

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: May 3, 2018

J. Arkko
Ericsson
J. Tantsura
Futurewei, Future Networks
October 30, 2017

Low Latency Applications and the Internet Architecture
draft-arkko-arch-low-latency-02

Abstract

Some recent Internet technology developments relate to improvements in communications latency. For instance, improvements in radio communications or the recent work in IETF transport, security, and web protocols. There are also potential applications where latency would play a more significant role than it has traditionally been in the Internet communications. Modern networking systems offer many tools for building low-latency networks, from highly optimised individual protocol components to software controlled, virtualised and tailored network functions. This memo views the developments from a system viewpoint, and considers the potential future stresses that the strive for low-latency support for applications may bring.

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 May 3, 2018.

Copyright Notice

Copyright (c) 2017 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	2
2. Applications with Special Focus on Low Latency	3
3. Role of Low-Latency vs. Other Communications	4
4. Selected Improvements to Communications Latency	5
5. Architectural Considerations	6
5.1. Background	6
5.2. Implications	7
5.2.1. Service Distribution	7
5.2.2. Edge Computing	8
5.2.3. Routing and tunnels	8
5.2.4. Alternative Paths and Control Tension	9
5.2.5. Quality-of-Service	9
5.2.6. Cross-Layer Optimisations	10
5.3. Recommendations for Further Work	11
6. Acknowledgements	12
7. Informative References	12
Authors' Addresses	15

1. Introduction

Some recent Internet technology developments relate to improvements in communications latency. For instance, improvements in radio communications or the recent work in IETF transport, security, and web protocols.

There are also potential applications where latency would play a more significant role than it has traditionally been in the Internet communications.

New applications or technologies do not necessarily imply that latency should be the main driving concern, or that any further efforts are needed, beyond those already ongoing. Indeed, modern networking systems offer many tools for building low-latency networks, across the stack. At the IETF, for instance, there has been a recent increase in work related to transport, security, and web application protocols, in part to make significant improvements in latency and connection set-up times. Similar efforts for other

components of communications technology exist in 3GPP, IEEE, and other standards organisations.

Despite a large number of specific developments, it may be interesting to view the developments from a system viewpoint, and to consider the potential future stresses that the strive for low-latency support applications may bring.

The rest of this memo is organised as follows: Section 2 discusses potential applications for low-latency communications. Section 4 reviews some of the recent work across the stack, related to latency improvements. Finally, Section 5 discusses some of the implications (and non-implications) from an architectural perspective.

2. Applications with Special Focus on Low Latency

Most Internet applications enjoy significant benefits from low-latency communications in the form of faster setup and response times as well as higher bandwidth communications enabled by transport protocol behaviour [RFC7323].

There are also potential applications where latency would play an even more significant role. For instance, embedding communications technology in automation or traffic systems, or consumer applications such as augmented or virtual reality where due to the human brain's perceptual limits variability in latency may not be feasible, i.e., render the service unusable due to motion sickness caused.

Many of the Internet-of-Things and critical services use cases in 5G, for instance, have been listed as requiring low latency and high reliability for communications [ER2015] [HU2015] [NGMN2015] [NO2015] [QU2016] [IMT2020].

Some example use cases include optimisation of utility services such as electricity networks, connected automation systems in factories, remote control of machinery such as mining equipment, or embedded technology in road or railway traffic.

The different applications vary in terms of their needs. Some may be very focused on high-speed local area communication, others need to connect at optimal speed over a wide-area network, and yet others need to find the right ways to provide global services without incurring unreasonable delays.

For these reasons it is difficult to specify what "low latency" means in terms of specific delays. Applications and network scenarios differ. Reaching a 50ms latency may be enough for some applications while others may require 50us. Obviously, latency is ultimately

limited by physics, location, and topology. Individual link characteristics are important, but system level communication needs both in terms of what is being communicated and between what parties matter more.

Note that when we say "low-latency capabilities", there is no intent to imply any specific implementation of those capabilities. In particular, we look at the low-latency requirements from a broader perspective than Quality-of-Service guarantees or separating traffic onto different classes. Indeed, while today's virtualisation and software-driven technologies give us more tools to deal with those kinds of arrangements as well, past experience on deploying Quality-of-Service mechanisms in the Internet should give us a pause [CC2015].

It is not the purpose of this memo to analyse the application requirements for low-latency applications much further; for our purposes it suffices to note that there are applications that are enabled by low-latency capabilities of the underlying network infrastructure.

3. Role of Low-Latency vs. Other Communications

There are some limited applications that rely solely on local communication. One example of such an application is vehicles communicating braking status to nearby ones.

