore, even if a network has been engineered to have enough capacity on the appropriate links to carry all traffic after the IGP has converged after the failure, the network may still not have enough capacity on the appropriate links to carry the flow of traffic during a fast-reroute event. This can result in more traffic loss during the fast-reroute event than might otherwise be expected.

Note that there are two somewhat distinct aspects to this phenomenon. The first is that the path from the PLR to the destination during the fast-reroute event may be different from the path after the IGP converges. In this case, any traffic for the destination that reaches the PLR during the fast-reroute event will follow a different path from the PLR to the destination than will be followed after IGP convergence.

The second aspect is that the amount of traffic arriving at the PLR for affected destinations during the fast-reroute event may be larger than the amount of traffic arriving at the PLR for affected destinations after IGP convergence. Immediately after a failure, any non-PLR routers that were sending traffic to the PLR before the failure will continue sending traffic to the PLR, and that traffic will be carried over backup paths from the PLR to the destinations. After IGP convergence, upstream non-PLR routers may direct some traffic away from the PLR.

In order to reduce or eliminate the potential for transient traffic loss due to inadequate capacity during fast-reroute events, an operator can model the amount of traffic taking different paths during a fast-reroute event. If it is determined that there is not enough capacity to support a given fast-reroute event, the operator can address the issue either by augmenting capacity on certain links or modifying the backup paths themselves.

The MRT Lowpoint algorithm produces a pair of diverse paths to each destination. These paths are generated by following the directed links on a common GADAG. The decision process for constructing the GADAG in the MRT Lowpoint algorithm takes into account individual IGP link metrics. At any given node, links are explored in order from lowest IGP metric to highest IGP metric. Additionally, the process for constructing the MRT-Red and Blue trees uses SPF traversals of the GADAG. Therefore, the IGP link metric values affect the computed backup paths. However, adjusting the IGP link metrics is not a generally applicable tool for modifying the MRT backup paths. Achieving a desired set of MRT backup paths by adjusting IGP metrics while at the same time maintaining the desired flow of traffic along the shortest paths is not possible in general.

MRT-FRR allows an operator to exclude a link from the MRT Island, and thus the GADAG, by advertising it as MRT-Ineligible. Such a link will not be used on the MRT forwarding path for any destination. Advertising links as MRT-Ineligible is the main tool provided by MRT-FRR for keeping backup traffic off of lower bandwidth links during fast-reroute events.

Note that all of the backup paths produced by the MRT Lowpoint algorithm are closely tied to the common GADAG computed as part of that algorithm. Therefore, it is generally not possible to modify a subset of paths without affecting other paths. This precludes more fine-grained modification of individual backup paths when using only paths computed by the MRT Lowpoint algorithm.

However, it may be desirable to allow an operator to use MRT-FRR alternates together with alternates provided by other FRR technologies. A policy-based alternate selection process can allow an operator to select the best alternate from those provided by MRT and other FRR technologies. As a concrete example, it may be desirable to implement a policy where a downstream LFA (if it exists for a given failure mode and destination) is preferred over a given MRT alternate. This combination gives the operator the ability to affect where traffic flows during a fast-reroute event, while still producing backup paths that use no additional labels for LDP traffic and will not loop under multiple failures. This and other choices of alternate selection policy can be evaluated in the context of their effect on fast-reroute traffic flow and available capacity, as well as other deployment considerations.

Note that future documents may define MRT profiles in addition to the default profile defined here. Different MRT profiles will generally produce alternate paths with different properties. An implementation may allow an operator to use different MRT profiles instead of or in addition to the default profile.

14.3. MRT IP Tunnel Loopback Address Management

As described in Section 6.1.2, if an implementation uses IP tunneling as the mechanism to realize MRT forwarding paths, each node must advertise an MRT-Red and an MRT-Blue loopback address. These IP addresses must be unique within the routing domain to the extent that they do not overlap with each other or with any other routing table entries. It is expected that operators will use existing tools and processes for managing infrastructure IP addresses to manage these additional MRT-related loopback addresses.

14.4. MRT-FRR in a Network with Degraded Connectivity

Ideally, routers in a service provider network using MRT-FRR will be initially deployed in a 2-connected topology, allowing MRT-FRR to find completely diverse paths to all destinations. However, a network can differ from an ideal 2-connected topology for many possible reasons, including network failures and planned maintenance events.

MRT-FRR is designed to continue to function properly when network connectivity is degraded. When a network contains cut-vertices or cut-links dividing the network into different 2-connected blocks, MRT-FRR will continue to provide completely diverse paths for destinations within the same block as the PLR. For a destination in a different block from the PLR, the redundant paths created by MRT-FRR will be link and node diverse within each block, and the paths will only share links and nodes that are cut-links or cut-vertices in the topology.

If a network becomes partitioned with one set of routers having no connectivity to another set of routers, MRT-FRR will function independently in each set of connected routers, providing redundant paths to destinations in same set of connected routers as a given PLR.

14.5. Partial Deployment of MRT-FRR in a Network

A network operator may choose to deploy MRT-FRR only on a subset of routers in an IGP area. MRT-FRR is designed to accommodate this partial deployment scenario. Only routers that advertise support for a given MRT profile will be included in a given MRT Island. For a PLR within the MRT Island, MRT-FRR will create redundant forwarding paths to all destinations with the MRT Island using maximally redundant trees all the way to those destinations. For destinations outside of the MRT Island, MRT-FRR creates paths to the destination which use forwarding state created by MRT-FRR within the MRT Island and shortest path forwarding state outside of the MRT Island. The

paths created by MRT-FRR to non-Island destinations are guaranteed to be diverse within the MRT Island (if topologically possible).

However, the part of the paths outside of the MRT Island may not be diverse.

15. Acknowledgements

The authors would like to thank Mike Shand for his valuable review and contributions.

The authors would like to thank Joel Halpern, Hannes Gredler, Ted Qian, Kishore Tiruveedhula, Shraddha Hegde, Santosh Esale, Nitin Bahadur, Harish Sitaraman, Raveendra Torvi, Anil Kumar SN, Bruno Decraene, Eric Wu, Janos Farkas, Rob Shakir, Stewart Bryant, and Alvaro Retana for their suggestions and review.

16. IANA Considerations

IANA is requested to create a registry entitled "MRT Profile Identifier Registry". The range is 0 to 255. The Default MRT Profile defined in this document has value 0. Values 1-200 are allocated by Standards Action. Values 201-220 are for Experimental Use. Values 221-254 are for Private Use. Value 255 is reserved for future registry extension. (The allocation and use policies are described in [RFC5226].)

The initial registry is shown below.

Value	Description	Reference
-----	-----	-----
0	Default MRT Profile	[This draft]
1-200	Unassigned	
201-220	Experimental Use	
221-254	Private Use	
255	Reserved (for future registry extension)	

The MRT Profile Identifier Registry is a new registry in the IANA Matrix. Following existing conventions, <http://www.iana.org/protocols> should display a new header entitled "Maximally Redundant Tree (MRT) Parameters". Under that header, there should be an entry for "MRT Profile Identifier Registry" with a link to the registry itself at <http://www.iana.org/assignments/mrt-parameters/mrt-parameters.xhtml#mrt-profile-registry>.

17. Security Considerations

In general, MRT forwarding paths do not follow shortest paths. The transit forwarding state corresponding to the MRT paths is created during normal operations (before a failure occurs). Therefore, a malicious packet with an appropriate header injected into the network from a compromised location would be forwarded to a destination along a non-shortest path. When this technology is deployed, a network security design should not rely on assumptions about potentially malicious traffic only following shortest paths.

It should be noted that the creation of non-shortest forwarding paths is not unique to MRT.

MRT-FRR requires that routers advertise information used in the formation of MRT backup paths. While this document does not specify the protocol extensions used to advertise this information, we discuss security considerations related to the information itself. Injecting false MRT-related information could be used to direct some MRT backup paths over compromised transmission links. Combined with the ability to generate network failures, this could be used to send traffic over compromised transmission links during a fast-reroute event. In order to prevent this potential exploit, a receiving router needs to be able to authenticate MRT-related information that claims to have been advertised by another router.

18. Contributors

Robert Kebler
Juniper Networks
10 Technology Park Drive
Westford, MA 01886
USA
Email: rkebler@juniper.net

Andras Csaszar
Ericsson
Konyves Kalman krt 11
Budapest 1097
Hungary
Email: Andras.Csaszar@ericsson.com

Jeff Tantsura
Ericsson
300 Holger Way
San Jose, CA 95134
USA
Email: jeff.tantsura@ericsson.com

Russ White
VCE
Email: russw@riw.us

19. References

19.1. Normative References

- [I-D.ietf-rtgwg-mrt-frr-algorithm]
Envedi, G., Csaszar, A., Atlas, A., Bowers, C., and A. Gopalan, "Algorithms for computing Maximally Redundant Trees for IP/LDP Fast- Reroute", draft-ietf-rtgwg-mrt-frr-algorithm-06 (work in progress), October 2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.

- [RFC7307] Zhao, Q., Raza, K., Zhou, C., Fang, L., Li, L., and D. King, "LDP Extensions for Multi-Topology", RFC 7307, DOI 10.17487/RFC7307, July 2014, <<http://www.rfc-editor.org/info/rfc7307>>.

19.2. Informative References

- [EnyediThesis] Enyedi, G., "Novel Algorithms for IP Fast Reroute", Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics Ph.D. Thesis, February 2011, <http://timon.tmit.bme.hu/theses/thesis_book.pdf>.
- [I-D.atlas-rtgwg-mrt-mc-arch] Atlas, A., Kebler, R., Wijnands, I., Csaszar, A., and G. Enyedi, "An Architecture for Multicast Protection Using Maximally Redundant Trees", draft-atlas-rtgwg-mrt-mc-arch-02 (work in progress), July 2013.
- [RFC2328] Moy, J., "OSPF Version 2", STD 54, RFC 2328, DOI 10.17487/RFC2328, April 1998, <<http://www.rfc-editor.org/info/rfc2328>>.
- [RFC4379] Kompella, K. and G. Swallow, "Detecting Multi-Protocol Label Switched (MPLS) Data Plane Failures", RFC 4379, DOI 10.17487/RFC4379, February 2006, <<http://www.rfc-editor.org/info/rfc4379>>.
- [RFC5286] Atlas, A., Ed. and A. Zinin, Ed., "Basic Specification for IP Fast Reroute: Loop-Free Alternates", RFC 5286, DOI 10.17487/RFC5286, September 2008, <<http://www.rfc-editor.org/info/rfc5286>>.
- [RFC5331] Aggarwal, R., Rekhter, Y., and E. Rosen, "MPLS Upstream Label Assignment and Context-Specific Label Space", RFC 5331, DOI 10.17487/RFC5331, August 2008, <<http://www.rfc-editor.org/info/rfc5331>>.
- [RFC5340] Coltun, R., Ferguson, D., Moy, J., and A. Lindem, "OSPF for IPv6", RFC 5340, DOI 10.17487/RFC5340, July 2008, <<http://www.rfc-editor.org/info/rfc5340>>.
- [RFC5443] Jork, M., Atlas, A., and L. Fang, "LDP IGP Synchronization", RFC 5443, DOI 10.17487/RFC5443, March 2009, <<http://www.rfc-editor.org/info/rfc5443>>.

- [RFC5714] Shand, M. and S. Bryant, "IP Fast Reroute Framework", RFC 5714, DOI 10.17487/RFC5714, January 2010, <<http://www.rfc-editor.org/info/rfc5714>>.
- [RFC5715] Shand, M. and S. Bryant, "A Framework for Loop-Free Convergence", RFC 5715, DOI 10.17487/RFC5715, January 2010, <<http://www.rfc-editor.org/info/rfc5715>>.
- [RFC6976] Shand, M., Bryant, S., Previdi, S., Filsfils, C., Francois, P., and O. Bonaventure, "Framework for Loop-Free Convergence Using the Ordered Forwarding Information Base (oFIB) Approach", RFC 6976, DOI 10.17487/RFC6976, July 2013, <<http://www.rfc-editor.org/info/rfc6976>>.
- [RFC6981] Bryant, S., Previdi, S., and M. Shand, "A Framework for IP and MPLS Fast Reroute Using Not-Via Addresses", RFC 6981, DOI 10.17487/RFC6981, August 2013, <<http://www.rfc-editor.org/info/rfc6981>>.
- [RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, DOI 10.17487/RFC6982, July 2013, <<http://www.rfc-editor.org/info/rfc6982>>.
- [RFC6987] Retana, A., Nguyen, L., Zinin, A., White, R., and D. McPherson, "OSPF Stub Router Advertisement", RFC 6987, DOI 10.17487/RFC6987, September 2013, <<http://www.rfc-editor.org/info/rfc6987>>.
- [RFC7490] Bryant, S., Filsfils, C., Previdi, S., Shand, M., and N. So, "Remote Loop-Free Alternate (LFA) Fast Reroute (FRR)", RFC 7490, DOI 10.17487/RFC7490, April 2015, <<http://www.rfc-editor.org/info/rfc7490>>.

Appendix A. Inter-level Forwarding Behavior for IS-IS

In the description below, we use the terms "Level-1-only interface", "Level-2-only interface", and "Level-1-and-Level-2 interface" to mean in interface which has formed only a Level-1 adjacency, only a Level-2 adjacency, or both Level-1 and Level-2 adjacencies. Note that IS-IS also defines the concept of areas. A router is configured with an IS-IS area identifier, and a given router may be configured with multiple IS-IS area identifiers. For an IS-IS Level-1 adjacency to form between two routers, at least one IS-IS area identifier must match. IS-IS Level-2 adjacencies do not require any area identifiers to match. The behavior described below does not explicitly refer to IS-IS area identifiers. However, IS-IS area identifiers will

indirectly affect the behavior by affecting the formation of Level-1 adjacencies.

First consider a packet destined to Z on MRT-Red or MRT-Blue received on a Level-1-only interface. If the best shortest path route to Z was learned from a Level-1 advertisement, then the packet should continue to be forwarded along MRT-Red or MRT-Blue. If instead the best route was learned from a Level-2 advertisement, then the packet should be removed from MRT-Red or MRT-Blue and forwarded on the shortest-path default forwarding topology.

Now consider a packet destined to Z on MRT-Red or MRT-Blue received on a Level-2-only interface. If the best route to Z was learned from a Level-2 advertisement, then the packet should continue to be forwarded along MRT-Red or MRT-Blue. If instead the best route was learned from a Level-1 advertisement, then the packet should be removed from MRT-Red or MRT-Blue and forwarded on the shortest-path default forwarding topology.

Finally, consider a packet destined to Z on MRT-Red or MRT-Blue received on a Level-1-and-Level-2 interface. This packet should continue to be forwarded along MRT-Red or MRT-Blue, regardless of which level the route was learned from.

An implementation may simplify the decision-making process above by using the interface of the next-hop for the route to Z to determine the level that the best route to Z was learned from. If the next-hop points out a Level-1-only interface, then the route was learned from a Level-1 advertisement. If the next-hop points out a Level-2-only interface, then the route was learned from a Level-2 advertisement. A next-hop that points out a Level-1-and-Level-2 interface does not provide enough information to determine the source of the best route. With this simplification, an implementation would need to continue forwarding along MRT-Red or MRT-Blue when the next-hop points out a Level-1-and-Level-2 interface. Therefore, a packet on MRT-Red or MRT-Blue going from Level-1 to Level-2 (or vice versa) that traverses a Level-1-and-Level-2 interface in the process will remain on MRT-Red or MRT-Blue. This simplification may not always produce the optimal forwarding behavior, but it does not introduce interoperability problems. The packet will stay on an MRT backup path longer than necessary, but it will still reach its destination.

Appendix B. General Issues with Area Abstraction

When a multi-homed prefix is connected in two different areas, it may be impractical to protect them without adding the complexity of explicit tunneling. This is also a problem for LFA and Remote-LFA.

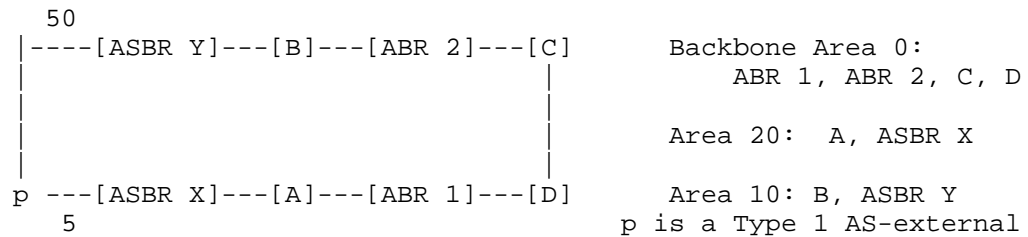


Figure 4: AS external prefixes in different areas

Consider the network in Figure 4 and assume there is a richer connective topology that isn't shown, where the same prefix is announced by ASBR X and ASBR Y which are in different non-backbone areas. If the link from A to ASBR X fails, then an MRT alternate could forward the packet to ABR 1 and ABR 1 could forward it to D, but then D would find the shortest route is back via ABR 1 to Area 20. This problem occurs because the routers, including the ABR, in one area are not yet aware of the failure in a different area.

The only way to get it from A to ASBR Y is to explicitly tunnel it to ASBR Y. If the traffic is unlabeled or the appropriate MPLS labels are known, then explicit tunneling MAY be used as long as the shortest-path of the tunnel avoids the failure point. In that case, A must determine that it should use an explicit tunnel instead of an MRT alternate.

Authors' Addresses

Alia Atlas
Juniper Networks
10 Technology Park Drive
Westford, MA 01886
USA

Email: akatlas@juniper.net

Chris Bowers
Juniper Networks
1194 N. Mathilda Ave.
Sunnyvale, CA 94089
USA

Email: cbowers@juniper.net

Gabor Sandor Enyedi
Ericsson
Konyves Kalman krt 11.
Budapest 1097
Hungary

Email: Gabor.Sandor.Enyedi@ericsson.com

Operations and Management Area Working Group
Internet-Draft
Intended status: Informational
Expires: January 5, 2015

D. King
Old Dog Consulting
M. Boucadair
France Telecom
S. Aldrin
Huawei USA
G. Mirsky
Ericsson
Q. Wu
Huawei
July 4, 2014

Use Cases and Requirements for Transport-Independent Multiple Layer OAM
draft-king-opsawg-time-multi-layer-oam-use-case-01

Abstract

This document identifies and discusses use-cases and high level requirements for transport technology independent OAM that need to interface multi-layer or multi-domain transport networks to cover heterogeneous networking technologies. As providers face multi-layer networks and diverse transport technologies, generic and integrated OAM is desirable for simplifying network operations and maintenance.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 5, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Terminology	3
2.1. Conventions used in this document	3
2.2. Acronyms and Abbreviations	3
3. Multi-Layer OAM use cases illustration	4
3.1. Multi-layer multi-domain OAM Consolidated in the Data Plane and Management Plane	4
3.2. OAM at Top of Layer 3	5
3.3. Overlay OAM	7
4. Requirements	8
5. IANA Considerations	9
6. Security Considerations	9
7. References	9
7.1. Normative References	9
7.2. Informative References	10
Authors' Addresses	10

1. Introduction

This document discusses use-cases for transport-independent OAM that need to interface multi-layer or multi-domain transport networks to cover heterogeneous networking technologies. As providers (e.g., network providers, data center providers, etc.) face multi-layer networks and diverse transport technologies, generic and integrated OAM is desirable for keeping network complexity down and simplifying O&M (OAM and O&M are used as specified in [RFC6291]).

This document is part of Transport Independent OAM in Multi-Layer Environment (TIME) effort which is meant to:

- o Understand and discuss situations where an OAM protocol can be tuned and optimized for a specific data plane.
- o OAM consolidations in the data plane:
 - * Exchange OAM information at the service layer atop of layer 3.

- * Deployed over various encapsulating protocols, and in various medium types.
- o OAM consolidations in the management plane:
 - * Abstract OAM information common to different layers.
 - * Expose OAM information via unified interface to management entities, independently of the layer they belong to.
 - * Discuss how information gathered from various layers can be correlated for the sake of network operations optimization purposes.
 - * Propose means to help during service diagnosis; these means may rely on filtering information to be leaked to other layers so that time recovery can be optimized. A typical example would be efficient root cause analysis that is fed with input from various layers.
 - * Propose means that would help to optimize a network as a whole instead of the monolithic approach that is specific to a given layer. For example, investigate means that would help in computing diverse and completely disjoint paths, not only at layer 3 but also at the physical layer.

These objectives are not frozen; further discussion is required to target key issues and scope the work to be conducted within IETF accordingly.

The problem statement and architecture is discussed in [TIME-PS].

2. Terminology

2.1. Conventions used in this document

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

2.2. Acronyms and Abbreviations

TIME - Transport Independent OAM in Multi-Layer Environment

OAM - Operations, Administration, and Maintenance

O&M - OAM and Management

3. Multi-Layer OAM use cases illustration

3.1. Multi-layer multi-domain OAM Consolidated in the Data Plane and Management Plane

Figure 1 illustrates a multi-layer network in which IP traffic between two customer edges is transported over both an IP/MPLS provider network and an Ethernet/MPLS provider network and multiple layers OAM are used. Ethernet OAM is used at the customer level for monitoring the end-to-end connection between the two customer edges, while IP OAM and MPLS OAM is used at the provider level for monitoring the connection between any two provider edges in each network. In addition to Ethernet OAM, transport independent OAM is also used for monitor end to end connection between the two customer edges at the abstract level.

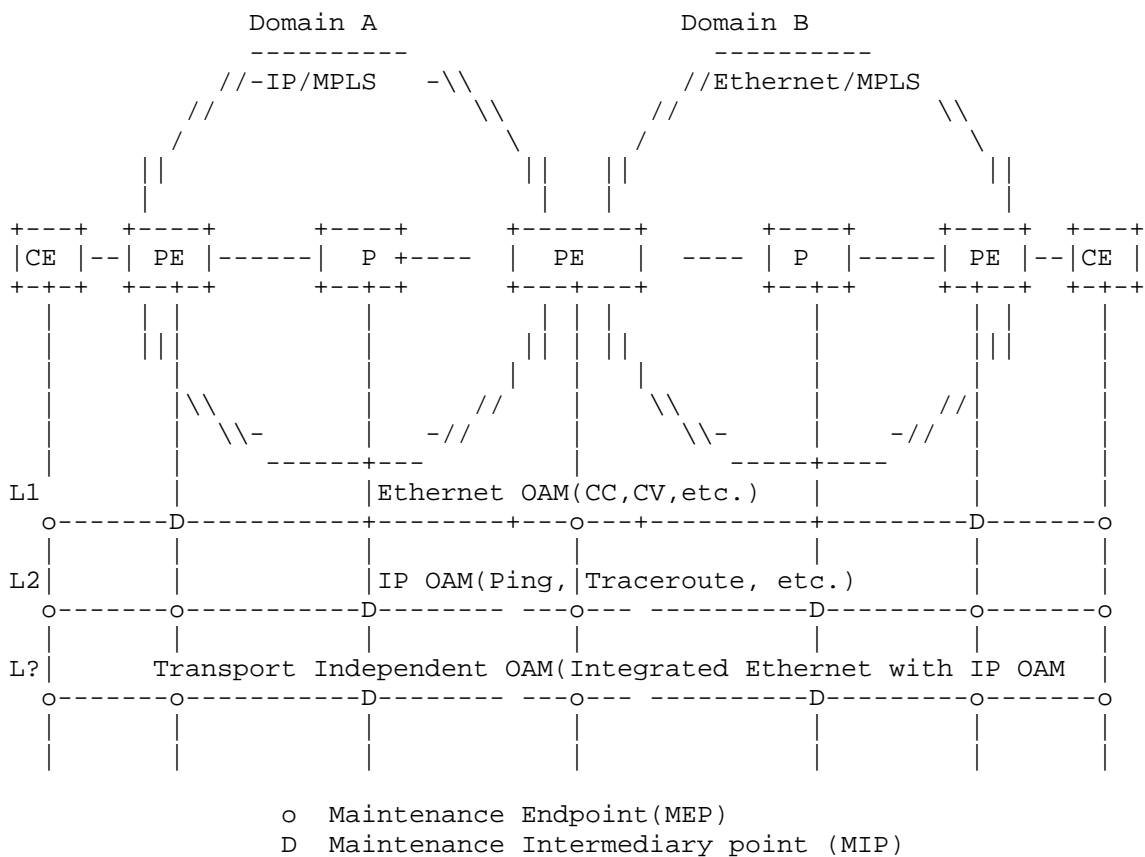


Figure 1: Multi-Domain Multi-Layer OAM

With transport independent OAM in the data plane, a user who wishes to issue a IP Ping Command or use connectivity verification command can do so in the same manner regardless of the underlying protocol or transport technology. Consider a scenario where both Ethernet OAM and IP OAM can be decomposed into a set of various OAM functions and an Ethernet OAM can be integrated with IP OAM in one protocol. When one OAM function is invoked, it will be invoked in the same way as the other OAM function regardless of the underlying protocol.

Alternatively, when Ethernet OAM and IP OAM can be consolidated through uniformed interface at the management plane, A user who wishes to issue a IP Ping command or a IP Traceroute or initiate a session monitoring can also do so in the same manner regardless of the underlying protocol or technology.

Consider a scenario where an IP ping to PE B from CE A failed. Between CE A and PE B there are IEEE 802.1 [IEEE-802.1Q] bridges a,b and c. Let's assume a,b and c are using [IEEE-802.1ag] CFM. Upon detecting IP layer ping failure, the user may wish to "go down" to the Ethernet layer and issue the corresponding fault verification (LBM/LBR) and fault isolation (LTM/LTR) tools, using the same API.

3.2. OAM at Top of Layer 3

In Service Function Chain ([I-D.ietf-sfc-problem-statement]), the service packets are steered through a set of Service Function Nodes distributed in the network. Overlay technologies (or tunneling techniques in general) can be used to stitch these Service Function Nodes in order to form end to end path (see Figure 2).

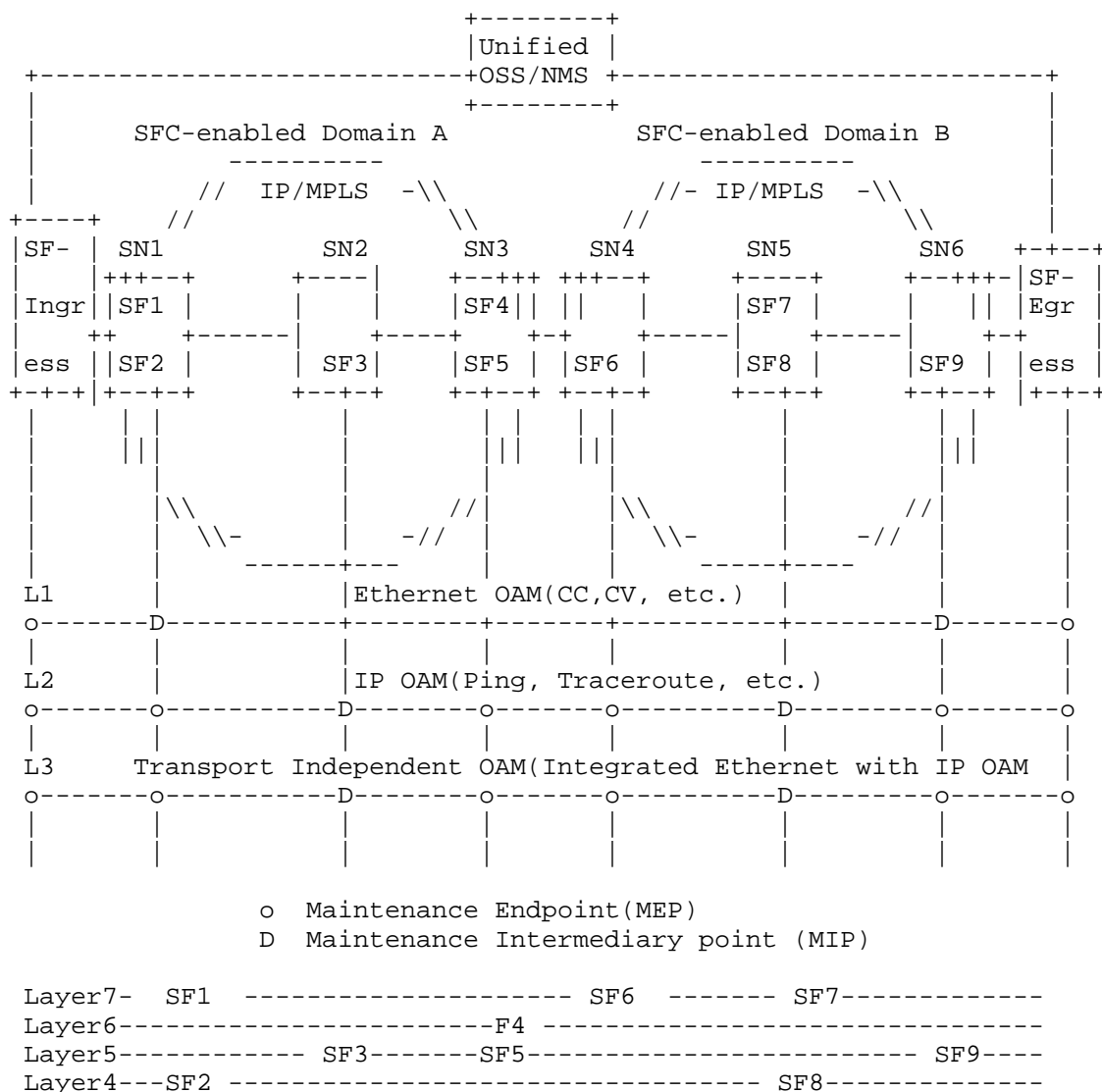


Figure 2: OAM at Top of Layer 3

When the service packet enters into the network, OAM information needs to be imposed by ingress node of the network into the packet (e.g., packet header extension or TLV extension in the overlay header) and pass through the network in the same path as the service traffic and processed by a set of Service Functions that are hosted in Service Nodes and located in different layers at the top of layer 3.

When any Service Nodes or any service segment between two Service Nodes fails to deliver user traffic, there is a need to provide a tool that would enable users to detect such failures, and a mechanism to isolate faults.

In case of several SFs co-located in the same Service Node, the packet is processed by all SFs in the Service Node, Once the packet is successfully handled by one SF, the packet is forwarded to the next SF that is in the same Service Node.

When the packet leaves the network, the OAM information needs to be stripped out from the packet.

To provide unified view of OAM information common to different layers and different domains, these OAM information needs to be gathered from various layers using different encapsulation and tunneling techniques and abstracted and provided to the management application via the unified management interface.

As indicated in [I-D.boucadair-sfc-requirements], the following OAM functions are to be supported:

- o Support means to verify the completion of the forwarding actions until the SFC Border Node is reached (see Section 3.4.1 of [RFC5706]).
- o Support means to ensure coherent classification rules are installed in and enforced by all the Classifiers of the SFC-enabled domain.
- o Support means to correlate classification policies with observed forwarding actions.
- o Support in-band liveness and functionality checking mechanisms for the instantiated Service Function Chains and the Service Functions that belong to these chains.

Other service diagnosis and troubleshooting requirements are discussed in [I-D.boucadair-sfc-requirements].

3.3. Overlay OAM

Overlay network is referred to a network that is built on top of another underlying network and provides various services to tenant system. With the growth of network virtualization technology, the needs for inter-connection between various overlay technologies/networks (e.g., VXLAN or NVGRE) in the Wide Area Network (WAN) become important since it can provide end-to-end connectivity.

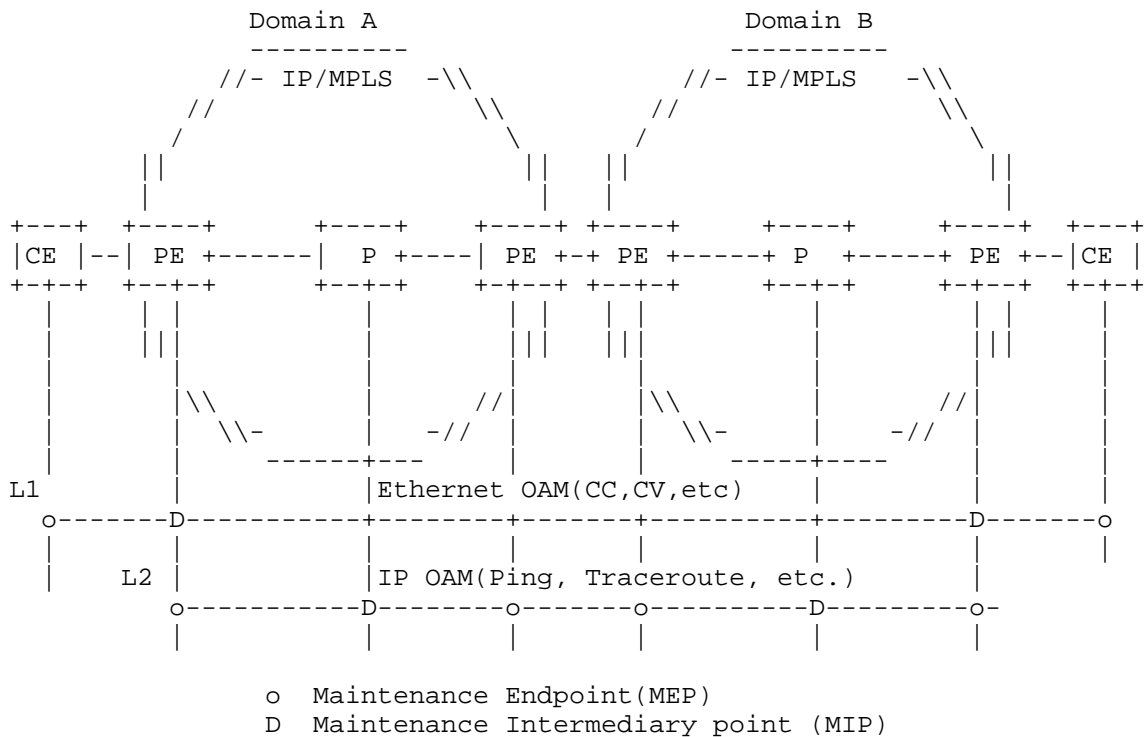


Figure 3: Overlay OAM

When a packet traverses a set of overlay networks in the data path, each overlay network will comprise an overlay segment used to connect overlay nodes in the same network and these overlay segment are stitched together to form end to end data path (Figure 3).

When any Overlay Segment fails to deliver user traffic, there is a need to provide a tool that would enable users to detect such failures, and a mechanism to isolate faults. It may also be desirable to test the data path before mapping user traffic to the Overlay Segment.

4. Requirements

This section identifies high-level requirements to fulfill transport independent OAM in Multi-layer Environment to support various use cases discussed in the previous sections.

- o The interfaces between the management entity and each Managed device in the transport network domain SHOULD support standards-based abstraction with a common information/data model.

- o The management entity should be able to create a single unified view of OAM information that is common to various layers, various domain and various operators.
- o The following capability should be supported:
 - * Support customized service diagnostic.
 - * Support diagnose the availability of a end-to-end path.
 - * Support diagnose the availability of a segment Path that is sub-path of end to end path.
 - * Support verification on the correct value of Path ID between any two pair of overlay nodes or any two pair of service nodes.
 - * Support verifying Overlay Control Plane and Data Plane consistency at either two overlay nodes or two service nodes.
 - * Support local diagnostic procedures specific to each Service Node.
 - * Support in-band liveness and functionality checking mechanisms for the overlay node or service node.
 - * Support Trace on the underlying network.

5. IANA Considerations

This memo includes no request to IANA.

6. Security Considerations

TBD.

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", March 1997.
- [TIME-PS] Wu, Q., "Problem Statement and Architecture for Transport-Independent Multiple Layer OAM", ID draft-ww-opsawg-multi-layer-oam-01, June 2014.

7.2. Informative References

- [I-D.boucadair-sfc-requirements]
Boucadair, M., Jacquenet, C., Jiang, Y., Parker, R., Pignataro, C., and K. Kengo, "Requirements for Service Function Chaining (SFC)", draft-boucadair-sfc-requirements-05 (work in progress), July 2014.
- [I-D.ietf-sfc-problem-statement]
Quinn, P. and T. Nadeau, "Service Function Chaining Problem Statement", draft-ietf-sfc-problem-statement-07 (work in progress), June 2014.
- [IEEE-802.1Q]
IEEE 802.1Q-2011, "IEEE standard for local and metropolitan area networks: Media access control (MAC) bridges and virtual bridged local area networks", August 2011.
- [IEEE-802.1ag]
IEEE 802.1ag-2007, "IEEE Standard for Local and Metropolitan Area Networks Virtual Bridged Local Area Networks Amendment 5: Connectivity Fault Management", December 2007.
- [RFC5706] Harrington, D., "Guidelines for Considering Operations and Management of New Protocols and Protocol Extensions", RFC 5706, November 2009.
- [RFC6291] Andersson, L., Helvoort, H., Bonica, R., Romascanu, D., and S. Mansfield, "Guidelines for the Use of the "OAM" Acronym in the IETF", RFC 6291, June 2011.

Authors' Addresses

Daniel King
Old Dog Consulting
UK

Email: daniel@olddog.co.uk

Mohamed Boucadair
France Telecom
Rennes 35000
France

Email: mohamed.boucadair@orange.com

Sam Aldrin
Huawei Technologies USA
2330 Central Expressway
NSanta Clara, CA 95051
USA

Email: aldrin.ietf@gmail.com

Greg Mirsky
Ericsson

Email: gregory.mirsky@ericsson.com

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: bill.wu@huawei.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 20, 2015

Z. Li
N. Wu
Q. Zhao
Huawei Technologies
A. Atlas
C. Bowers
Juniper Networks
J. Tantsura
Ericsson
January 16, 2015

Intermediate System to Intermediate System (IS-IS) Extensions for
Maximally Redundant Trees (MRT)
draft-li-isis-mrt-02

Abstract

This document describes necessary extensions to IS-IS to support the distributed computation of Maximally Redundant Trees (MRT). Some example uses of the MRTs include IP/LDP Fast-Reroute and global protection or live-live for multicast traffic. The extensions indicate what MRT profile(s) each router supports. Different MRT profiles can be defined to support different uses and to allow transition of capabilities. An extension is introduced to flood MRT-Ineligible links, due to administrative policy.

Status of This Memo

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

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

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

This Internet-Draft will expire on July 20, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
2. Requirements Language	3
3. Terminology	3
4. Using MRT with Multi-Topology IGP Routing	4
5. Overview of IS-IS Signaling Extensions for MRT	5
5.1. Supporting MRT Profiles	6
5.2. Electing GADAG Root	6
5.3. Advertising MRT-Ineligible Links for MRT	7
5.4. Triggering an MRT Computation	7
6. MRT Capability Advertisement	7
6.1. Advertising MRT Capability in IS-IS LSP	7
6.2. MRT Profile sub-TLV in IS-IS Router CAPABILITY TLV	8
6.3. MRT-Ineligible Links sub-TLV in IS-IS Router CAPABILITY TLV	9
7. Controlled Convergence sub-TLV in IS-IS Router CAPABILITY TLV	10
8. Handling MRT Capability Sending and Receiving	11
8.1. Advertising MRT extension	12
8.2. Parsing MRT extension	12
9. Backwards Compatibility	12
10. Implementation Status	13
11. Security Considerations	13
12. IANA Considerations	13
13. References	13
13.1. Normative References	13
13.2. Informative References	14
Authors' Addresses	14

1. Introduction

The IS-IS protocol is specified in [ISO10589], with extensions for supporting IPv4 and IPv6 specified in [RFC1195] and [RFC5308]. Each Intermediate System (IS) (router) advertises one or more IS-IS Link State Protocol Data Units (LSPs) with routing information. Each LSP is composed of a fixed header and a number of tuples, each consisting of a Type, a Length, and a Value. Such tuples are commonly known as TLVs, and are a good way of encoding information in a flexible and extensible format.

[I-D.ietf-rtgwg-mrt-frr-architecture] gives a complete solution for IP/LDP fast-reroute using Maximally Redundant Trees (MRT) to provide alternates. This document describes the necessary signaling extensions for supporting MRT-FRR used in IS-IS routing domain.

2. Requirements Language

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

3. Terminology

Redundant Trees (RT): A pair of trees where the path from any node X to the root R along the first tree is node-disjoint with the path from the same node X to the root R along the second tree. These can be computed in 2-connected graphs.

Maximally Redundant Trees (MRT): A pair of trees where the path from any node X to the root R along the first tree and the path from the same node X to the root R along the second tree share the minimum number of nodes and the minimum number of links. Each such shared node is a cut-vertex. Any shared links are cut-links. Any RT is an MRT but many MRTs are not RTs.

MRT Island: From the computing router, the set of routers that support a particular MRT profile and are connected via MRT-eligible links.

GADAG: Generalized Almost Directed Acyclic Graph - a graph which is the combination of the ADAGs of all blocks. Transforming a network graph into a GADAG is part of the MRT algorithm.

MRT-Red: MRT-Red is used to describe one of the two MRTs; it is used to describe the associated forwarding topology and MT-ID. Specifically, MRT-Red is the decreasing MRT where links in the GADAG

are taken in the direction from a higher topologically ordered node to a lower one.

MRT-Blue: MRT-Blue is used to describe one of the two MRTs; it is used to describe the associated forwarding topology and MT-ID. Specifically, MRT-Blue is the increasing MRT where links in the GADAG are taken in the direction from a lower topologically ordered node to a higher one.

4. Using MRT with Multi-Topology IGP Routing

Both IS-IS and OSPF have support for multi-topology routing (see [RFC5120] for ISIS and [RFC4915] for OSPF.) In addition to the standard topology (identified by MT-ID=0), these extensions allow the IGP to identify particular links and nodes as participating in additional topologies (identified by MT-ID!=0). A given link can belong to several topologies and be assigned different metrics in each topology. The IGP runs an independent SPF computation for each topology, finding independent shortest paths to prefixes in each topology.

It is straightforward to extend the MRT computations to multi-topology IGP routing. For each IGP topology identified by an IGP MT-ID, we need to identify the node and links belonging to an MRT Island for that IGP MT-ID. This process creates a graph for the MRT Island for that specific IGP MT-ID, which can then be used to compute the transit next-hops and alternate next-hops for MRT-Red and MRT-Blue for that specific IGP MT-ID.

We expect that initial implementation and deployments of MRT will be primarily concerned with computing MRT-Red and Blue trees for the standard topology (IGP MT-ID=0). However, we have chosen to specify the IS-IS MRT extensions to accommodate the computation of MRT-Red and MRT-Blue in a multi-topology IS-IS environment. This comes at the expense of 2-6 octets per TLV for MT-ID values, but it will allow for standards-based multi-topology aware MRT implementations for ISIS without any future standards work.

Using MRT in a multi-topology IGP environment does have one complication which should be discussed. Forwarding LDP traffic over MRT paths in the standard IGP topology requires the use of labels bound to topology-scoped FECs to identify traffic on MRT-Red and Blue trees. This is described in Section 6 of [I-D.ietf-rtgwg-mrt-frr-architecture]. To facilitate this, an MRT profile specifies IANA-assigned MRT-Red and MRT-Blue LDP MT-ID values, which are then used by LDP to advertise labels for the MRT-Red and Blue forwarding topologies. Note that the MRT-Red and MRT-Blue LDP MT-ID values assigned by IANA for a given MRT profile

correspond to the MRT-Red and Blue forwarding trees associated with the standard IGP topology with IGP MT-ID=0. For example, suppose that a future MRT profile X is assigned (hypothetical) MRT-Red and MRT-Blue LDP MT-ID values of 2001 and 2002. Then labels for shortest path forwarding trees associated with the standard IGP topology will be advertised using FECs with MT-ID=0, while the labels for the MRT-Red and Blue forwarding trees for profile X will be advertised using FECs with MT-ID=2001 and 2002, respectively. In the absence of multi-topology IGP routing, all MT-IDs used by LDP for MRT are assigned by IANA, so there are no potential conflicts in LDP MT-ID usage.

When MRT is used together with multi-topology IGP routing, additional LDP MT-IDs need to be specified for carrying traffic on the MRT-Red and Blue forwarding trees associated with the additional IGP routing topologies. Building on the previous example, suppose that a network is configured with an additional IGP routing topology using MT-ID=20, in addition to the standard topology with MT-ID=0. The router advertises support for MRT with respect to MT-ID=20 with profile X, as well as support for MRT with respect to MT-ID=0 with profile X. The MRT-Red and Blue LDP MT-IDs for MT-ID=0 with profile X are still inherited from profile X, as in the previous example. In order to use LDP to create the MRT-Red and Blue forwarding trees for the IGP topology with MT-ID=20, the router could, for example, advertise MRT-Red and MRT-Blue LDP MT-ID values of 21 and 22 for IGP MT-ID=20 and profile X. This overrides the (hypothetical) IANA-assigned values MRT-Red and MRT-Blue LDP MT-ID values for profile X, but maintains all other properties of profile X. Care must be taken to avoid advertising LDP MT-ID values that conflict with implicitly advertised IANA-assigned values LDP MT-ID.

The semantics of the IS-IS MRT extensions in this document are designed to handle the most common case (MRT in the absence of multi-topology IGP routing) in a simple manner. Setting the IGP MT-ID field as well as the MRT-Blue and MRT-Red LDP MT-ID fields to 0 in the TLV and sub-TLVs in this document results in the desired behavior for the standard IGP topology.

5. Overview of IS-IS Signaling Extensions for MRT

As stated in [I-D.ietf-rtgwg-mrt-frr-algorithm], it is necessary for each MRT-Capable router to compute MRT next hops in a consistent fashion. This is achieved by using same MRT profile and selecting the unique root in a MRT Island which is connected by MRT-Eligible links. Each of these issues will be discussed in following sections separately.

5.1. Supporting MRT Profiles

The contents and requirements of an MRT profile has been defined in [I-D.ietf-rtgwg-mrt-frr-architecture]. The parameters and behavioral rules contained in an MRT profile define one router's MRT capabilities. Based on common capabilities, one unified MRT Island is built.

The MRT-Capable router MUST advertise its corresponding MRT profiles by IS-IS protocol extension within IS-IS routing domain. The capabilities of advertiser MUST conform to the profile it claimed completely, especially the MT-IDs, the algorithm and the corresponding forwarding mechanism. This advertisement MUST have level scope. One router MAY support multiple MRT profiles and it MUST advertise these profiles in corresponding IS-IS level. The MT-IDs used in one supported MRT Profile MUST NOT overlap with those MT-IDs used in a different supported MRT Profile.

The default MRT Profile is defined in [I-D.ietf-rtgwg-mrt-frr-architecture]. Its behavior is intended to support IP/LDP unicast and multicast Fast-Reroute. MRT-Capable routers SHOULD support the default MRT profile.

5.2. Electing GADAG Root

As per [I-D.ietf-rtgwg-mrt-frr-algorithm], a GADAG root MUST be selected for one MRT Island. An unique GADAG root in common-sense among MRT Island routers is a necessity to do MRT computation. Since the selection of the GADAG root can affect the alternates and the traffic through it, the selection rules give network operator a knob to control the alternates and the traffic inside the MRT Island. Relevant discussion for the relationship between GADAG root role and MRT Island alternates is out of the scope of this document.

Each MRT-Capable router MUST advertise its priority for GADAG root selection. One router can only have one priority in the same MRT Island. It can have multiple priorities for different MRT Islands it supports. Routers that are marked as overloaded([RFC3787]) are not qualified as candidate for root selection.

The GADAG Root Selection Policy (defined as part of an MRT profile) may make use of the GADAG Root Selection Priority value advertised in the MRT Profile in the IS-IS Router CAPABILITY TLV. For example, the GADAG Root Selection Policy for the default MRT profile is the following: Among the routers in the MRT Island and with the highest priority advertised, an implementation MUST pick the router with the highest Router ID to be the GADAG root.

When the current root is out of service or new router with higher priority joined into the MRT Island, the GADAG root MUST be re-selected. A new MRT computation will be triggered because of such a topology change.

5.3. Advertising MRT-Ineligible Links for MRT

For certain administrative or management reason, some links may not be involved into MRT computation. In this scenario, MRT-Capable router MUST claim those MRT-Ineligible links are out of MRT Island scope. If such claim splits current MRT Island then MRT computation has to be done inside the modified MRT Island which the computing router belongs to.

5.4. Triggering an MRT Computation

A MRT Computation can be triggered through topology changes or MRT capability changes of any router in the MRT Island. It is always triggered for a given MRT Profile in the corresponding level. First, the associated MRT Island is determined. Then, the GADAG Root is selected. Finally, the actual MRT algorithm is run to compute the transit MRT-Red and MRT-Blue topologies. Additionally, the router MAY choose to compute MRT-FRR alternates or make other use of the MRT computation results.

Prefixes can be attached and detached and have their associated MRT-Red and MRT-Blue next-hops computed without requiring a new MRT computation.

6. MRT Capability Advertisement

MRT-Capable router MUST identify its MRT capabilities through IS-IS Link State Packet(LSP) in level scope.

6.1. Advertising MRT Capability in IS-IS LSP

One new M-bit is introduced into TLV 229 to identify router is MRT-Capable. Structure of TLV 229 is stated in [RFC5120] as pictured below:

TYPE: 229

LENGTH: total length of the value field, it SHOULD be 2 times the number of MT components.

VALUE: one or more 2-byte MT components, structured as follows:

		No. of Octets
+-----+ O A M R MT ID +-----+		2

Bit M identifies the originator is of MRT-Capable. The MRT-Blue and the MRT-Red alternates will be calculated for the MT identified by MT-ID.

This M-bit MUST be set and checked in LSP fragment 0. A MRT-Capable router MUST advertise this TLV with M-bit set for corresponding MT. For instance, if M-bit is set for MT-ID #0, MRT alternates will be calculated for standard topology.

If only M-bit is advertised for MRT-Capabilities without any other MRT information then the router is regarded as supporting default MRT profile with default GADAG root selection priority.

6.2. MRT Profile sub-TLV in IS-IS Router CAPABILITY TLV

A new MRT Profile sub-TLV is introduced into IS-IS Router CAPABILITY TLV[RFC4971] to advertise MRT capabilities. Since MRT is per level scope, the S-bit and D-bit of IS-IS Router CAPABILITY TLV MUST be set to zero. The structure of the MRT Profile sub-TLV is pictured as below:

TYPE: TBA-MRT-ISIS-1 (To Be Allocated by IANA)

LENGTH: 8

VALUE:

MT ID (2 octet with 4 bits reserved)

Profile ID (1 octet)

MRT-Red LDP MT-ID (2 octet)

MRT-Blue LDP MT-ID (2 octet)

+-----+ R R R R MT ID +-----+	2
+-----+ Profile ID +-----+	1
+-----+ GADAG Priority +-----+	1
+-----+ MRT-Red LDP MT-ID +-----+	2
+-----+ MRT-Blue LDP MT-ID +-----+	2

12-bit MT ID represents the base MT topology which MRT computation is based on. Profile ID represents the MRT profile this router supports and GADAG Root Selection Priority is the priority for root selection. The range of this priority is [0, 255] with 128 as the default value. The GADAG Root Selection Policy defined as part of a given MRT profile determine how the GADAG Root Selection Priority value is used.

If the MRT-Blue LDP MT-ID is 0, then the value specified in the associated MRT Profile is assumed. If the MRT-Red LDP MT-ID is 0, then the value specified in the associated MRT profile is assumed. The MRT-Blue LDP MT-ID and MRT-Red LDP MT-ID MUST NOT be the reserved values for LDP MT-IDs ([I-D.ietf-mpls-ldp-multi-topology]). The value for MRT-Blue LDP MT-ID and MRT-Red LDP MT-ID MUST be different except for 0. As stated above, the MRT-Blue LDP MT-ID and MRT-Red LDP MT-ID MUST NOT overlap among profiles if multiple MRT-Profile sub-TLVs are advertised.

This sub-TLV can occur multiple times if this router support multiple MRT profiles. This can happen during transition or to support multiple uses of MRT which prefer different profiles.

6.3. MRT-Ineligible Links sub-TLV in IS-IS Router CAPABILITY TLV

As a matter of policy, some links may not be available for the MRT computation, which can prevent alternates or traffic using these links. For instance, policy can be made to prevent fast-rerouted traffic from taking those links.

For a link to be excluded from the MRT computation, it MUST be advertised as sub-TLV in IS-IS Router CAPABILITY TLV which is in level scope with S-bit and D-bit unset. The MRT-Ineligible Link sub-TLV is structured as below:

TYPE: TBA-MRT-ISIS-2 (To Be Allocated by IANA)

LENGTH: from 9 to 255 octets

VALUE:

MT ID (2 octet with 4 bits reserved)

System ID and pseudo-node number (7 octet for each MRT-Ineligible Link)

	No. of Octets
+-----+ R R R R MT ID +-----+	2
+-----+ System ID and pseudonode number +-----+	7
+-----+ Default metric +-----+	3
. .	.
+-----+ System ID and pseudonode number +-----+	7
+-----+ Default metric +-----+	3

Each MRT-Ineligible Link is identified by neighbor's System ID and pseudo-node number and Default metric, same as IS Reachability TLV. This sub-TLV MAY occur multiple times if multiple links are ineligible.

7. Controlled Convergence sub-TLV in IS-IS Router CAPABILITY TLV

Section 12.2 of [I-D.ietf-rtgwg-mrt-frr-architecture] describes the need to wait for a configured or advertised period after a network failure to insure that all routers are using their new SPTs. Similarly, avoiding micro-forwarding loops during convergence [RFC5715] requires determining the maximum among all routers in the area of the worst-case route computation and FIB installation time. More details on the specific reasoning and need for flooding this value are given in [I-D.atlas-bryant-shand-lf-timers].

A new Controlled Convergence sub-TLV is introduced into the IS-IS Router CAPABILITY TLV [RFC4971] to advertise the worst-case time for a router to compute and install all IS-IS routes in the level after a change to a stable network. This advertisement has per level scope, so the S-bit and D-bit of IS-IS Router CAPABILITY TLV MUST be set to zero. The advertisement is scoped by IGP MT-ID, allowing a router supporting multi-topology IGP routing to advertise a different worst-

case compute and install time for each IGP topology. This make sense as the SPF computations for each IGP topology are independent of one another, and may have different worst-case compute and install times.

The structure of the Controlled Convergence sub-TLV is shown below:

TYPE: TBA-MRT-ISIS-3 (To Be Allocated by IANA)

LENGTH: 3

VALUE:

MT ID (2 octet with 4 bits reserved)

FIB compute/install time (1 octet)

```

+-----+
|R |R |R |R |          MT ID          |          2
+-----+-----+
|FIB comp/in time|          1
+-----+

```

The FIB compute/install time is the worst-case time the router may take to compute and install all IS-IS routes in the level after a change to a stable network. The value is in milliseconds.

The FIB compute/install time value sent by a router SHOULD be an estimate taking into account network scale or real-time measurements, or both. Advertisements SHOULD be dampened to avoid frequent communication of small changes in the FIB compute/install time.

A router receiving the Controlled Convergence sub-TLV SHOULD estimate the network convergence time as the maximum of the FIB compute/install times advertised by the routers in a level, including itself. In order to account for routers that do not advertise the Controlled Convergence sub-TLV, a router MAY use a locally configured minimum network convergence time as a lower bound on the computed network convergence time. A router MAY use a locally configured maximum network convergence time as an upper bound on the computed network convergence time.

8. Handling MRT Capability Sending and Receiving

The M-bit which identifies router's MRT capability MUST be advertised in LSP fragment 0. Those MRT related sub-TLVs SHOULD be ignored when MRT Capability bit is unset. When changes in MRT capabilities are received, a MRT computation SHOULD be triggered but MAY be delayed for a while to allow reception of all MRT-related information.

8.1. Advertising MRT extension

MRT sub-TLVs are encapsulated in the Router Capability TLV and advertised through LSP PDU for the level-wide. MRT sub-TLVs are optional. If one router does not support MRT, it MUST NOT advertise those sub-TLVs.

Since the advertisement scope of the MRT sub-TLV is level-wide, the D-Bit and S-Bit of the Router Capability TLV MUST be set as 0 when it is advertised. If other sub-TLVs in the Router Capability TLV need different values for those two bits, there MUST be an independent Router Capability TLV for MRT sub-TLVs.

When MRT related information is changed for the router or existing IS-IS LSP mechanisms are triggered for refreshing or updating, MRT sub-TLVs MUST be advertised if the router is MRT-Capable.

For administrative policies or reasons, it may be desirable to exclude certain links from the MRT computation. MRT-Ineligible sub-TLV is used to advertise which links should be excluded. Note that an interface advertised as MRT-Ineligible by a router is ineligible with respect to all profiles advertised by that router.

8.2. Parsing MRT extension

MRT extension MUST NOT affect the peer setup and the routing calculation of the standard topology.

MRT sub-TLVs SHOULD be validated like other sub-TLVs when received. MRT sub-TLVs SHOULD also be taken for the checksum calculation and authentication.

If MT-ID conflict is found for MRT-Red or MRT-blue from multiple sub-TLVs then those associated sub-TLVs MUST be ignored.

Links advertised in MRT-Ineligible sub-TLV MUST be precluded from MRT Computation. The removal of those links may change the computing router's MRT Island significantly.

9. Backwards Compatibility

The M-bit for MRT capability, the MRT Profile sub-TLV and the MRT-Ineligible Link sub-TLV defined in this document SHOULD NOT introduce any interoperability issues. Routers that do not support these MRT extensions SHOULD silently ignore them. Alternates or traffic MUST NOT be affected in current IS-IS routing domain.

10. Implementation Status

[RFC Editor: please remove this section prior to publication.]

Please see [I-D.ietf-rtgwg-mrt-frr-architecture] for details on implementation status.

11. Security Considerations

This IS-IS extension is not believed to introduce new security concerns.

12. IANA Considerations

Please allocate values for the following IS-IS Router CAPABILITY TLV Types [RFC4971]: MRT Profile sub-TLV (TBA-MRT-ISIS-1), MRT-Ineligible Link sub-TLV (TBA-MRT-ISIS-2), and Controlled Convergence sub-TLV (TBA-MRT-ISIS-3).

13. References

13.1. Normative References

- [I-D.ietf-mpls-ldp-multi-topology]
Zhao, Q., Raza, K., Zhou, C., Fang, L., Li, L., and D. King, "LDP Extensions for Multi Topology", draft-ietf-mpls-ldp-multi-topology-12 (work in progress), April 2014.
- [I-D.ietf-rtgwg-mrt-frr-algorithm]
Enyedi, G., Csaszar, A., Atlas, A., Bowers, C., and A. Gopalan, "Algorithms for computing Maximally Redundant Trees for IP/LDP Fast-Reroute", draft-rtgwg-mrt-frr-algorithm-01 (work in progress), July 2014.
- [I-D.ietf-rtgwg-mrt-frr-architecture]
Atlas, A., Kebler, R., Bowers, C., Enyedi, G., Csaszar, A., Tantsura, J., Konstantynowicz, M., and R. White, "An Architecture for IP/LDP Fast-Reroute Using Maximally Redundant Trees", draft-rtgwg-mrt-frr-architecture-04 (work in progress), July 2014.
- [RFC3137] Retana, A., Nguyen, L., White, R., Zinin, A., and D. McPherson, "OSPF Stub Router Advertisement", RFC 3137, June 2001.
- [RFC3787] Parker, J., "Recommendations for Interoperable IP Networks using Intermediate System to Intermediate System (IS-IS)", RFC 3787, May 2004.

13.2. Infomative References

- [I-D.atlas-bryant-shand-lf-timers]
K, A. and S. Bryant, "Synchronisation of Loop Free Timer Values", draft-atlas-bryant-shand-lf-timers-04 (work in progress), February 2008.
- [RFC1195] Callon, R., "Use of OSI IS-IS for routing in TCP/IP and dual environments", RFC 1195, December 1990.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4915] Psenak, P., Mirtorabi, S., Roy, A., Nguyen, L., and P. Pillay-Esnault, "Multi-Topology (MT) Routing in OSPF", RFC 4915, June 2007.
- [RFC4971] Vasseur, JP., Shen, N., and R. Aggarwal, "Intermediate System to Intermediate System (IS-IS) Extensions for Advertising Router Information", RFC 4971, July 2007.
- [RFC5120] Przygienda, T., Shen, N., and N. Sheth, "M-ISIS: Multi Topology (MT) Routing in Intermediate System to Intermediate Systems (IS-ISs)", RFC 5120, February 2008.
- [RFC5308] Hopps, C., "Routing IPv6 with IS-IS", RFC 5308, October 2008.
- [RFC5715] Shand, M. and S. Bryant, "A Framework for Loop-Free Convergence", RFC 5715, January 2010.

Authors' Addresses

Zhenbin Li
Huawei Technologies
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095
China

Email: lizhenbin@huawei.com

Nan Wu
Huawei Technologies
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095
China

Email: eric.wu@huawei.com

Quintin Zhao
Huawei Technologies
125 Nagog Technology Park
Acton, MA 01719
USA

Alia Atlas
Juniper Networks
10 Technology Park Drive
Westford, MA 01886
USA

Email: akatlas@juniper.net

Chris Bowers
Juniper Networks
1194 N. Mathilda Ave.
Sunnyvale, CA 94089
USA

Email: cbowers@juniper.net

Jeff Tantsura
Ericsson
300 Holger Way
San Jose, CA 95134
USA

Email: jeff.tantsura@ericsson.com

Routing Area Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 5, 2015

S. Litkowski
Orange Business Service
March 4, 2015

Link State protocols SPF trigger and delay algorithm impact on IGP
microloops
draft-litkowski-rtgwg-spf-uloop-pb-statement-02

Abstract

A micro-loop is a packet forwarding loop that may occur transiently among two or more routers in a hop-by-hop packet forwarding paradigm.

In this document, we are trying to analyze the impact of using different Link State IGP implementations in a single network in regards of microloops. The analysis is focused on the SPF triggers and SPF delay algorithm.

Requirements Language

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

Status of This Memo

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

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

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

This Internet-Draft will expire on September 5, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Problem statement	3
3. SPF trigger strategies	4
4. SPF delay strategies	5
4.1. Two step SPF delay	5
4.2. Exponential backoff	6
5. Mixing strategies	7
6. Proposed work items	11
7. Security Considerations	13
8. Acknowledgements	13
9. IANA Considerations	13
10. Normative References	13
Author's Address	13

1. Introduction

Link State IGP protocols are based on a topology database on which a SPF (Shortest Path First) algorithm like Dijkstra is implemented to find the optimal routing paths.

Specifications like IS-IS ([RFC1195]) propose some optimization of the route computation (See Appendix C.1) but not all the implementations are following those not mandatory optimizations.

We will call SPF trigger, the events that would lead to a new SPF computation based on the topology.

Link State IGP protocols, like OSPF ([RFC2328]) and IS-IS ([RFC1195]), are using plenty of timers to control the router behavior in case of churn : SPF delay, PRC delay, LSP generation delay, LSP flooding delay, LSP retransmission interval ...

Some of those timers are standardized in protocol specification, some are not especially the SPF computation related timers.

For non standardized timers, implementations are free to implement it in any way. For some standardized timer, we can also see that rather than using static configurable values for such timer, implementations may offer dynamically adjusted timers to help controlling the churn.

We will call SPF delay, the delay timer that exists in most implementations that makes codes to wait before running SPF computation after a SPF trigger is received.

A micro-loop is a packet forwarding loop that may occur transiently among two or more routers in a hop-by-hop packet forwarding paradigm. We can observe that these micro-loops are formed when two routers do not update their Forwarding Information Base (FIB) for a certain prefix at the same time. The micro-loop phenomenon is described in [I-D.ietf-rtgwg-microloop-analysis].

Routers have more and more powerful controlplane and dataplane that reduce the Control plane to Forwarding plane overhead during the convergence process. Even if FIB update is still reasonably the highest contributor in the convergence time for large network, its duration is reducing more and more and may become comparable to protocol timers. This is particular true in small and medium networks.

In multi vendor networks, using different implementations of a link state protocol may favor micro-loops creation during convergence time due to deprecancies of timers. Service Providers are already aware to use similar timers for all the network as best practice, but sometimes it is not possible due to limitation of implementations.

This document will present why it sounds important for service provider to have consistent implementations of Link State protocols across vendors. We are particularly analyzing the impact of using different Link State IGP implementations in a single network in regards of microloops. The analysis is focused on the SPF triggers and SPF delay algorithm in a first step.

This document is only stating the problem, and defining some work items but its not intended to provide a solution.

2. Problem statement

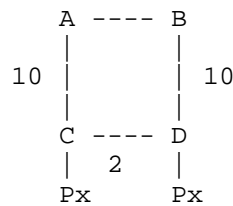


Figure 1

In the figure above, A uses primarily the AC link to reach C. When the AC link fails, IGP convergence occurs. If A converges before B, A will forward traffic to C through B, but as B as not converged yet, B will loop back traffic to A, leading to a microloop.

The micro-loop appears due to the asynchronous convergence of nodes in a network when a event occurs.

Multiple factors (and combination of these factors) may increase the probability for a micro-loop to appear :

- o delay of failure notification : the more B is advised of the failure later than A, the more a micro-loop may appear.
- o SPF delay : most of the implementations supports a delay for the SPF computation to try to catch as many events as possible. If A uses a SPF delay timer of x msec and B uses a SPF delay timer of y msec and $x < y$, B would start converging after A leading to a potential microloop.
- o SPF computation time : mostly a matter of CPU power and optimizations like incremental SPF. If A computes SPF faster than B, there is a chance for a microloop to appear. CPUs are today faster enough to consider SPF computation time as negligible (order of msec in a large network).
- o RIB and FIB prefix insertion speed or ordering : highly implementation dependant.

This document will focus on analysis SPF delay (and associated triggers).

3. SPF trigger strategies

Depending of the change advertised in LSP/LSA, the topology may be affected or not. An implementation can decide to not run SPF (and only run IP reachability) if the advertised change is not affecting topology.

Different strategies exists to trigger SPF :

1. Always run full SPF whatever the change to process.
2. Run only Full SPF when required : e.g. if a link fails, a local node will run an SPF for its local LSP update. If the LSP from the neighbor (describing the same failure) is received after SPF has started, the local node can decide that a new full SPF is not required as the topology has not change.
3. If topology does not change, only recompute reachability.

As pointed in Section 1, SPF optimization are not mandatory in specifications, leading to multiple strategies to be implemented.

4. SPF delay strategies

Implementations of link state routing protocols use different strategies to delay SPF :

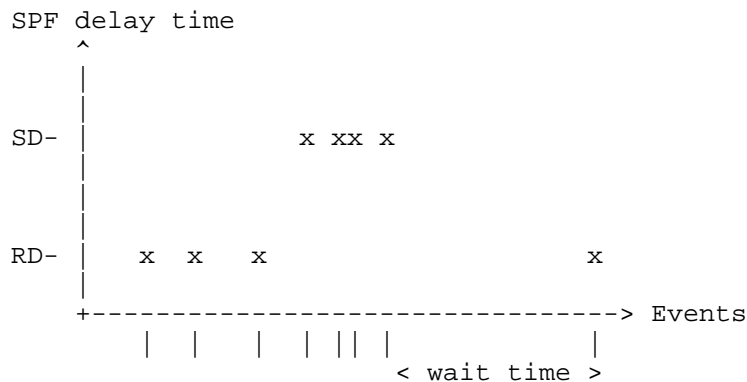
1. Two steps.
2. Exponential backoff.

4.1. Two step SPF delay

The SPF delay is managed by four parameters :

- o Rapid delay : amount of time to wait before running SPF.
- o Rapid runs : amount of consecutive SPF runs that can run using rapid delay. When amount is exceeded router moves to slow delay.
- o Slow delay : amount of time to wait before running SPF.
- o Wait time : amount of time to wait without events before going back to rapid delay.

Example : Rapid delay = 50msec, Rapid runs = 3, Slow delay = 1sec,
Wait time = 2sec

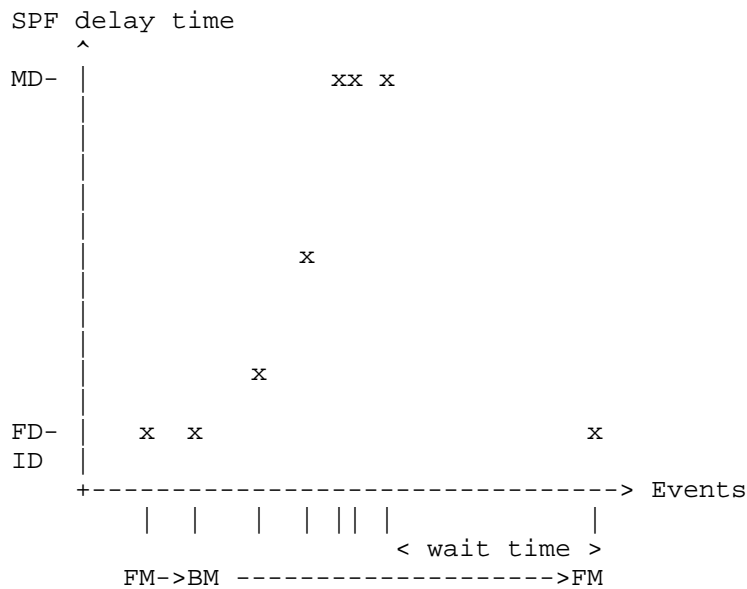


4.2. Exponential backoff

The algorithm has two mode : fast mode and backoff mode. In backoff mode, the SPF delay is increasing exponentially at each run. The SPF delay is managed by four parameters :

- o First delay : amount of time to wait before running SPF. This delay is used on when SPF is in fast mode.
- o Incremental delay : amount of time to wait before running SPF. This delay is used on when SPF is in backoff mode and increments exponentially at each SPF run.
- o Maximum delay : maximum amount of time to wait before running SPF.
- o Wait time : amount of time to wait without events before going back to fast mode.

Example : First delay = 50msec, Incremental delay = 50msec, Maximum delay = 1sec, Wait time = 2sec



5. Mixing strategies

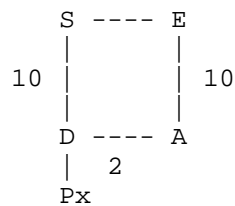
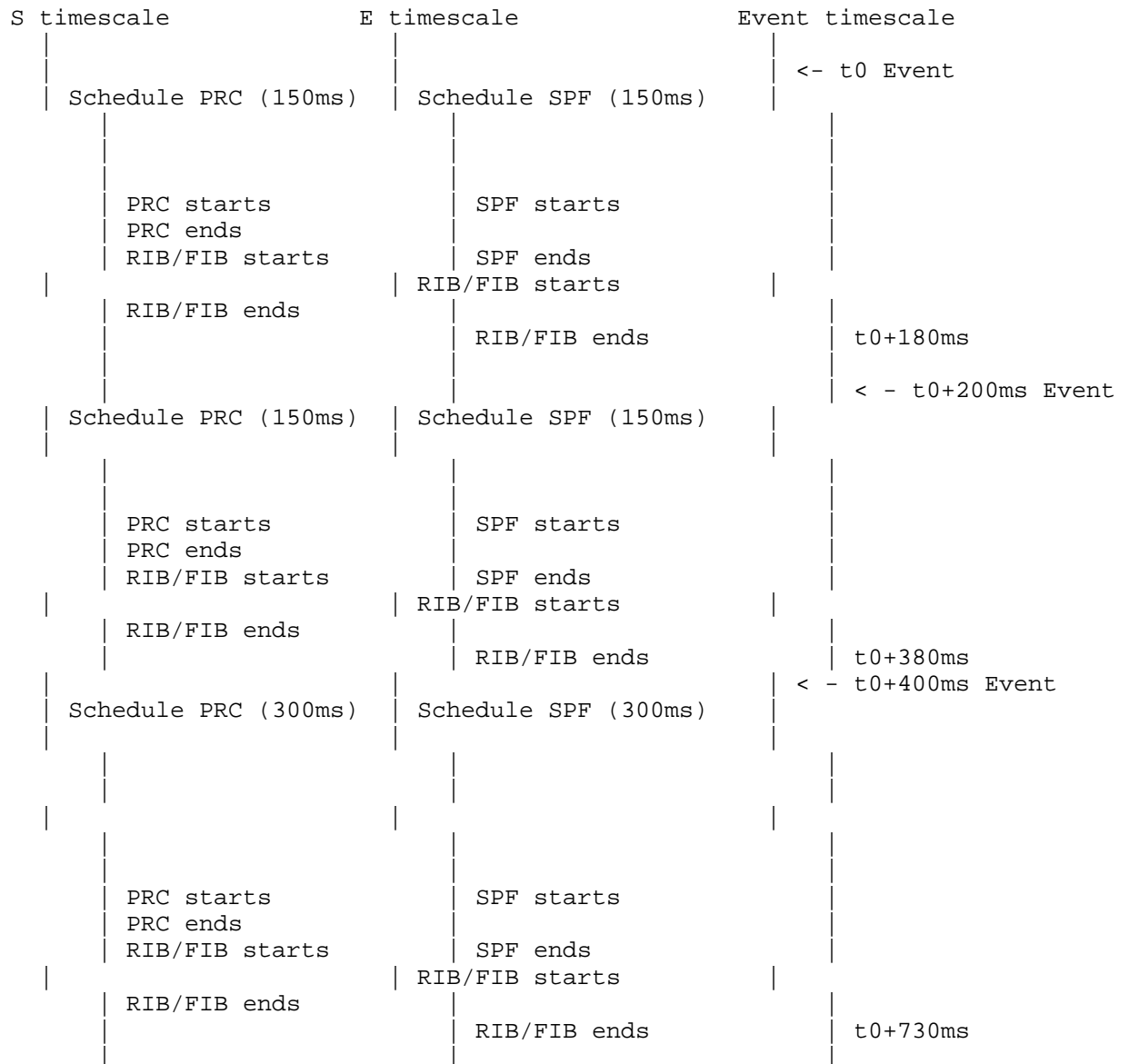


Figure 2

In the diagram above, we consider a flow of packet from S to D. We consider that S is using optimized SPF triggering (Full SPF is triggered only when necessary), and two steps SPF delay (rapid=150ms,rapid-runs=3, slow=1s). As implementation of S is optimized, Partial Reachability Computation (PRC) is available. We consider the same timers as SPF for delaying PRC. We consider that E is using a SPF trigger strategy that always compute Full SPF and exponential backoff strategy for SPF delay (start=150ms, inc=150ms, max=1s)

We also consider the following sequence of events (note : the timescale does not intend to represent a real router timescale where jitters are introduced to all timers) :

- o t0 : a prefix is declared down in the network.
- o t0+200ms : the prefix is declared as up.
- o t0+400ms : a prefix is declared down in the network.
- o t0+1000ms : S-D link fails.



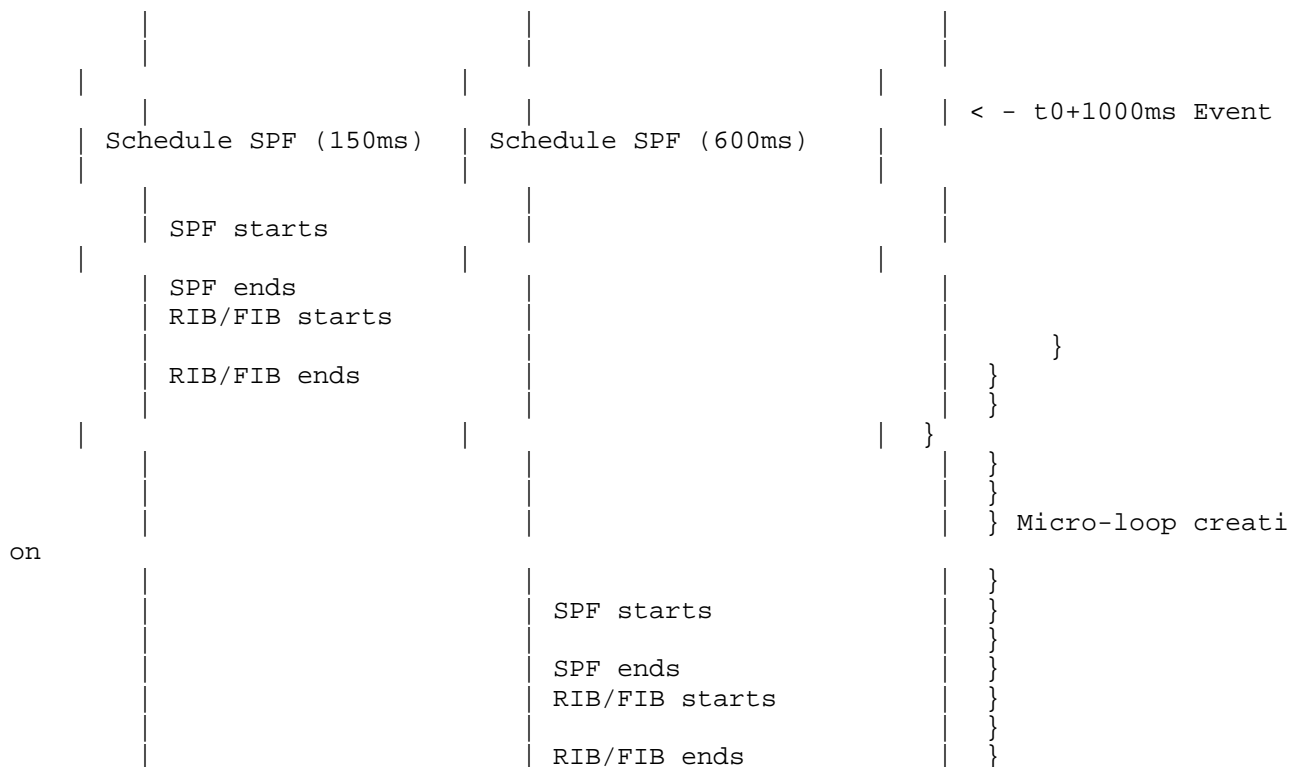
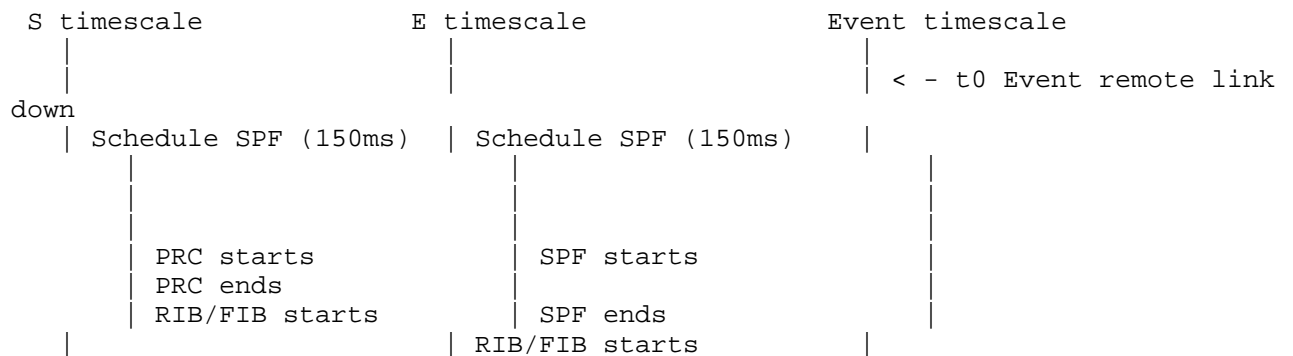
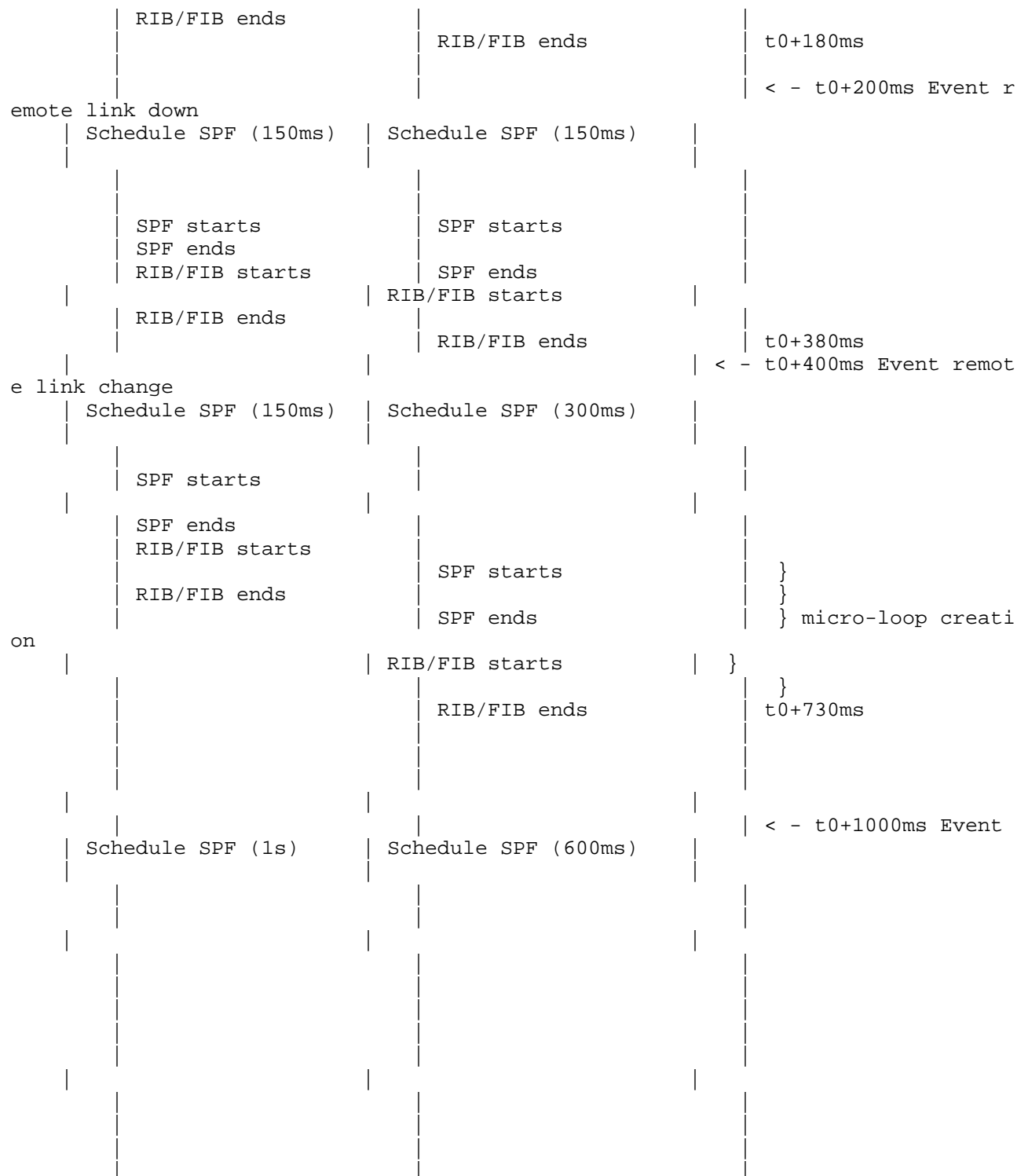


Figure 3

In the figure above, we can see that due to deprecancies in SPF management, after multiple events (different types of event), SPF delays are completely misaligned between nodes leading to long microloop creation.

The same issue can also appear with only single type of events as displayed below :





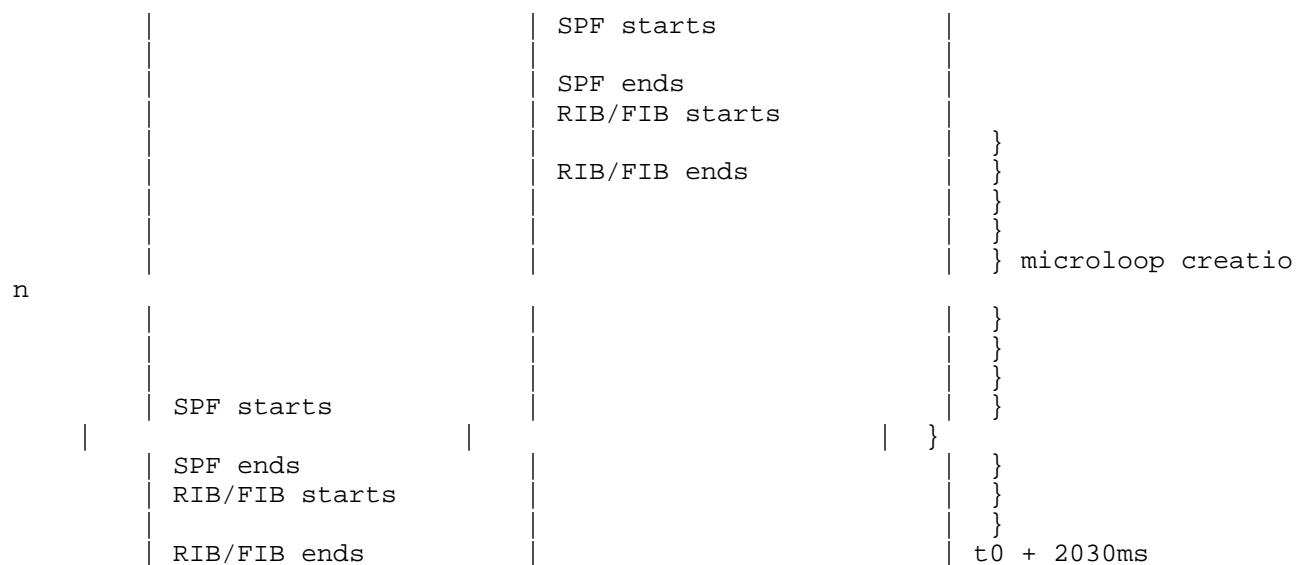


Figure 4

6. Proposed work items

In order to enhance the current LinkState IGP behavior, authors would encourage working on standardization of some behaviors.

Authors are proposing the following work items :

- o Standardize SPF trigger strategy.
- o Standardize computation timer scope : single timer for all computation operations, separated timers ...
- o Standardize "slowdown" timer algorithm including its association to a particular timer : authors of this document does not presume that the same algorithm must be used for all timers.

Using the same event sequence as in figure 2, we may expect fewer and/or shorter microloops using standardized implementations.

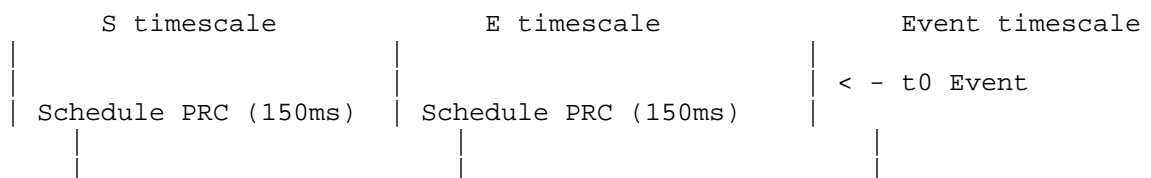






Figure 5

As displayed above, there could be some other parameters like router computation power, flooding timers that may also influence microloops. In the figure 5, we consider E to be a bit slower than S, leading to microloop creation. Despite of this, we expect that by aligning implementations at least on SPF trigger and SPF delay, service provider may reduce number or duration of microloops.

7. Security Considerations

This document does not introduce any security consideration.

8. Acknowledgements

9. IANA Considerations

This document has no action for IANA.

10. Normative References

- [I-D.ietf-rtgwg-microloop-analysis]
Zinin, A., "Analysis and Minimization of Microloops in Link-state Routing Protocols", draft-ietf-rtgwg-microloop-analysis-01 (work in progress), October 2005.
- [RFC1195] Callon, R., "Use of OSI IS-IS for routing in TCP/IP and dual environments", RFC 1195, December 1990.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2328] Moy, J., "OSPF Version 2", STD 54, RFC 2328, April 1998.

Author's Address

Stephane Litkowski
Orange Business Service

Email: stephane.litkowski@orange.com

Routing Area Working Group
Internet-Draft
Intended status: Informational
Expires: June 3, 2022

Bin Liu
ZTE Inc., ZTE Plaza
Yantao Sun
Jing Cheng
Yichen Zhang
Beijing Jiaotong University
Bhumip Khasnabish
Individual contributor
Nov 30, 2021

Generic Fault-Avoidance Routing Protocol for Data Center Networks
draft-sl-rtgwg-far-dcn-17

Abstract

This document describes a generic routing method and protocol for a regular data center network, named the Fault-Avoidance Routing (FAR) protocol. The FAR protocol provides a generic routing method for all types of regular topology network architectures that have been proposed for large-scale cloud-based data centers over the past few years. The FAR protocol is designed to leverage any regularity in the topology and compute its routing table in a concise manner. Fat-tree is taken as an example architecture to illustrate how the FAR protocol can be applied in real operational scenarios.

Status of This Memo

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

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

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

This Internet-Draft will expire on June 3, 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Acronyms & Definitions	5
2. Conventions used in this document	5
3. Problem Statement	5
3.1. The Impact of Large-scale Networks on Route Calculation .	6
3.2. Issues of Conventional Routing Methods in a Large-scale Network with Giant Number Nodes of Routers	6
3.3. Network Addressing Issues	9
3.4. Big Routing Table Issues	9
3.5. Adaptivity Issues for Routing Algorithms	9
3.6. Virtual Machine Migration Issues	10
4. The FAR Framework	10
5. Data Format	11
5.1. Data Tables	11
5.2. Messages	14
6. FAR Modules	18
6.1. Neighbor and Link Detection Module(M1)	18
6.2. Device Learning Module(M2)	18
6.3. Invisible Neighbor and Link Failure Inferring Module(M3)	19
6.4. Link Failure Learning Module(M4)	19
6.5. BRT Building Module(M5)	19
6.6. NRT Building Module(M6)	20
6.7. Routing Table Lookup(M7)	20
7. How a FAR Router Works	20
8. Compatible Architecture	23
9. Topology identification and broadcast storm suppression . . .	23
10. Application Example	24
10.1. BRT Building Procedure	26
10.2. NRT Building Procedure	27
10.2.1. Single Link Failure	27
10.2.2. A Group of Link Failures	28
10.2.3. Node Failures	29
10.3. Routing Procedure	29
10.4. FAR's Performance in Large-scale Networks	31
10.4.1. The number of control messages required by FAR . . .	31
10.4.2. The Calculating Time of Routing Tables	31

10.4.3. The Size of Routing Tables	31
11. Implementations Examples	32
12. Security Considerations	34
13. Conclusions	35
14. Acknowledgments	35
15. References	35
15.1. Normative References	35
15.2. Informative References	35
16. Appendix	36
16.1. Application Area of the Solution	36
16.2. Technical evolution roadmap	36
16.3. Updating roadmap	36
Authors' Addresses	36

1. Introduction

In recent years, with the rapid development of cloud computing technologies, the widely deployed cloud services, such as Amazon EC2 and Google search, bring about huge challenges to data center networking (DCN). Today's cloud-based data centers (DCs) require large-scale networks with larger internal bandwidth and smaller transfer delay. However, conventional networks cannot meet such requirements due to limitations in their network architecture. In order to satisfy the requirements of cloud computing services, many new network architectures have been proposed for data centers, such as Fat-tree, MatrixDCN[MatrixDCN], and BCube[BCube]. These new architectures can support non-blocking large-scale datacenter networks with more than tens of thousands of physical servers.

All of these architectures have regular topologies, which are common features. The regular topology refers to the network topology structure with obvious regularity and symmetry, which is conducive to automatic configuration of the network, such as the Fat-tree network. In a regular topology, each network node such as a switch or router can be addressed by its location and through a node's address, the node's connections to its neighbors in a network can be determined, and furthermore, the route to the node from other nodes in the network can be determined. So nodes can compute route entries without learning topology.

This document describes a generic routing method and protocol, the Fault-Avoidance Routing (FAR) protocol, for DCNs. This method leverages the regularity in the topologies of data center networks to simplify routing learning and accelerate the query of routing tables. This routing method has a better fault tolerance and can be applied to any DCN with a regular topology.

FAR is not a routing protocol to replace generic routing protocols such as OSPF(Open Shortest Path First)[RFC2328] and IS-IS(Intermediate System-to-Intermediate System). It cannot be used in general local networks whose topological structures are arbitrary, and whose scales are also not very large. OSPF and IS-IS work very well in such a network. But in a large-scale network with regular topology, FAR has better performance. Compared with OSPF and IS-IS, FAR has shorter time of network convergence and lower PDU(Protocol Data Unit) overhead. Furthermore, FAR requires less computing and storage resources, which lets FAR routers to run at a lower cost of production than the generic routers.

In addition, for each type of network architecture, researchers designed a routing algorithm according to the features of its topology. Because these routing algorithms are different and lack compatibility with each other, it is very difficult to develop a routing protocol for network routers supporting multiple routing algorithms. FAR has better adaptability than these specified routing methods.

FAR consists of three components, i.e., link state learning unit, routing table building unit and routing table querying unit. In the link state learning unit, FAR exchanges link failures among routers to establish a consistent knowledge of the entire network. In this stage, the regularity in topology is exploited to infer failed links and routers. In the routing table building unit, FAR builds up two routing tables, i.e., a basic routing table (BRT) and a negative routing table (NRT), for each router according to the network topology and link states. In the last component, routers forward incoming packets by looking up the two routing tables. The matched entries in BRT minus the matched entries in NRT are the final route entries to be used to forward an incoming packet.

This document describes a protocol developed by ZTE and Beijing Jiaotong University. It is just presented here to record the work and to make it available for use in later IETF work if desirable.

The remainder of this draft is organized as follows. The problem to be addressed by FAR is described in Section 3. The framework of FAR routing protocol is described in Section 4. Section 5 and 6 introduce FAR's data format FAR and modules in detail. Section 7 describe how FAR works by finite state machine (FSM). In Section 8, we discussed how FAR works with variable network architectures. Section 9 takes Fat-tree network as an example to illuminate how FAR works.

1.1. Acronyms & Definitions

DCN - Data Center Network

FAR - Fault-Avoidance Routing

BRT - Basic Routing Table

NRT - Negative Routing Table

NDT - Neighbor Devices Table

ADT - All Devices Table

LFT - Link Failure Table

DA - Device Announcement

LFA - Link Failure Announcement

DLR - Device and Link Request

IP - Internet Protocol

VM - Virtual Machine

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here. In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying special significance.

3. Problem Statement

The problem to be addressed by FAR as proposed in this draft is described in this section. The expansion of Cloud data center networks has brought significant challenges to the existing routing technologies. FAR mainly solves a series of routing problems faced by large-scale data center networks.

3.1. The Impact of Large-scale Networks on Route Calculation

In a large-scale cloud data center network, there may be thousands of routers. Running OSPF or IS-IS in such network will encounter these two challenges:

(1) Network convergence time would be too long, which will cause a longer time to elapse for creating and updating the routes. The response time to network failures may be excessively long;

(2) High resource consumption. Since a large number of routing protocol packets need to be sent, it causes the routing information consuming too much network bandwidth and CPU(central processing unit) resources, and easily leads to packet loss and makes the challenge (1) more prominent.

In order to solve these two challenges, a common practice is to partition a large network into some small areas, where the route calculation runs independently within different areas. However, nowadays the cloud data centers typically require very large internal bandwidth. To meet this requirement, a large number of parallel equivalent links are deployed in a network, such as a Fat-tree network. Partitioning such a network will affect the utilization of routing algorithm on equivalent multi-path and reduce internal network bandwidth requirements.

In the FAR routing calculation process, a Basic Routing Table (BRT) is built on local network topology leveraging the regularity of the network topologies. In addition to BRT, FAR also builds a Negative Routing Table (NRT). FAR gradually builds NRT in the process of learning network link failure information, which does not require learning a complete network fault information. FAR does not need to wait for the completion of the network convergence in the process of building these two tables. Therefore, it avoids the problem of excessive network convergence overheads in the route calculation process. In addition, FAR only needs to exchange a small amount of link change information between routers, and hence consumes less network bandwidth.

3.2. Issues of Conventional Routing Methods in a Large-scale Network with Giant Number Nodes of Routers

There are many real world scenario where tens of thousands of nodes(or much more nodes) need to be deployed in a flat area, such as infiniband routing and switching system, high-performance computer network, and many IDC(Internet Data Center) networks in China. The similar problems have been existed long ago. People have solved the problems through similar solutions, such as the traditional regular

topology-based RFC3619[RFC3619] protocol, the routing protocols of infiniband routing and switching system, and high-performance computer network routing protocol.

Infiniband defines a switch-based network to interconnect processing nodes and the I/O nodes. Infiniband can support very large scale networks, use the regularity in topology to simplify its routing algorithm, which is just the same to what we do in FAR.

Why OSPF and IS-IS do not work well in a large-scale network with giant number nodes of routers?

As we know, the OSPF protocol uses multiple databases, more topological exchange information (as seen in the following example) and complicated algorithm. It requires routers to consume more memory and CPU processing capability. But the processing rate of CPU on the protocol message per second is very limited. When the network expands, CPU will quickly approach its processing limits, and at this time OSPF can not continue to expand the scale of the management. The SPF(Shortest Path First) algorithm itself does not thoroughly solve these problems.

On the contrary, the FAR protocol does not need to calculate SPF, which saves calculation time and resources, so FAR does not have the convergence time delay and the additional CPU overheads, which SPF requires. Because in the initial stage, FAR already knows the regular information of the whole network topology and does not need to periodically do SPF operation.

One of the examples of "more topological exchange information": In the OSPF protocol, LSA(Link-State Advertisement) floods every 1800 seconds. Especially in the larger network, the occupation of CPU and band bandwidth will soon reach the router's performance bottleneck. In order to reduce these adverse effects, OSPF introduced the concept of Area, which still has not solved the problem thoroughly. By dividing the OSPF Area into several areas, the routers in the same area do not need to know the topological details outside their area. (In comparison with FAR, after OSPF introducing the concept of Area, the equivalent paths cannot be selected in the whole network scope)

OSPF can achieve the following results by Area : 1) Routers only need to maintain the same link state databases as other routers within the same Area, without the necessity of maintaining the same link state database as all routers in the whole OSPF domain. 2) The reduction of the link state databases means dealing with relatively fewer LSA, which reduces the CPU consumption of routers; 3) The large number of LSAs flood only within the same Area. But, its negative effect is that the smaller number of routers which can be managed in each OSPF area. On the contrary, because FAR does not have the above

disadvantages, FAR can also manage large-scale network even without dividing Areas.

The aging time of OSPF is set in order to adapt to routing transformation and protocol message exchange happened frequently in the irregular topology. Its negative effect is: when the network does not change, the LSA needs to be refreshed every 1800 seconds to reset the aging time. In the regular topology, as the routings are fixed, it does not need the complex protocol message exchange and aging rules to reflect the routing changes, as long as LFA mechanism in the FAR is enough.

Compared with the LSVR(Link State Vector Routing) protocol, the LSVR protocol has no special requirements for the network topology structure, however, the FAR draft is applicable to the regular topology network architecture and simplifies unnecessary processing. It is a solution proposed to greatly improve the routing efficiency of the regular network topology. The FAR solution is more efficient than the general methods such as LSVR in regular topology.

Therefore, in FAR, we can omit many unnecessary processing and the packet exchange. The benefits are fast convergence speed and much larger network scale than other dynamic routing protocol. Now there are some successful implementations of simplified routings in the regular topology in the HPC(High Performance Computing) environment. Conclusion: As FAR needs few routing entries and the topology is regular, the database does not need to be updated regularly. Without the need for aging, there is no need for CPU and bandwidth overhead brought by LSA flood every 30 minutes, so the expansion of the network has no obvious effect on the performance of FAR, which is contrary to OSPF.

Comparison of convergence time: The settings of OSPF `spf_delay` and `spf_hold_time` can affect the change of convergence time. The convergence time of the network with 2480 nodes is about 15-20 seconds; while the FAR does not need to calculate the SPF, so there is no such convergence time.

These issues still exist in rapid convergence technology of OSPF, ISIS (such as I-SPF, Incremental SPF) and LSVR. The convergence speed and network scale constraint each other. FAR does not have the above problems, and the convergence time is almost negligible. Can FRR(Fast Reroute) solve these problems? IP FRR has some limitations. The establishment of IP FRR backup scheme will not affect the original topology and traffic forwarding which are established by protocol, however, we can not get the information of whereabouts and status when the traffic is switched to an alternate next hop.

3.3. Network Addressing Issues

Routers are typically configured with multiple network interfaces, each connected to a subnet. OSPF and other routing algorithms require that each interface of a router must be configured with an IP address. A large-scale data center network may contain thousands of routers and each router has dozens of network interfaces, thus, there are tens of thousands of IP addresses needed to be configured in a data center. It will be very complex to configure and manage a large number of network interfaces and will be difficult to troubleshoot network problems, then network maintenance will be costly and error-prone.

In FAR, the device position information is encoded in the IP address of the router. Each router only needs to be assigned a unique IP address according its location, which greatly solves complex network addressing issues in large-scale networks.

3.4. Big Routing Table Issues

There are a large number of subnets in the large-scale data center network. A router may build a routing entry for each subnet, and therefore the size of routing tables on each router may be very large. It will increase a router's cost and reduce the querying speed of the routing table.

FAR uses two measures to reduce the size of its routing tables: a) It builds a BRT on the regularity of the network topologies; b) It introduces a new routing table, i.e., a NRT. In this way FAR can reduce the size of routing tables to only a few dozen routing entries.

3.5. Adaptivity Issues for Routing Algorithms

To implement efficient routing in large-scale datacenters, besides FAR, some other routing methods are proposed for some specific network architectures, such as Fat-tree and BCube. These routing methods are different (from both design and implementation viewpoints) and not compatible with the conventional routing methods, which brings big troubles to network equipment providers to develop new routers supporting various new routing methods.

FAR is a generic routing method. With slight modification, FAR method can be applied to most of regular datacenter networks. Furthermore, the structure of routing tables and querying a routing table in FAR are the same as conventional routing method. If FAR is adopted, the workload of developing a new type of router will be significantly decreased.

3.6. Virtual Machine Migration Issues

Supporting VM migration is very important for cloud-based datacenter networks. However, in order to support layer-3 routing, routing methods including OSPF and FAR require limiting VM migration within a subnet. For this paradox, the mainstream methods still utilize layer-3 routing on routers or switches, transmit packets encapsulated by IPinIP or MACinIP between hosts by tunnels passing through network to the destination access switch, and then extract original packet out and send it to the destination host.

By utilizing the aforementioned methods, FAR can be applied to Fat-tree, MatrixDCN or BCube networks for supporting VM migration in entire network.

4. The FAR Framework

FAR requires that a DCN has a regular topology, and network devices, including routers, switches, and servers, are assigned IP addresses according to their locations in the network. In other word, we can locate a device in the network according to its IP address.

FAR is a distributed routing method. In order to support FAR, each router needs to have a routing module that implements the FAR algorithm. FAR algorithm is composed of three parts, i.e., link-state learning, routing table building and routing table querying, as shown in Fig. 1.

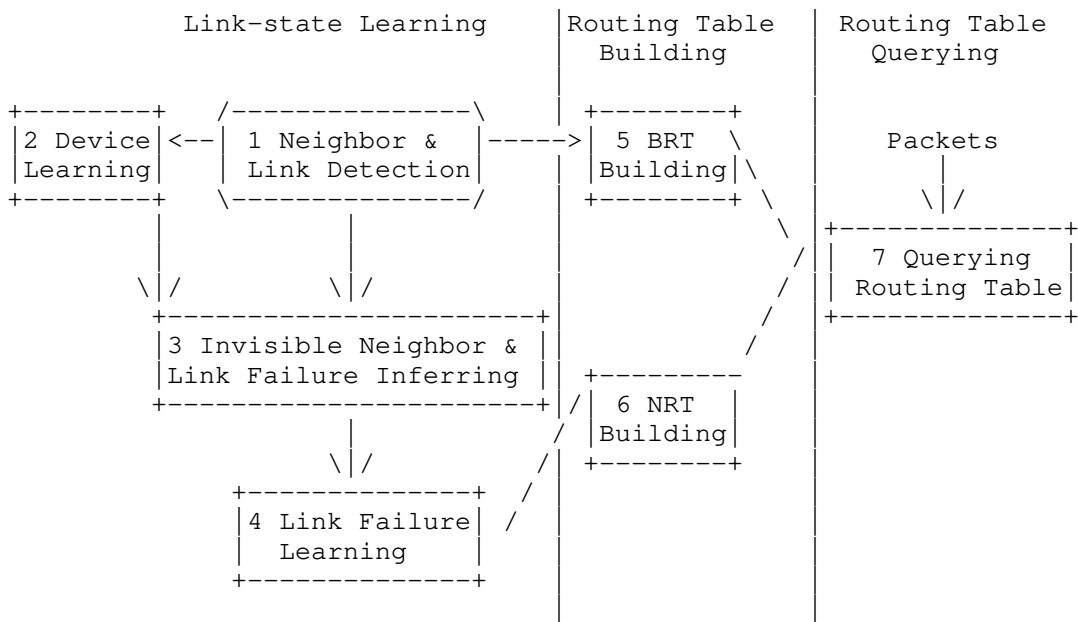


Figure 1: The FAR framework

- 1:Neighbor and Link Detection Module(M1)
- 2:Device Learning Module(M2)
- 3:Invisible Neighbor and Link Failure Inferring Module(M3)
- 4:Link Failure Learning Module(M4)
- 5:BRT Building Module(M5)
- 6:NRT Building Module(M6)
- 7:Routing Table Lookup(M7)

The meanings of M1-M7 are explained in detail in section 6. Link-state learning is responsible for a router to detect the states of its connected links and learn the states of all the other links in the entire network. The second part builds two routing tables, a basic routing table (BRT) and an negative routing table (NRT), according to the learned link states in the first part. The third part queries the BRT and the NRT to decide a next forwarding hop for the received (ingress) packets.

5. Data Format

5.1. Data Tables

Some data tables are maintained on each router in FAR. They are:

Neighbor Device Table (NDT): To store neighbor routers and related links.

All Devices Table (ADT): To store all routers in the entire network.

Link Failures Table (LFT): To store all link failures in the entire network.

Basic Routing Table (BRT): To store the candidate routes.

Negative Routing Table(NRT): To store the avoiding routes.

The format of NDT

```
-----  
Device ID | Device IP | Port ID | Link State | Update Time  
-----
```

Device ID: The ID of a neighbor router.

Device IP: The IP address of a neighbor router.

Port ID: The port ID that a neighbor router is attached to.

Link State: The state of the link between a router and its neighbor router. There are two states: Up and Down.

Update Time: The time of updating the entry.

The format of ADT

```
-----  
Device ID | Device IP | Type | State | Update Time  
-----
```

Device ID: The ID of a neighbor router.

Device IP: The IP address of a neighbor router.

Type: The type of a neighbor router.

State: The state of a neighbor router. There are two states: Up and Down.

Update Time: The time of updating the entry.

The format of LFT

```
-----
No | Router 1 IP | Router 2 IP | Timestamp
-----
```

No: The entry number.

Router 1 IP: The IP address of one router that a failed link connects to.

Router 2 IP: The IP address of another router that a failed link connects to.

Timestamp: It identifies when the entry is created.

The format of BRT

```
-----
Destination | Mask | Next Hop | Interface | Update Time
-----
```

Destination: A destination network

Mask: The subnet mask of a destination network.

Next Hop: The IP address of a next hop for a destination.

Interface: The interface related to a next hop.

Update Time: The time of updating the entry.

The format of NRT

```
-----
Destination| Mask| Next Hop| Interface| Failed Link No| Timestamp
-----
```

Destination: A destination network.

Mask: The subnet mask of a destination network.

Next Hop: The IP address of a next hop that should be avoided for a destination.

Interface: The interface related to a next hop that should be avoided.

Failed Link No: A group of failed link numbers divided by "/", for example 1/2/3.

Timestamp: The time of updating the entry.

5.2. Messages

Some protocol messages are exchanged between routers in FAR.

Hello Message: This message is exchanged between neighbor routers to learn adjacency.

Device Announcement (DA): Synchronize the knowledge of routers between routers.

Link Failure Announcement (LFA): Synchronize link failures between routers.

Device and Link Request (DLR): When a router starts, it requests the knowledge of routers and links from its neighbors by a DLR message.

A FAR Message is directly encapsulated in an IP packet. The protocol field of IP header indicates an IP packet is an FAR message.

The four types of FAR messages have same format of packet header, called FAR header (as shown in Figure 2).

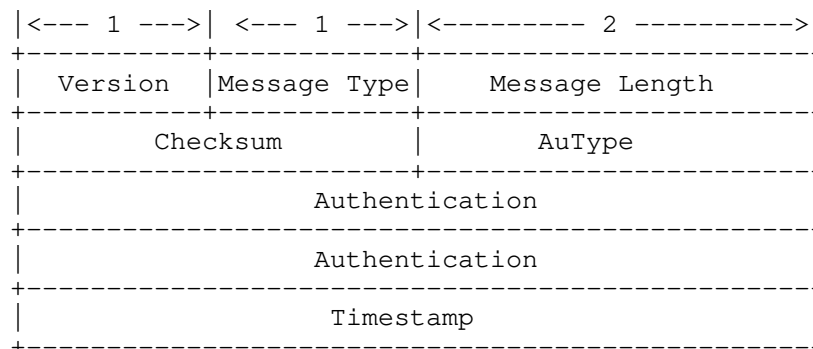


Figure 2: The format of FAR header

Version: FAR version

Message Type: The type of FAR message.

Packet Length: The packet length of the total FAR message.

Checksum: The checksum of an entire FAR message.

AuType: Authentication type. 0: no authentication, 1: Plaintext Authentication, 2: MD5 Authentication.

Authentication: Authentication information. 0: undefined, 1: Key, 2: key ID, MD5 data length and packet number. MD5 data is appended to the backend of the packet.

AuType and Authentication can refer to the definition of OSPF packet.

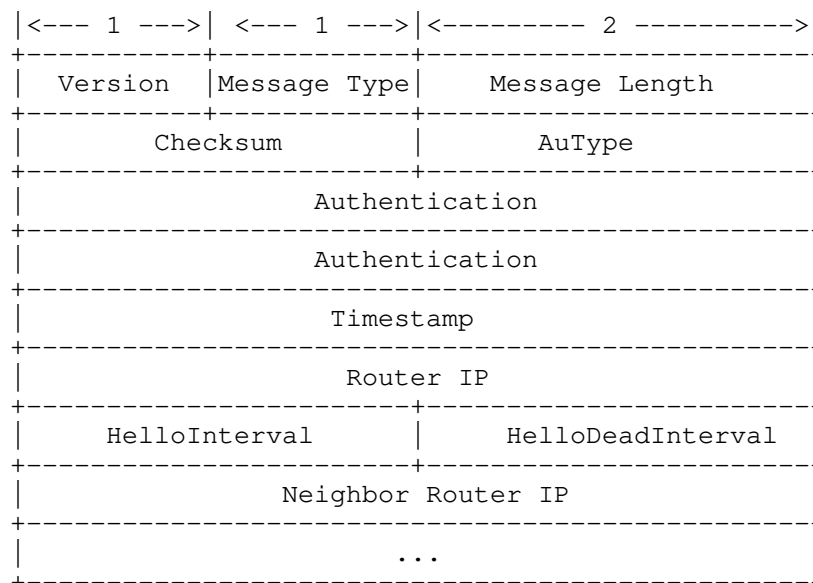


Figure 3: The Format of Hello Messages

For Hello messages, the Message Type in FAR header is set to 1. Besides FAR header, a Hello message (Fig. 3) requires the following fields:

Router IP: The router IP address.

HelloInterval: The interval of sending Hello messages to neighbor routers.

RouterDeadInterval: The interval to set a neighbor router dead(out-of-service). If in the interval time, a router doesn't receive a Hello message from its neighbor router, the neighbor router is treated as dead.

Neighbor Router IP: The IP address of a neighbor router. All the neighbor router's addresses should be included in a Hello message.

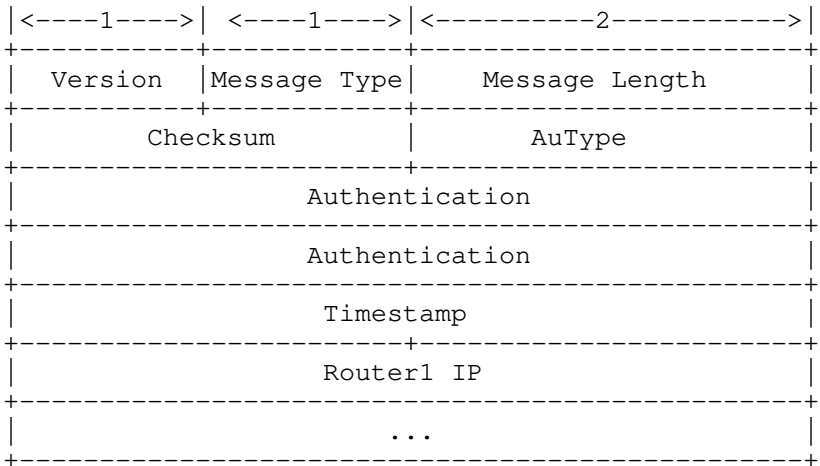


Figure 4: The Format of DA Messages

For DA messages(Fig. 4), the Message Type in FAR header is set to 2. Besides FAR header, a DA message includes IP addresses of all the announced routers.

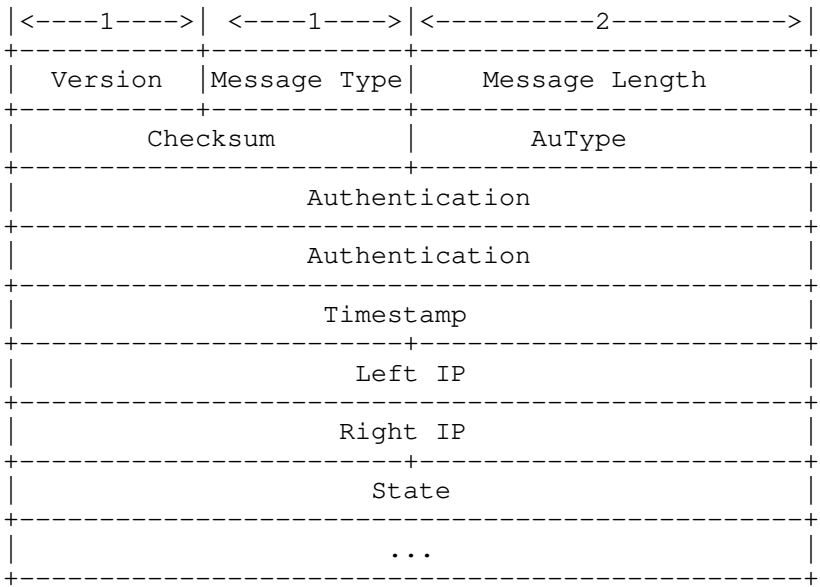


Figure 5: The Format of LFA Messages

For LFA messages(Fig. 5), the Message Type in FAR header is set to 3. Besides FAR header, a LFA message includes all the announced link failures.

Left IP: The IP address of the left endpoint router of a link.

Right IP: The IP address of the right endpoint router of a link.

State: Link state. 0: Up, 1: down

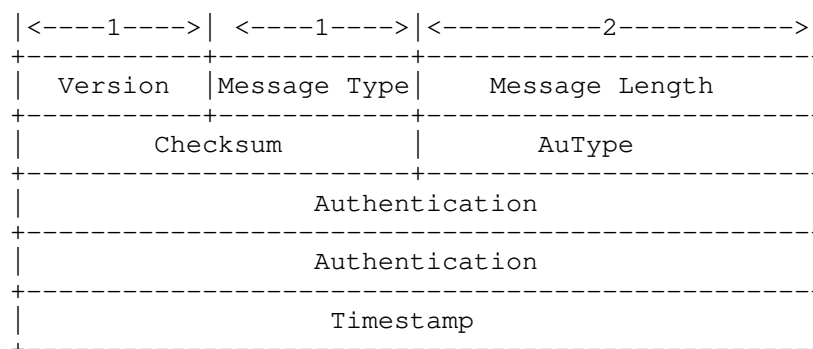


Figure 6: The Format of DLR Messages

For DLR messages (Fig. 6), the Message Type in FAR header is set to 1. Except for FAR header, DLR has no additional fields.

6. FAR Modules

6.1. Neighbor and Link Detection Module (M1)

M1 is responsible for sending and receiving Hello messages, and detecting directly-connected links and neighbor routers. Each Hello message is encapsulated in an IP packet. M1 sends Hello messages periodically to all the active router ports and receives Hello messages from its neighbor routers. M1 detects neighbor routers and directly-connected links according to received Hello Messages and stores these neighbors and links into a Neighbor Devices Table (NDT). Additionally, M1 also stores neighbor routers into an All Devices Table (ADT).

6.2. Device Learning Module (M2)

M2 is responsible for sending, receiving, and forwarding device announcement (DA) messages, learning all the routers in the whole network, and deducing faulted routers. When a router starts, it sends a DA message announcing itself to its neighbors and a DLR message requesting the knowledge of routers and links from its neighbors. If M2 module of a router receives a DA message, it checks whether the router encapsulated in the message is in an ADT. If the router is not in the ADT, M2 puts this router into the ADT and forwards this DA message to all the active ports except for the incoming one, otherwise, M2 discards this message directly. If M2 module of a router receives a DLR message, it replies a DA message that encapsulates all of the learned routers.

6.3. Invisible Neighbor and Link Failure Inferring Module(M3)

M3 is responsible for inferring invisible neighbors of the current router by means of the ADT. If the link between a router A and its neighbor B breaks, which results in that M1 module of A cannot detect the existence of B, then B is an invisible neighbor of A. Since a device's location is coded into its IP address, it can be judged whether two routers are adjacent, according to their IP addresses. Based on this idea, M3 infers all of the invisible neighbors of the current router and the related link failures. The results are stored into an NDT. Moreover, link failures also are added into a link-failure table (LFT). LFT stores all of the failed links in the entire network.

6.4. Link Failure Learning Module(M4)

M4 is responsible for sending, receiving and forwarding link failure announcement (LFA) and learning all the link failures in the whole network. M4 broadcasts each newly inferred link failure to all the routers in the network. Each link failure is encapsulated in a LFA message and one link failure is broadcasted only once. If a router receives a DLR request from its neighbor, it will reply a LFA message that encapsulates all the learned link failures through M4 module. If M4 receives a LFA message, it checks whether the link failure encapsulated in the message is in a LFT by comparing two link ends and timestamp. If the link failure is not in the LFT or timestamp is different, M4 puts this link failure into the LFT (or update timestamp only) and forwards this LFA message to all the active ports except for the incoming one, otherwise, M4 discards this message directly.

There is a special case a router will rebroadcast a link failure. If a router receives a data packet and must forward the packet going ahead to destination through a failed link, it means some previous router should avoid this failed link according to its NRT but it doesn't. In this case, maybe the previous router missed the LFA message of the link failure due to some uncertain reasons. So the forwarding router rebroadcasts the LFA message.

6.5. BRT Building Module(M5)

M5 is responsible for building a BRT for the current router. By leveraging the regularity in topology, M5 can calculate the routing paths for any destination without the knowledge of the topology of whole network, and then build the BRT based on an NDT. Since the IP addresses of network devices are continuous, M5 only creates one route entry for a group of destination addresses that have the same network prefix by means of route aggregation technology. Usually,

the size of a BRT is very small. The detail of how to build a BRT is described in section 5.

6.6. NRT Building Module (M6)

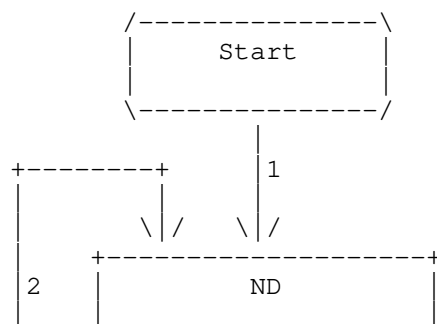
M6 is responsible for building a NRT for the current router. Because M5 builds a BRT without considering link failures in network, the routing paths calculated by the BRT cannot avoid failed links. To solve this problem, a NRT is used to exclude the routing paths that include some failed links from the paths calculated by a BRT. M6 calculate the routing paths that include failed links and stored them into the NRT. The details of how to build a NRT is described in section 5.

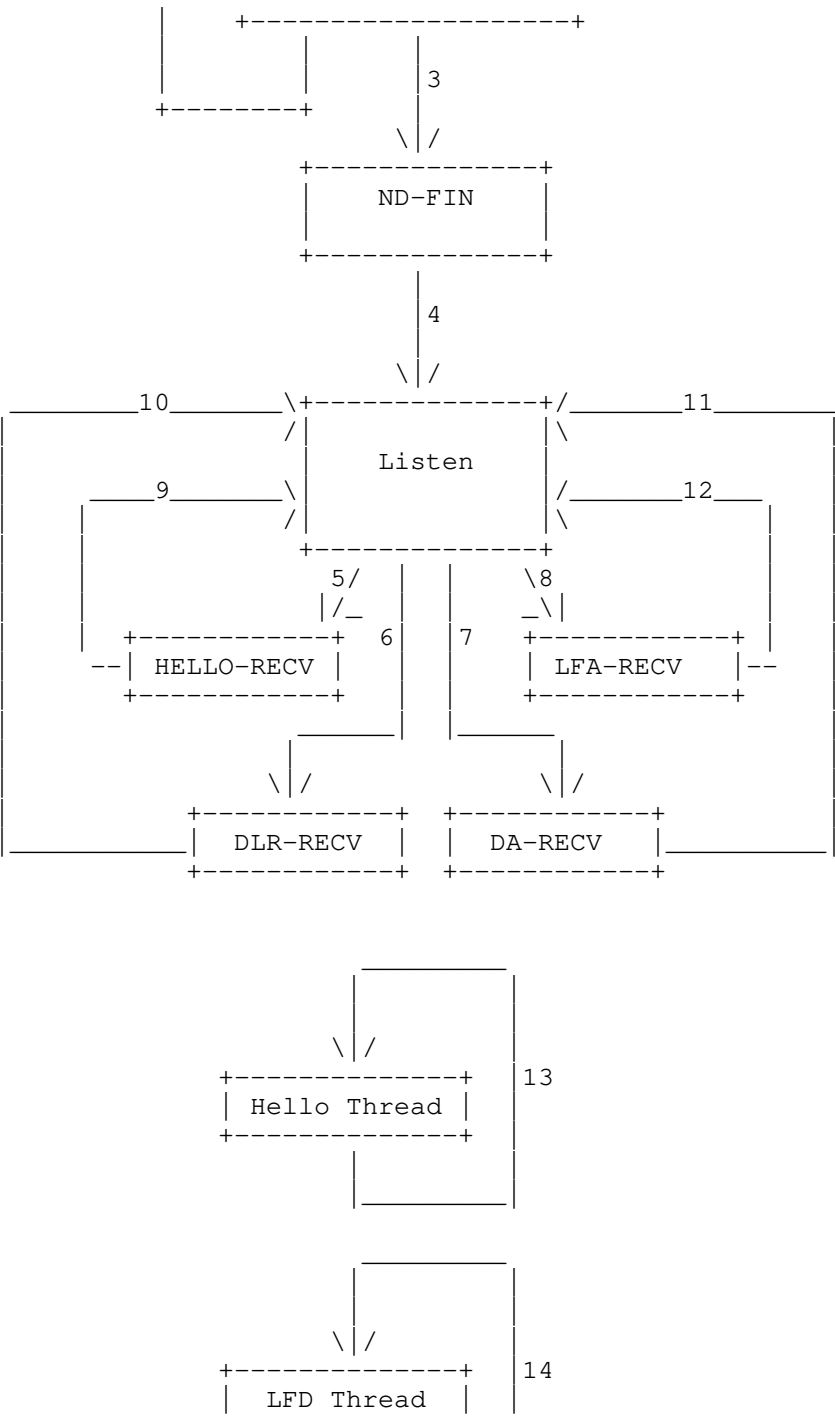
6.7. Routing Table Lookup (M7)

M7 is responsible for querying routing tables and selecting the next hop for forwarding the packets. Firstly, M7 takes the destination address of a forwarding packet as a criterion to look up route entries in a BRT based on longest prefix match. All of the matched entries are composed of a candidate hops list. Secondly, M7 look up negative route entries in a NRT taking the destination address of the forwarding packet as criteria. This lookup is not limited to the longest prefix match, any entry that matches the criteria would be selected and composed of an avoiding hops list. Thirdly, the candidate hops minus avoiding hops are composed of an applicable hops list. At last, M7 sends the forwarding packet to any one of the applicable hops. If the applicable list is empty, the forwarding packet will be dropped.

7. How a FAR Router Works

Figure 7 shows how a FAR router works by its FSM.





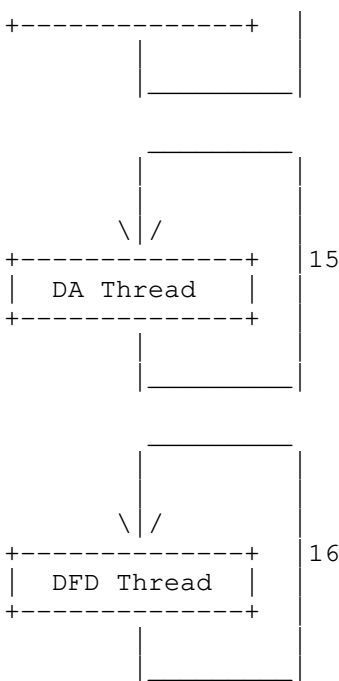


Figure 7: The Finite State Machine of FAR Router

- 1) When a router starts up, it starts a Hello thread and then starts ND (neighbor detection) timer (3 seconds). Next the router goes into ND (neighbor detection) state.
- 2) In the ND state, if a router received a Hello message, then it performs a Hello-message processing and goes back to the ND state.
- 3) When the ND timer is over, a router goes into ND-FIN (neighbor detection finished) state.
- 4) A router starts the LFD (link failure detection) thread and DFD (device failure detection) state, and sends DA message and DLR message to all of its active ports. Then the router goes into Listen state.
- 5) If a router receives a Hello message, then goes into HELLO-RECV state.
- 6) If a router receives a DLR message, then goes into DLR-RECV state.
- 7) If a router receives a DA message, then goes into DA-RECV state.
- 8) If a router receives a LFA message, then goes into LFA-RECV state.
- 9) A router performs the Hello-message processing. After that, it goes back to Listen state.
- 10) A router performs the DLR-message processing. After that, it goes back to Listen state.
- 11) A router performs the DA-message processing. After that, it goes back to Listen state.

- 12) A router performs the LFA-message processing. After that, it goes back to Listen state.
- 13) Hello thread produces and sends Hello messages to all its ports periodically.
- 14) LFD thread calls link-failure-detection processing to check link failures in all links periodically
- 15) DA thread produces and sends DA messages periodically (30 minutes).
- 16) When DFD thread starts up, it sleep a short time (30 seconds) to wait for a router learning all the active routers in the network. Then the thread calls the device-failure-detection processing to check device failures periodically (30 minutes).

8. Compatible Architecture

As a generic routing protocol, FAR can be run in various DCNs with regular topology. Up to now, we have implemented the FAR protocol for 4 types of DCN, including Fat-tree, BCube, MatrixDCN and Diamond.

For different network architectures, most processing of FAR is same besides calculation of routing tables. BRT routing tables are calculated based on Hello messages and NRT routing tables are calculated based on LFA messages in FAR. To extend FAR to support a new network architecture, only processing of Hello and LFA messages need providing to build BRT and NRT routing tables.

In this protocol, FAR can support maximally 12 network architectures and at least support 1 built-in network architecture, such as Fat-tree, BCube and MatrixDCN, etc. Each network architecture is assigned a unique number from 1 to 12. For example, if the 1 built-in architectures are assigned 1, and other customized architectures are assigned 2 to 12.

- 1: Fat-tree
- 2: BCube
- 3: MatrixDCN.
- 4: xxx.
-
- 12: xxx.

9. Topology identification and broadcast storm suppression

In this design, the initial topology discovery process is not a mandatory option for a FAR routing protocol. The recommended solution here is to use a pre-configured configuration file, which contains topology parameters of the current system, each node device as long as according to these configuration parameters will be able to know the topology information. In this way, we do not have to deal with complex topology discovery processes, nor do we need to

calculate the shortest path, because the optimal path can be calculated from the parameters. This protocol also allows the formation of configuration files to be submitted to the topology discovery protocol, allowing for a variety of different implementation options.

Regarding the flood suppression processing of broadcast packets, it has been considered in the previous content. Since the hello packets is only transmitted between the two nodes, it cannot be spread out. The link error message is only sent to the CPU, and are not forwarded to the nodes in layer 2 broadcasting. Moreover, each node will discard the repeated error messages when the node receives them. In this way, the broadcast storm can be suppressed. If a link is unstable and repeatedly up or down, the system will not send new messages after sending notifications, and the system will not oscillate repeatedly. The topology is updated only when the link is later detected to be stable for a long time.

10. Application Example

In this section, we take a Fat-tree network(Fig. 7) as an example to describe how to apply FAR routing. Since M1 to M4 are very simple, we only introduce how the modules M5, M6, and M7 work in a Fat-tree network.

A Fat-tree network is composed of 4 layers. The top layer is core layer, and the other layers are aggregation layer, edge layer and server layer. There are k pods, each one containing two layers of $k/2$ switches. Each k -port switch in the edge layer is directly connected to $k/2$ hosts. The remaining $k/2$ ports are connected to $k/2$ of the k -port switches in the aggregation layer. There are $(k/2)^2$ k -port core switches. Each core switch has one port connected to each of the k pods.

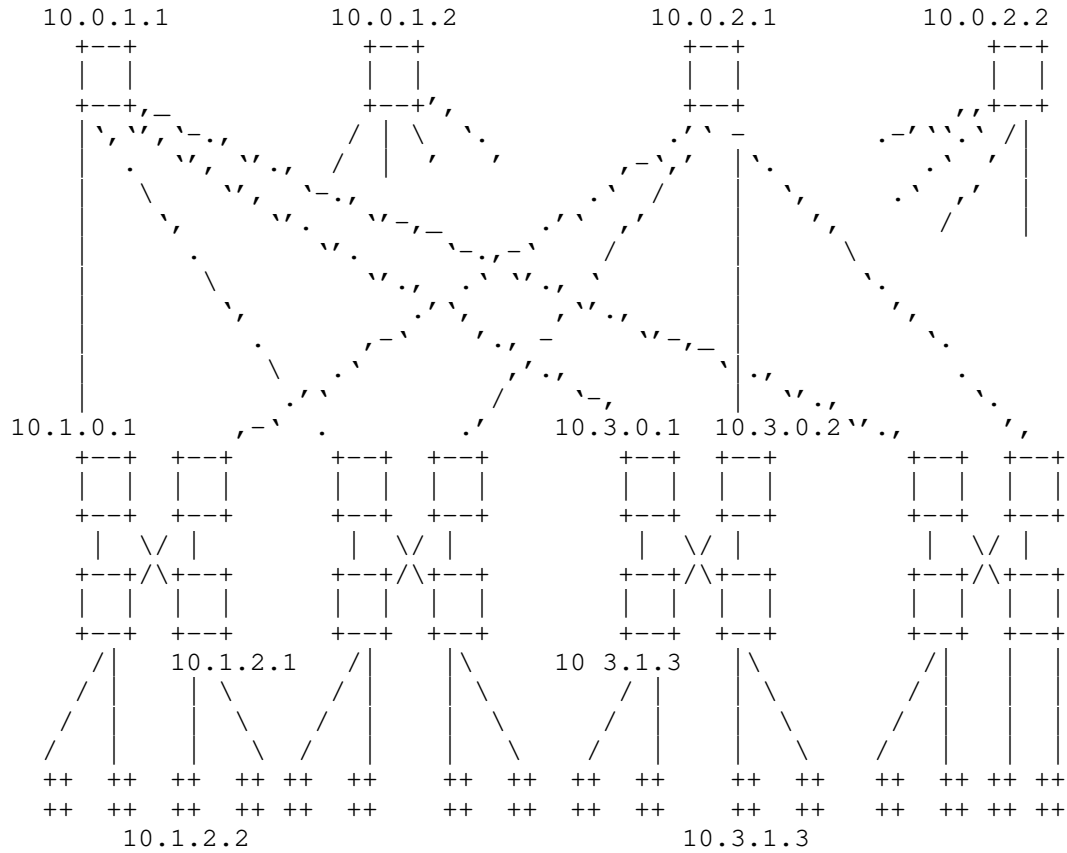


Figure 8: Fat-tree Network

Aggregation switches are given addresses of the form 10.pod.0.switch, where pod denotes the pod number, and switch denotes the position of that switch in the upper pod (in $[1, k/2]$). Edge switches are given addresses of the form 10.pod.switch.1, where pod denotes the pod number, and switch denotes the position of that switch in the lower pod (in $[1, k/2]$). The core switches are given addresses of the form 10.0.j.i, where j and i denote that switch's coordinates in the $(k/2)^2$ core switch grid (each in $[1, (k/2)]$, starting from top-left). The address of a host follows the pod switch to which it is connected to; hosts have addresses of the form: 10.pod.switch.ID, where ID is the host's position in that subnet (in $[2, k/2+1]$, starting from left to the right).

10.1. BRT Building Procedure

By leveraging the topology's regularity, every switch clearly knows how it forwards a packet. When a packet arrives at an edge switch, if the destination of the packet lies in the same subnet with the switch, then the switch directly forwards the packet to the destination server through layer-2 switching. Otherwise, the switch forwards the packet to any of aggregation switches in the same pod. When a packet arrives at an aggregation switch, if the destination of the packet lies in the same pod, the switch forwards the packet to the corresponding edge switch. Otherwise, the switch forwards the packet to any of core switches that it is connected to. If a core switch receives a packet, it forwards the packet to the corresponding aggregation switch that lies in the destination pod.

The forwarding policy discussed above is easily expressed through a BRT. The BRT of an edge switch, such as 10.1.1.1, is composed of the following entries:

Destination/Mask	Next hop
10.0.0.0/255.0.0.0	10.1.0.1
10.0.0.0/255.0.0.0	10.1.0.2

The BRT of an aggregation switch, such as 10.1.0.1, is composed of the following entries:

Destination/Mask	Next hop
10.1.1.0/255.255.0	10.1.1.1
10.1.2.0/255.255.255.0	10.1.2.1
10.0.0.0/255.0.0.0	10.0.1.1
10.0.0.0/255.0.0.0	10.0.1.2

The BRT of a core switch, such as 10.0.1.1, is composed of the following entries:

Destination/Mask	Next hop
10.1.0.0/255.255.0.0	10.1.0.1
10.2.0.0/255.255.0.0	10.2.0.1
10.3.0.0/255.255.0.0	10.3.0.1
10.4.0.0/255.255.0.0	10.4.0.1

10.2. NRT Building Procedure

The route entries in an NRT are related with link and node failures. We summarize all types of cases into three (3) catalogs.

10.2.1. Single Link Failure

In Fat-tree, Links can be classified as 3 types by their locations: 1) servers to edge switches; 2) edge to aggregation switches; 3) aggregation to core switches. Link failures between servers to edge switches only affect the communication of the corresponding servers and don't affect the routing tables of any switch, so we only discuss the second and third type of links failures.

Edge to Aggregation Switches

Suppose that the link between an edge switch, such as 10.1.2.1 (A), and an aggregation switch, such as 10.1.0.1(B), fails. This link failure may affect 3 types of communications.

- o Sources lie in the same subnet with A, and destinations do not. In this case, the link failure will only affect the routing tables of A. As this link is attached to A directly, A only needs to delete the route entries whose next hop is B in its BRT and add no entries to its NRT when A's M6 module detect the link failure.

- o Destinations lie in the same subnet with A, and sources lie in another subnet of the same pod. In this case, the link failure will affect the routing tables of all the edge switches in the same pod except for A. When an edge switch, such as 10.1.1.1, learns the link failure, it will add a route entry to its NRT:

Destination/Mask	Next hop
10.1.2.0/255.255.255.0	10.1.0.1

- o Destinations lie in the same subnet with A, sources lie in another pod. In this case, the link failure will affect the routing tables of all the edge switches in the other pods. When an edge switch in one other pod, such as 10.3.1.1, learns the link failure, because all the routings that pass through 10.3.0.1 to A will certainly pass through the link between A and B, 10.3.1.1 need add a route entry to its NRT:

Destination/Mask	Next hop
10.1.2.0/255.255.255.0	10.3.0.1

Aggregation to Core Switches

Suppose that the link between an aggregation switch, such as 10.1.0.1 (A), and a core switch, such as 10.0.1.2(B), fails. This link failure may affect 2 types of communications.

- o Sources lie in the same pod (pod 1) with A, and destinations lie in the other pods. In this case, the link failure will only affect the routing tables of A. As this link is attached to A directly, A only need to delete the route entries whose next hop is B in its BRT and add no entries to its NRT when A's M6 module detect the link failure.

- o Destinations lie in the same pod (pod 1) with A, and sources lie in another pod. In this case, the link failure will affect the routing tables of all the aggregation switches in other pods except for pod 1. When an aggregation switch in one other pod, such as 10.3.0.1, learns the link failure, because all the routings that pass through 10.0.1.2 to the pod 1 where A lies will certainly pass through the link between A and B, 10.3.0.1 need add a route entry to its NRT:

Destination/Mask	Next hop
10.1.0.0/255.255.0.0	10.0.1.2

10.2.2. A Group of Link Failures

If all the uplinks of an aggregation switch fail, then this switch cannot forward packets, which will affect the routing of every edge switches. Suppose that all the uplinks of the node A (10.1.0.1) fail, it will affect two types of communications.

- o Sources lie in the same pod (pod 1) with A, and destinations lie in the other pods. In this case, the link failures will affect the routing of the edge switches in the Pod of A. To avoid the node A, each edge switch should remove the route entry "10.0.0.0/255.0.0.0 10.1.0.1" in which the next hop is the node A.

- o Destinations lie in the same pod (pod 1) with A, and sources lie in other pods. In this case, the link failures will affect the routing of edge switches in other pods. For example, if the edge switch 10.3.1.1 communicates with some node in the pod of A, it should avoid the node 10.3.0.1, because any communication through 10.3.0.1 to the pod of A will pass through the node A. So a route entry should be added to 10.3.1.1:

Destination/Mask	Next hop
10.1.0.0/255.255.0.0	10.3.0.1

10.2.3. Node Failures

At last, we discuss the effect of node failures to a NRT. There are 3 types of node failures: the failure of edge, aggregation and core switches.

- o An edge switch fails. The failure doesn't affect the routing table of any switch.

- o A core switch fails. Only when all the core switches connected to the same aggregation switch fail, they will affect the routing of other switches. This case is equal to the case that all the uplinks of an aggregation switch fail, so the process of link failures can cover it.

- o An aggregation switch fails. This case is similar to the case that all the uplinks of an aggregation switch fail. It affects the routing of edge switches in other pods, but doesn't affect the routing of edge switches in pod of the failed switch. The process of this failure is same to the second case in section 6.2.2.

10.3. Routing Procedure

FAR decides a routing by looking up its BRT and NRT. We illuminate the routing procedure by an example. In this example, we suppose that the link between 10.3.1.1 and 10.3.0.2 and the link between 10.1.2.1 and 10.1.0.2 have failed. Then we look into the routing procedure of a communication from 10.3.1.3 (source) to 10.1.2.2 (destination).

Step 1: The source 10.3.1.3 sends packets to its default router 10.3.1.1

Step 2: The routing of 10.3.1.1.

1) Calculate candidate hops

10.3.1.1 looks up its BRT and gets the following matched entries:

Destination/Mask	Next hop
10.0.0.0/255.0.0.0	10.3.0.1

So the candidate hops = {10.3.0.1}

2) Calculate avoiding hops

Its NRT is empty, so the set of avoiding hop is empty too.

3) Calculate applicable hops

The applicable hops are candidate hops minus avoiding hops, so:

The applicable hops = {10.3.0.1}

4) Forward packets to 10.3.0.1

Step 3: The routing of 10.3.0.1

1) Calculate candidate hops.

10.3. 0.1 looks up its BRT and gets the following matched entries:

Destination/Mask	Next hop
10.1.0.0/255.255.0.0	10.0.1.1
10.1.0.0/255.255.0.0	10.0.1.2

So the candidate hops = {10.0.1.1, 10.0.1.2}

2) Calculate avoiding hops

Destination/Mask	Next hop
10.1.0.0/255.255.0.0	10.0.1.2

So the avoiding hops = {10.0.1.2}

3) Calculate applicable hops

The applicable hops are candidate hops minus avoiding hops, so:

The applicable hops = {10.0.1.1}

4) Forward packets to 10.0.1.1

Step 4: 10.0.1.1 forwards packets to 10.1.0.1 by looking up its routing tables.

Step 5: 10.1.0.1 forwards packets to 10.1.2.1 by looking up its routing tables.

Step 6: 10.1.2.1 forwards packets to the destination 10.1.2.2 by layer-2 switching.

10.4. FAR's Performance in Large-scale Networks

FAR has good performance to support large-scale networks. In this section, we take a Fat-tree network composed of 2,880 48-port switches and 27,648 servers as an example to show FAR's performance.

10.4.1. The number of control messages required by FAR

FAR exchanges a few messages between routers and only consumes a little network bandwidth. Tab. 1 shows the required messages in the example Fat-tree network.

Table 1: Required messages in a Fat-tree network.

Message Type	Scope	size(bytes)	Rate	Bandwidth
Hello	adjacent switches	less than 48	10 messages/sec	less than 4 kbps
DLR	adjacent switches	less than 48	(1)	48bytes
DA	entire network	less than 48	(2)	1.106M
LFA	entire network	less than 48	(3)	48 bytes

(1) Produce one when a router starts

(2) The number of switches(2,880) in a period

(3) Produce one when a link fails or recovers

10.4.2. The Calculating Time of Routing Tables

A BRT is calculated according to the states of its neighbor routers and attached links. An NRT is calculated according to device and link failures in the entire network. So FAR does not calculate network topology and has no problem of network convergence, which greatly reduces the calculating time of routing tables. The detection and spread time of link failures is very short in FAR. Detection time is up to the interval of sending Hello message. In FAR, the interval is set to 100ms, and a link failure will be detected in 200ms. The spread time between any pair of routers is less than 200ms. If a link fails in a data center network, FAR can detect it, spread it to all the routers, and calculate routing tables in no more than 500ms.

10.4.3. The Size of Routing Tables

For the test Fat-tree network, the sizes of BRTs and NRTs are shown in Tab. 2.

Table 2: The size of routing tables in FAR

Routing Table	Core Switch	Aggregation Switch	Edge Switch
BRT	48	48	24
NRT	0	14	333

The BRT's size at a switch is determined by the number of its neighbor switches. In the example network, a core switch has 48 neighbor switches (aggregation switch), so it has 48 entries in its BRT. Only aggregation and edge switches have NRTs. The NRT size at a switch is related to the number of link failures in the network. Suppose that there are 1000 link failures in the example network, the number of failed links is 1.2% of total links, which is a very high failure ratio. We suppose that link failures are uniformly distributed in the entire network. The NRT size at an edge switch is about 333 and the NRT size of an aggregation switch is about 14 in average.

11. Implementations Examples

In the FAR draft scenario, Fat-Tree topology has only three layers of routers. To expand the network scale is achieved through horizontal expansion: increase the number of core switches, and increase the number of aggregation switches and edge switches in pod.

For example, the following two scenarios.

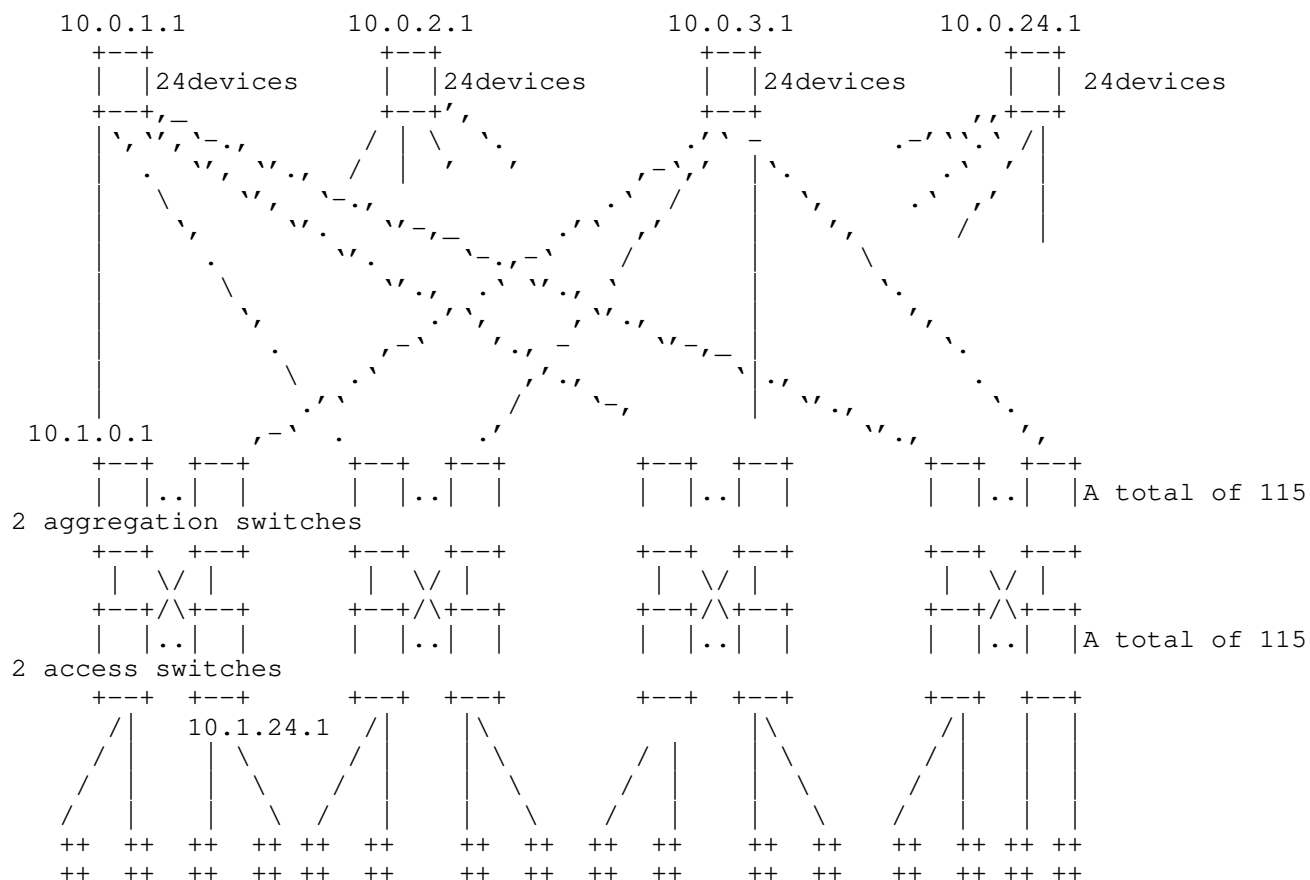


Figure 9: 48 pods, each of which has 24 aggregation switches and 24 access switches

In the Fat-tree network of Figure 9, there are a total of 48 pods, each of which has 24 aggregation switches and 24 access switches, and each access switch is connected to 24 servers. 576 core switches, 1152 aggregation switches, and 1152 access switches are required, for a total of 2880 switches, which can accommodate 27,648 servers.

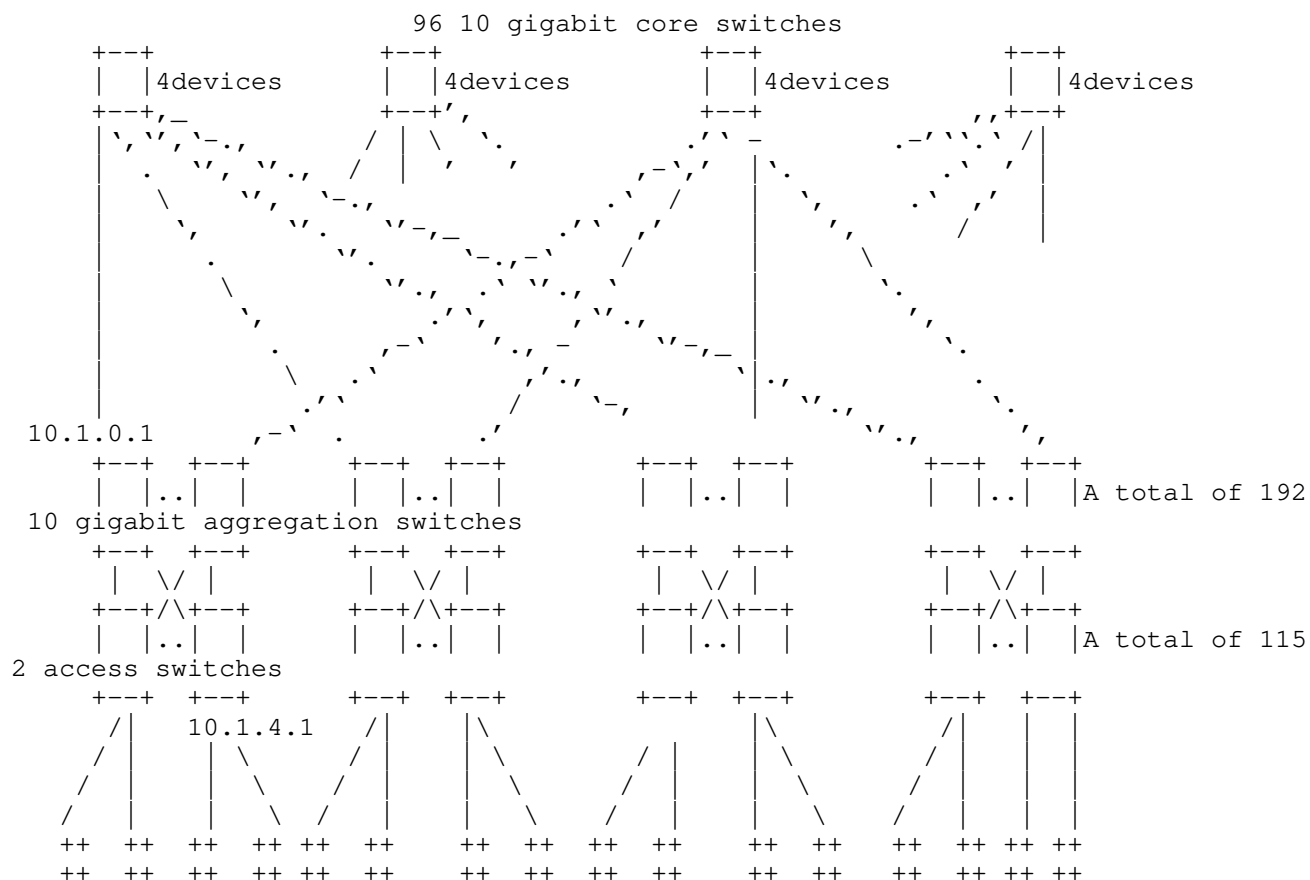


Figure 10: 48 pods. Each pod has 4 10G aggregation switches and 24 Gigabit access switches

In the Fat-Tree network of Figure 10, there are a total of 48 pods. Each pod has 4 10G aggregation switches and 24 Gigabit access switches. Each access switch is connected to 40 servers. Requires 96 core switches, 192 aggregation switches, and 1152 access switches, for a total of 1,440 switches, which can accommodate 46,080 servers.

12. Security Considerations

The security considerations will be discussed in a future version of this document.

13. Conclusions

This draft introduces FAR protocol, a generic routing method and protocol, for data centers that have a regular topology. It uses two routing tables, a BRT and an NRT, to store the normal routing paths and the forbidden (to-be-avoided) routing paths, respectively. This makes the FAR protocol very simple and efficient. The sizes of these two tables are very small. Usually, a BRT has only several tens of entries and an NRT has only several or about a dozen entries.

14. Acknowledgments

This document is supported by ZTE Enterprise-University-Research Joint Project.

15. References

15.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.

[RFC2328] J. Moy, "OSPF Version 2", BCP 14, RFC2328, April 1998.

[RFC3619] SHAH, S.; YIP, M. RFC3619: Extreme Networks' Ethernet Automatic Protection Switching (EAPS) Version 1. 2003.

15.2. Informative References

[FAT-TREE] M. Al-Fares, A. Loukissas, and A. Vahdat. "A Scalable, Commodity, Data Center Network Architecture", In ACM SIGCOMM 2008.

[BCube] Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., ... Lu, S. (2009, August). BCube: a high performance, server-centric network architecture for modular data centers. In Proceedings of the ACM SIGCOMM 2009 conference on Data communication (pp. 63-74).

[MatrixDCN] Sun, Y., Chen, M., Peng, L., Hassan, M. M., Alelaiwi, A. (2016). MatrixDCN: a high performance network architecture for large-scale cloud data centers. Wireless Communications and Mobile Computing, 16(8), 942-959.

16. Appendix

16.1. Application Area of the Solution

According to the horizontal expansion mode of the above scenarios, the whole Fat-Tree network does not need to be expanded to 4 layers (4 order Fat-Tree) even if it is expanded. Using a standard three-tier Fat-tree network, we can scale the network to meet all the problems of commercial network applications. This scheme is suitable for non-SDN distributed Fat-Tree network architecture.

16.2. Technical evolution roadmap

In this draft, we should design different rules for FAR switches in different regular networks to calculate routing tables, which limits FAR's extensibility. Fortunately, the latest SDN technology make it is easy to update the control plane of switches, since all the function of control plane are centralized to a controller in SDN. We are designing the next generation routing scheme for regular networks based on SDN. In the new scheme, we design a regular ToPoogy Description Language (TPDL) to descript a regular network. In TPDL, the distance between different type of node groups is defined by a group of distance formulas and the number of formulas is finite and fixed without increasing by the scale of a network. And then, switches learn the topology of a network by taking advantage of TPDL and generate flow table entries to forward packets without help of the SDN controller. If no entry is found for a forwarding packet, switches transport the packet to the SDN controller, and the controller recalculates a new routing path using A* algorithm by taking TPDL's distance formulas as a heuristic function and dispatch flow table entries down to related switches on the path.

16.3. Updating roadmap

In the next version, we will continue to advance the remaining issues such as protocol fields, section 3.2 simplification, etc.

Authors' Addresses

Bin Liu
ZTE Inc., ZTE Plaza
No.19 East Huayuan Road,Hai Dian District
Beijing 100191
China

Phone: +86 -010-59932039
Email: 13683610386@139.com

Yantao Sun
Beijing Jiaotong University
No.3 Shang Yuan Cun, Hai Dian District
Beijing 100044
China

Email: ytsun@bjtu.edu.cn

Jing Cheng
Beijing Jiaotong University
No.3 Shang Yuan Cun, Hai Dian District
Beijing 100044
China

Email: journey.j@gmail.com

Yichen Zhang
Beijing Jiaotong University
No.3 Shang Yuan Cun, Hai Dian District
Beijing 100044
China

Email: snowfall_dan@sina.com

Bhumip Khasnabish
Individual contributor
55 Madison Avenue, Suite 160
Morristown, New Jersey 07960
USA

Phone: +001-781-752-8003
Email: vumipl@gmail.com
URI: <http://tinyurl.com/bhumip/>

Operations and Management Area Working Group
Internet-Draft
Intended status: Informational
Expires: January 5, 2015

Q. Wu
M. Wexler
Huawei
M. Boucadair
France Telecom
S. Aldrin
Huawei USA
G. Mirsky
Ericsson
P. Jain
Nuage Networks
July 4, 2014

Problem Statement and Architecture for Transport-Independent Multiple
Layer OAM
draft-ww-opsawg-multi-layer-oam-02.txt

Abstract

Operations, Administration, and Maintenance (OAM) mechanisms are critical building blocks in network operations that are used for service assurance, fulfillment, or service diagnosis, troubleshooting, and repair. The current practice is that many technologies rely on their own OAM protocols that are exclusive to a given layer. There is little consolidation of OAM in either data plane or management plane nor well-documented inter-layer OAM operations. Vendors and Operators dedicate significant resources and effort through the whole OAM life-cycle each time when a new technology is (to be) introduced. This is even exacerbated when dealing with integration of OAM across multiple technologies.

This document describes the problem space and defines an architecture for the generic and integrated OAM with a focus of multi-layer and cross-layer considerations.

Status of This Memo

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

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 5, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
2. Terminology	4
2.1. Acronyms and Abbreviations	6
3. Problem Statement	6
3.1. Use of Existing Protocols	7
3.2. Strong Technology dependency	8
3.3. Weakness of Cross-Layer OAM	8
3.4. Lack of OAM above Layer 3	9
3.5. Issues of Abstraction	9
3.6. Issue of OAM Information Gathering from Layers Covering Heterogeneous Network Technologies	10
3.6.1. Focus on Service Function Chaining	10
4. Architecture Overview	11
5. Existing Work	13
6. Architectural Consideration	14
6.1. Basic Components	14
6.1.1. Overlay OAM	14
6.1.2. OAM at the top of Layer 3	14
6.2. OAM Functions in the Data Plane	14
6.2.1. Continuity Check	14
6.2.2. Connectivity Verification	14
6.2.3. Path Discovery	14
6.2.4. Performance Measurement	14
6.2.5. Protection Switching Coordination	15
6.2.6. Alarm/defect Indication	15
6.2.7. Maintenance Commands	15

6.3. OAM in Management Plane	15
7. Building on Existing Protocols	16
8. Scoping Future Work	16
9. Manageability Considerations	17
10. Security Considerations	17
11. Acknowledgements	17
12. References	17
12.1. Normative References	17
12.2. Informative References	17
Authors' Addresses	19

1. Introduction

Operations, Administration, and Maintenance (OAM) mechanisms being understood and used in context of RFC 6291 [RFC6291] are critical building blocks in network operations that are used for service assurance, fulfillment, or service diagnosis, troubleshooting, and repair. The key foundations of OAM and its functional roles in monitoring and diagnosing the behavior of networks have been studied at OSI layers 1, 2 and 3 since a while. As a reminder, OAM functions are used in many management applications for various objectives such as (i) failure detection, (ii) reporting the defect/ failure information, (iii) defect/failure localization, (iv) performance monitoring, and (v) service recovery.

The current practice that consists in enabling OAM techniques for each layer has shown its limits; this is a need for cross-layer and inter-layer OAM considerations [RFC7276]. This need for inter-layer OAM is motivated by the need to achieve: network optimization, efficient enforcement of TE (Traffic Engineering) techniques including ensuring path diversity at distinct layers or computing completely disjoint paths at several layers, fine-grain tweaking, ease of root cause analysis, ability to maintain a network-wise visibility in addition to layer-specific one, etc.

It is worth to mention also that there are two restrictions for multi-layer structure as discussed in [RFC7276]:

- o Each layer has its own OAM protocol, OAM should not cross layer boundaries.
- o Each layer OAM used at different level of hierarchy in the network.

Moreover, there is little consolidation of OAM in either data plane or management plane. Vendors and operators dedicate a lot resources and effort through the whole OAM life-cycle each time a new technology is (to be) introduced. Integration of OAM across multiple

technologies in either data plane or management plane is extremely difficult to achieve.

When operating networks with more than one technology, maintenance and troubleshooting are achieved per technology and per layer, operation process can be very cumbersome since OAM is not defined to cross layer boundaries. Another challenge is presented by use of different technologies and corresponding OAM on the same layer of adjacent network domains. Interworking between different OAM often not defined and are left to proprietary solutions. In many cases when keeping network complexity down and simplifying OAM is needed, it is desirable to have a generic and integrated OAM to cover heterogeneous networking technologies.

This document defines the problem space and describes an architecture for the generic and integrated OAM in the multi-layer and multi-domain networks. In particular, it outlines the problems encountered with existing OAM protocols and their impact on introduction of new technologies (see Section 3).

This document covers the following:

- o Data plane OAM consolidation by looking at the common active OAM functions (including, Connectivity Verification (CV), Path Verification and Continuity Checks (CC), Path Discovery, Performance Measurement) necessary to monitor and diagnose a network;
- o Management plane consolidation by interacting with data plane OAM and abstracting OAM information common to different layer via uniformed interface.

2. Terminology

This document defines the following terms:

Transport Independent Multi-Layer OAM:

In an multi-layer network, transport independent OAM is OAM that can be deployed independent of media, data protocols, and routing protocols. It denotes the ability to exchange OAM information across layers and domains between nodes along forwarding path, and gather OAM information that are common to different layers and expose it to the management application through a unified interface. These aspects are not specific to a given transport technology.

OAM function:

Refers to the atomic building blocks of OAM; an OAM function defines an OAM capability (See section 2.2.3 of [RFC7276]).

OAM protocol:

Refers to a protocol used for implementing one or more OAM functions (See section 2.2.3 of [RFC7276]).

OAM tool:

Denotes a specific means of applying one or more OAM functions. An OAM protocol can be an OAM tool. An OAM tool can use a set of OAM protocols or a set of protocols that are not strictly OAM related (See section 2.2.3 of [RFC7276]).

OAM packet:

Refers to a packet generated at Maintenance Point using an OAM protocol. An OAM packet, which carries OAM information, is usually forwarded through the same route/path as the data traffic and receive the same (forwarding) treatment.

Maintenance Domain (MD):

Refers to the part of a network where OAM function is performed (initiated).

Maintenance Point (MP):

Is a generic functional entity that is associated with a particular MD, defined at a specific layer of a network and can initiate and/or react to OAM packets.

Maintenance Endpoint (MEP):

Is an endpoint MP that initiates OAM packets and responds to them.

Maintenance Intermediary Point(MIP):

In between MEPs, there are zero or more intermediate points, called Maintenance Intermediary Point. A Maintenance Intermediary Point (MIP) is an intermediate MP that does not generally initiate OAM packets but is able to respond to OAM packets that are destined to it.

Maintenance Association (MA):

The relationship between a set of MEPs to which maintenance and monitoring operations apply.

Network Element (NE):

Denotes a physical or virtual network device/function that connects directly to the network. NE can host MPs and provide network connectivity to one or many MPs.

2.1. Acronyms and Abbreviations

CC - Continuity Check

CV - Connectivity Verification

SNMP - Simple Network Management Protocol

NETCONF - Network Configuration

ETH - Ethernet

APS - Automatic Protection Switching

LT - LinkTrace

RDI - Remote Defect Indication

AIS - Alarm indication Signal

OWAMP - One Way Active Measurement Protocol

TWAMP - Two Way Active Measurement Protocol

CFM - Connectivity Fault Management

3. Problem Statement

OAM mechanisms are usually oriented toward a single network technology or a single layer. Each technology or layer has its best suited OAM tools. Some of them providing rich functionality rely on the capabilities of one protocol, while the others provide each function with a different protocol; In the current situation, there is little, or no re-use, of software and hardware for each OAM protocol.

Integration of OAM across multiple technologies is extremely difficult. Vendors and operators waste a lot through the whole OAM life-cycle when a new technology is introduced:

(1) Design and development: For every new protocol there is a need to invest in complete life-cycle (i.e., the design and development of data, control and management planes). In some cases, even adding a single OAM function requires the above complete life-cycle.

(2) Operation and Maintenance: There is a need to re-train operation people for almost every newly introduced technology or feature. The above causes a slow time-to-market and a waste of time and effort for any new technology and/or OAM function.

Specifically, in Service Function Chaining environment, every Service Function may operate at a different layer and may use different encapsulation and tunneling techniques. When taking into account virtualization related technologies, the number of encapsulation and tunneling options increase even more. Still, end-to-end service OAM mechanisms and information exchanges between Service Functions should be provided to operate and maintain the network as a whole. This requires a generic toolkit that can provide all necessary tools in context of multi-technology, multi-layer, physical and virtual environments.

A particular problem is how OAM information at different layer is made available to a management application for use and learnt via the unified management interface. For example, in the case of an multi-layer network, OAM information needs to be imposed to the packet and injected into the network and at last abstracted from various layers and expose them to the management application.

3.1. Use of Existing Protocols

OAM information resides at each layer and may currently be exchanged at each network layer in a domain by using various encapsulation technologies at the Layer 2 & Layer 3 levels. OAM information may be gathered and exported from a domain (for example, northbound) using SNMP [RFC3411] or NETCONF/YANG [RFC6241].

It is desirable that a solution to the problem described in this document does not require the implementation of a new, network-wide protocol or introduce a shim layer to carry OAM information. Instead, it would be advantageous to make use of an existing protocols or functionalities that are commonly implemented and are currently deployed in operational networks. This has many benefits in network stability, time to deployment, and operator training.

It is recognized, however, that existing protocols or functionalities are unlikely to be immediately suitable to this problem space without some protocol extensions. Extending protocols must be done with care

and with consideration for the stability of existing deployments. In extreme cases, when there is a lack of functionality, although similar mechanisms exist in other technologies, a new protocol can be preferable to a "messy" hack of an existing protocol.

3.2. Strong Technology dependency

OAM protocols are relying heavily on the specific network technology they are associated with. For example, ICMP, LSP Ping are using different network technologies but provide the same OAM functionality, i.e., Path Discovery. Another example is BFD, LSP Ping are using different network technologies but provide the same functionality, i.e., Continuity Verification. Figure 1 shows common OAM functionalities shared by various existing IETF OAM protocols.

	Continuity Check	Connectivity Verification	Path Discovery	Performance Measurement
ICMP	Echo(Ping)		Traceroute	-Delay -Loss rough measurement
BFD	BFD Control /Echo	BFD Control		
LSP Ping		Ping	Traceroute	- Delay - Packet Loss
IPPM				-OWAMP -TWAMP
MPLS-TP OAM	CC (use of BFD)	CV (use of BFD) or LSP Ping)	Traceroute	-Delay -Packet Loss

Figure 1: Examples of IETF OAM tools

3.3. Weakness of Cross-Layer OAM

Troubleshooting is cumbersome due to protocol variety and lack of multi-layer OAM. Usually OAM messages should not cross layer boundaries. Each of the service, network and transport layers

possesses its well-discernible and native OAM stream. In addition, OAM messages should not be leaked outside of a management domain within a layer, where a management domain is governed by a single business organization. When having networks with more than one technology, maintenance and troubleshooting are done per technology and layer.

These rules could in some cases ease the understanding in which technology the operation is done or fault is located. In some cases, when one layer OAM fails, it may be desirable to drop down to the another layer OAM and issue the corresponding OAM command, using the same APIs, if OAM in multiple layers can be supported. However, in most cases switching tools and layers in the same operation process is cumbersome and not serving the main idea - to find the root cause location. It would be very helpful to have a generic mechanisms that is end to end basis, allow management application interact with data plane OAM and can ping IPv4 host by an IPv6 source or having one tool to troubleshoot combined IP, MPLS, Ethernet, GRE and VXLAN network.

In Service Function Chaining environment, it is necessary to provide end-to-end OAM across certain or all entities and involving many layers. Inter-layer OAM considerations are key in an SFC context because problems may occur at the network layer or at the service chaining layer.

3.4. Lack of OAM above Layer 3

The Layer 2/3 OAM protocols are quite rich in their functionality, well defined, standardized and heavily used. In the last years a lot of work was conducted to consider maintenance domains and levels in order to better handle the issues of technology re-use, smooth interoperability and interworking between domains.

The above mechanisms are not defined for the technologies above Layer 3. Therefore, in the SFC environment where a Service Function Chaining is composed by a set of Service Functions, but providing an end-to-end chain or path from a source to destination in a given order [I.D-ietf-sfc-problem-statement], no standard exists as a reference for OAM since when the service packets is steered through a set of service nodes distributed in the network, each service node may act at different layers above layer 3.

3.5. Issues of Abstraction

In multi-layer network, OAM functions are enabled at different layers and various OAM information needs to be gathered from various layers. Without multi-layer OAM in place, it is hard for management applications to understand what information at different layers

stands for. One possible solution to these issues is to abstract the OAM information shared across layers, i.e., using the same tool or API to activate the OAM functions at different layers and retrieve the results.

The challenge is to abstract in a way that retains as much useful information as possible while filtering the data that is not needed to be leaked to other layers. An important part of this effort is a clear understanding of what information is actually needed.

3.6. Issue of OAM Information Gathering from Layers Covering Heterogeneous Network Technologies

In SFC, the service packets are steered through a set of service nodes (virtual or physical) hosting the service function distributed in the network. In the NVO3 network, the data packet may also traverse a set of overlay nodes distributed in the network. Overlay technologies or other tunneling technologies can be used to stitch these service nodes or overlay node in order to form end to end path.

When any overlay Segment or segment of service chain in the network fails to deliver user traffic, there is a need to provide a tool that would enable users to detect such failures at different layer using various encapsulation protocols and locate faults in the specific part of the network, and a mechanism to isolate these faults. It may also be desirable to test the data path before mapping user traffic to the Overlay Segment or segment of service chain. When multiple layer OAMs are used in the different parts of the network; how these layers OAM interwork at the boundary of each part of network is also a serious issue.

3.6.1. Focus on Service Function Chaining

When the service packets are steered through a set of Service Nodes (virtual or physical) hosting the Service Function distributed in the network, each Service Node may work at different layer above layer 3 and may embed several SFs. When OAM mechanism is applied, it is necessary to allow OAM packets to be exchanged:

- o between Service Functions/Service Nodes and the SFC Management System,
- o between these Service Nodes,
- o between Service Functions at different layers,
- o or between Service Nodes and ingress node of the SFC-enabled domain.

When Service Functions that are part of the SFC-enabled domain do support the OAM capability (e.g., an SFC-unaware Service Function) and Service Node has OAM capability, Service Nodes may be responsible for monitoring and diagnosing and reporting service availability of these Service Functions. It is more desirable to allow Service Functions register with a Service Node. Either Service Functions report status to the Service Node or the Service Node performs liveness check of the Service Function.

In addition, some Service Functions may not have Layer 2-3 switching/routing capability and therefore are not aware of any OAM function at Layer 2-3. Also when there are no OAM functions at service Layers above layer 3, it is hard to identify the layer that can be used to gather OAM information when it comes to a fault situation or degradation of performance. For example, when a data packet is transmitted from SFC ingress node (i.e., Classifier) and traverse a set of Service Nodes that host Service Function, the data packet may be discarded either at the SFC ingress node, one specific Service Node or one specific Service Function. Also the data packet may be lost between SFC ingress and one Service Node, or between two Service Nodes, or between one Service Node and one Service Function, how to detect the fault between them and how to isolate problem to that layer?

Editor's Note: Section 3.6.1 is too specific. This text can be presented as an example to illustrate a problem not a problem per se or moved to a use case draft.

4. Architecture Overview

Figure 2 shows the reference architecture for Layering OAM. This reference architecture assumes that

- o Any network element can use different technologies and corresponding OAM on the same layer at the boundary of two adjacent domains
- o Any two network element may provide service delivery at different layer
- o Management entity can manage network devices in more than one maintenance domains.

In this architecture, three layers are defined:

M1: "Data Plane layer"

M2: "Management Plane layer"

M3: "Service Plane layer"

In the M1 layer, a typical network can be partitioned into several domains. Each domain has at least two MEPs and none or several MIPs. One domain can contain one or more maintenance associations (MAs). MEP is a maintenance functional entity that is implemented into a Network Element at the maintenance domain boundary and can send and receive OAM packets. MIP is a maintenance functional entity that is implemented into a Network Element in the maintenance domain and can forward OAM packets and respond OAM packets only when triggered by a specific OAM function (e.g., Path Discovery or Connectivity Verification). MEPs and MIPs can exist in the same maintenance domain and belong to different MAs. They can also exist at different layers and use various encapsulating protocols.

The M2 contains the interface which management entity uses to manage individual network devices. In this document, we further require management entities to use this interface as uniform interface (API and or UI) to gather OAM information from MEP and MIP in the network devices (either physical or virtual entity) and execute transactions or operations on MEP and MIP across domains, layers and vendors. Protocols that can be used to manipulate the configuration of a network device include SNMP [RFC1157], Command Line Interfaces, NETCONF [RFC6241], and other protocols.

On the M3 layer, there is a uniform interface (API and/or UI) that covers all the managed devices and can execute network-wide transactions. This layer allows applications and operators to execute configuration, monitoring and action tasks across multiple network devices, from a mix of domains, layers, vendors. Still the abstraction level is that of the network elements themselves, so whatever configuration, status, actions and notifications they provide, that is what you get here, but without having to worry about the location and the protocol to reach the device.

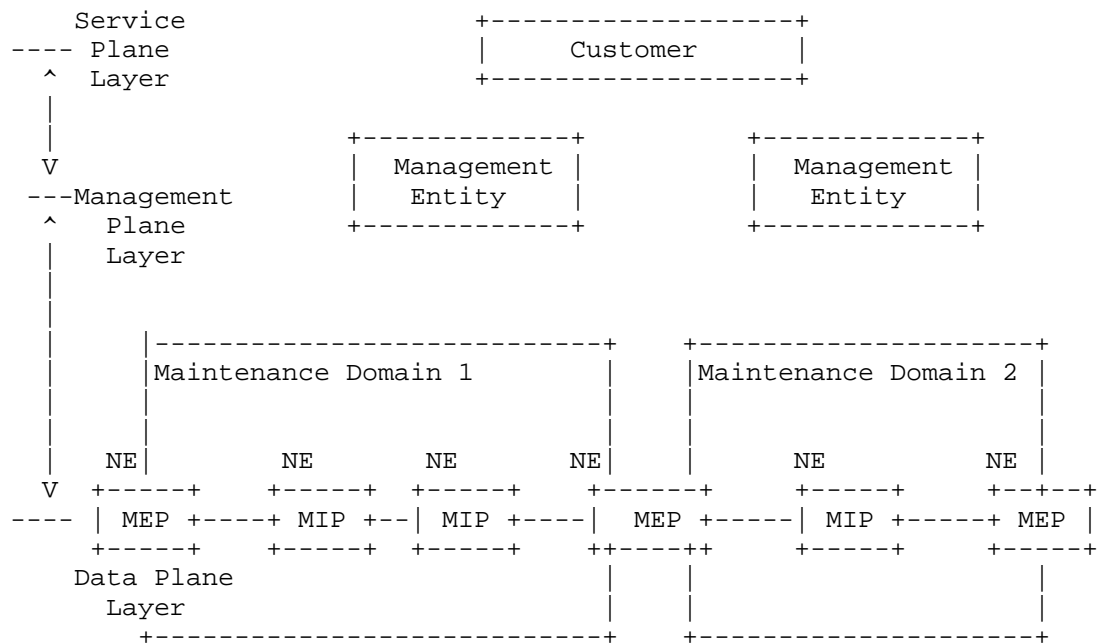


Figure 2: Architecture for Layering OAM in the management plane

An example of service-specific that depicts OAM layers can be found in [RFC4176] (L3VPN case).

5. Existing Work

The following discuss related IETF work and is provided for reference. This section is not exhaustive, rather it provides an overview of few initiatives focusing on the pain-points of OAM:

1. [I-D.tissa-netmod-oam] is an important work that creates a YANG unified data model for OAM that is based on IEEE CFM model. This model may be used also for IP OAM functionality. This effort is focused on the management plane of OAM and should be complemented by an accompanying data-plane and/or control-plane work. It may require also some extensions to address wider variety of functions and technologies.
2. Several contributions conducted in the past years, had tried to address new technologies using existing mechanisms. [I-D.jain-nvo3-overlay-oam] and MPLS-TP OAM documents are only examples for such efforts.

6. Architectural Consideration

6.1. Basic Components

6.1.1. Overlay OAM

6.1.2. OAM at the top of Layer 3

6.2. OAM Functions in the Data Plane

Many OAM functions may require protocol extensions or new protocol development to meet the transport requirements. In the existing OAM tools, Some of them providing rich functionality in one protocol, the other providing each function with a different protocol and each technology is developed independently.

To consolidate OAM in the data plane, the OAM in multi-layer Environment is expect to support the following common OAM functions used in OAM-related standards. These functions are used as building blocks in the data plane OAM standards described in this document.

6.2.1. Continuity Check

This type of mechanisms check that the monitored layer and/or entity are alive and providing path from specific point(s) to other point(s). Some examples are IP Ping, BFD [RFC5880] and ETH CC.

6.2.2. Connectivity Verification

Verifying that the actual connection is consistent with the required connection and no mis-connection occurred. Some examples are IP Ping, and ETH loopback.

6.2.3. Path Discovery

Used to discover the path that specific service traverses in the network. Some examples are LSP Traceroute, IP Traceroute and ETH-LT/linktrace.

6.2.4. Performance Measurement

A function that monitors the performance parameters of a network entity. Such parameters could be Delay, Delay-variation, loss, availability of services and class of services. Examples are TWAMP[RFC5357]/ OWAMP[RFC4656] and Y.1731, MPLS Loss and Delay Measurement [RFC6374].

6.2.5. Protection Switching Coordination

A function that is used to signal protection switching states and commands. Examples are ETH APS messages and MPLS-TP Protection Switching Coordination OAM [RFC6378].

6.2.6. Alarm/defect Indication

A function that is used to indicate that a failure occurred downstream or upstream within a connection/service. Used also to trigger fast protection or to suppress alarms. Examples are ETH AIS and ETH RDI, MPLS-TP RDI [RFC6428].

6.2.7. Maintenance Commands

A function that is used to signal a maintenance state or command within a connection/service. Examples can be ETH Lockout.

6.3. OAM in Management Plane

Management systems play an important role in configuring or provisioning OAM functionality consistently across all devices in the network, and for automating the monitoring and troubleshooting of network faults. However OAM is not provisioned. In general, provisioning is used to configure the network to provide new services, whereas OAM is used to keep the network in a state that it can support already existing services.

As we know each layer has its own OAM protocols. OAM can be used at different levels of hierarchy in the network to form a multi-layer OAM solution [RFC7276]. To support multi-layer OAM covering various heterogeneous transport technologies, the OAM in the management needs to be consolidated as follows:

- o OAM information needs to be abstracted that are common to different layer and different domain.
- o Support customized OAM service, e.g., customized service diagnose.
- o OAM information is provided to management entity from managed device via a uniform interface (API and/or UI)
- o Sets up MD MEP and MIP in the network provision phase
- o Enables basic OAM functionality(e.g., enable the origin of ping and trace packets or configure Connectivity Fault Management (CFM)) on the managed devices in the service activation phase.

The different OAM tools may be used in one of two basic types of activation:

- o Proactive activation - indicates that the tool is activated on a continual basis, where messages are sent periodically, and errors are detected when a certain number of expected messages are not received.
- o On-demand activation - indicates that the tool is activated "manually" to detect a specific anomaly.

7. Building on Existing Protocols

8. Scoping Future Work

This section includes a set of candidate items for activities to be conducted within IETF.

These objectives are not frozen; further discussion is required to target key issues and scope the work to be conducted within IETF accordingly.

Candidate investigation items are listed below:

- o Understand and discuss situations where an OAM protocol can be tuned and optimized for a specific data plane.
- o OAM consolidation in the data plane:
 - * Exchange OAM information at the service layer atop of layer 3.
 - * Deployed over various encapsulating protocols, and in various medium types
- o OAM consolidation in the management plane:
 - * Abstract OAM information common to different layers.
 - * Expose OAM information via unified interface to management entities, independently of the layer they belong to.
 - * Discuss how information gathered from various layers can be correlated for the sake of network operations optimization purposes.
 - * Propose means to help during service diagnosis; these means may rely on filtering information to be leaked to other layers so that time recovery can be optimized. A typical example would

be efficient root cause analysis that is fed with input from various layers.

- * Propose means that would help to optimize a network as a whole instead of the monolithic approach that is specific to a given layer. For example, investigate means that would help in computing diverse and completely disjoint paths, not only at layer 3 but also at the physical layer.

9. Manageability Considerations

10. Security Considerations

Security considerations are not addressed in this problem statement only document. Given the scope of OAM, and the implications on data and control planes, security considerations are clearly important and will be addressed in the specific protocol and deployment documents.

11. Acknowledgements

The authors would like to thank Romascanu, Dan, Tom Taylor, Tissa Senevirathne, Huub van Helvoort, Yuji Tochio for their valuable reviews and suggestions.

12. References

12.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", March 1997.
- [RFC6291] Andersson, L., Helvoort, H., Bonica, R., Romascanu, D., and S. Mansfield, "Guidelines for the Use of the "OAM" Acronym in the IETF", RFC 6291, June 2011.
- [RFC7276] Mizrahi, T. and N. Sprecher, "An Overview of Operations, Administration, and Maintenance (OAM) Tools", RFC 7276, June 2014.

12.2. Informative References

- [I-D.jain-nvo3-overlay-oam]
Jain, P., "Generic Overlay OAM and Datapath Failure Detection", ID draft-jain-nvo3-overlay-oam-01, February 2014.

- [I-D.tissa-netmod-oam]
Senevirathne , T., Finn, N., Kumar , D., and S. Salam ,
"YANG Data Model for Operations Administration and
Maintenance (OAM)", ID draft-tissa-netmod-oam-00, March
2014.
- [I-D.ietf-sfc-problem-statement]
Quinn, P., Guichard, J., and S. Surendra, "Network Service
Chaining Problem Statement", ID draft-ietf-sfc-problem-
statement, August 2013.
- [RFC3411] Harrington, D. and R. Presuhn, "An Architecture for
Describing Simple Network Management Protocol (SNMP)
Management Frameworks", RFC 3411, December 2002.
- [RFC4176] El Mghazli, Y., Nadeau, T., Boucadair, M., Chan, K., and
A. Gonguet, "Framework for Layer 3 Virtual Private
Networks (L3VPN) Operations and Management", RFC 4176,
October 2005.
- [RFC4656] Shalunov, S., Karp, A., Boote, J., and M. Zekauskas, "A
One-way Active Measurement Protocol (OWAMP)", RFC 4656,
September 2006.
- [RFC5357] Hedeyat, K., Krzanowski, R., Morton, A., Yum, K., and J.
Babiarz, "A Two-Way Active Measurement Protocol (TWAMP)",
RFC 5357, October 2008.
- [RFC5880] Katz, D. and D. Ward, "Bidirectional Forwarding Detection
(BFD)", RFC 5880, June 2010.
- [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A.
Bierman, "Network Configuration Protocol (NETCONF)", RFC
6241, June 2011.
- [RFC6374] Frost, D. and S. Bryant, "Packet Loss and Delay
Measurement for MPLS Networks", RFC 6374, September 2011.
- [RFC6378] Weingarten, Y., Bryant, S., Osborne, E., Sprecher, N., and
A. Fuligoli, "Packet Loss and Delay Measurement for MPLS
Networks", RFC 6378, October 2011.
- [RFC6428] Allan, D., Swallow, G., and J. Drake, "Proactive
Connectivity Verification, Continuity Check, and Remote
Defect Indication for the MPLS Transport Profile", RFC
6428, November 2011.

Authors' Addresses

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: bill.wu@huawei.com

Mishaël Wexler
Huawei
Riesstr. 25
Munich 80992
Germany

Email: mishaël.wexler@huawei.com

Mohamed Boucadair
France Telecom
Rennes 35000
France

Email: mohamed.boucadair@orange.com

Sam Aldrin
Huawei Technologies USA
2330 Central Expressway
Santa Clara, CA 95051
USA

Email: aldrin.ietf@gmail.com

Greg Mirsky
Ericsson

Email: gregory.mirsky@ericsson.com

Pradeep Jain
Nuage Networks
755 Ravendale Drive
Mountain View, CA 94043
USA

Email: pradeep@nuagenetworks.net