Also, while many applications run in the global Internet, some are designed for specialised networks that may not have full or even any Internet connectivity, but yet use IP technology.

However, many applications will include also wide-area communication. If the factory automation machines are not talking other than with themselves, at least their control systems are doing so in order to ensure parts orders, monitoring and maintenance by equipment manufacturers, and so on. This does not imply that these perhaps critical applications are openly accessible from the Internet, but many of them are likely to communicate outside their immediate surroundings.

Many applications also rely on wide-area connectivity for software updates.

As a result, this document recommends that when building architectures for low-latency applications it is important to take into account that these applications can also benefit from communications elsewhere. Or at the very least, the existence of a

specialised communications link or network should not be immediately taken to mean that no other communications are needed.

4. Selected Improvements to Communications Latency

It should be noted that latency is a very broad topic in communications protocol design, almost as broad as "security", or even "correctness".

Implementation techniques to satisfy these requirements vary, some applications can be built with sufficient fast local networking capabilities, others may require, for instance, building world-wide, distributed content delivery mechanisms.

Modern networking systems offer many tools for building low-latency networks, from highly optimised individual protocol components [I-D.ietf-tls-tls13] [I-D.ietf-quic-transport] [RFC7413] [RFC7540] to software controlled, virtualised and tailored network functions [NFV2012] [RFC7665] [I-D.ietf-sfc-nsh] [OF2008]. Data- and software-driven network management and orchestration tools enable networks to be built to serve particular needs as well as to optimize workload placement in a way low-latency requirements could be met.

Obviously, low-latency communications are not merely a matter of protocols and their optimisation. Implementation techniques matter, from the placement of network functions and nodes in the right places, to the quality of individual function implementations. Today's technology allows much freedom on placement of (virtual) functions at chosen locations and many options for all functions ranging from load balancing to storage to packet processing to management tools. Good design can provide significant gains by reducing latency in and between network components, reducing the necessary control traffic and state synchronization, and so on.

Across the stack there are also many other protocol tools, as well as tools being in development, e.g., a new transport design [L4S] at the IETF.

On the lower layers, improvements in radio communications are being made. For instance, the IEEE 802.1 Time-Sensitive Networking Task Group [TSN8021] has worked to define precise time synchronization mechanisms for a local area network, and scheduling mechanisms to enable different classes of traffic to use the same network while minimising jitter and latency. At the IETF, the DETNET working group is taking these capabilities and applying them for layer 3 networking [DETNET].

The 3GPP 5G requirements for next-generation access technology are stringent, and are leading to the optimization of the radio interfaces. The requirements specify a one-way latency limit of 0.5ms for ultra-reliable low-latency communications [TS38913]. But again, mere latency numbers mean very little without the context of a system and what an application needs to communicate and with whom.

5. Architectural Considerations

Despite a large number of specific developments, it may be interesting to view the developments from a system viewpoint, and to consider the potential future stresses that the strive for low-latency support for applications may bring.

5.1. Background

To begin with, it may be useful to observe that the requirements and developments outlined above do not necessarily imply that any specific new technology is needed or that the nature of communications in the Internet would somehow fundamentally change. And certainly not that latency should be the only or primary concern in technology development.

With the drive for a new class of applications, there is often an expectation that this means significant changes. However, all changes need to stand on their own, be justifiable and deployable on a global network. For instance, the discussion around the introduction of the newest 4K or 8K high-definition video streaming applications is reminiscent of the discussions about the introduction of VoIP applications in the Internet. At the time, there was some expectation that special arrangements and Quality-of-Service mechanisms might be needed to support this new traffic class. This turned out to be not true, at least not in general networks.

Experience tells us, for instance, that deploying Quality-of-Service mechanisms in the Internet is hard, not so much because of the technology itself, but due to lack of forces that would be able to drive the necessary business changes in the ecosystem for the technology to be feasibly deployable [CC2015]. As claffy and Clark note:

"Although the Internet has a standards body (the IETF) to resolve technical issues, it lacks any similar forum to discuss business issues such as how to allocate revenues among competing ISPs offering enhanced services. In the U.S., ISPs feared such discussions would risk anti-trust scrutiny. Thus, lacking a way to negotiate the business implications of QoS, it was considered a cost rather than a potential source of revenue. Yet, the

relentless growth of a diversity of applications with widely varying performance requirements continued on the public Internet, with ISPs using relatively primitive, and not always completely benign, mechanisms for handling them."

These difficulties should not be read as prohibiting all changes. Of course, change can also seem unlikely even in cases where it becomes absolutely necessary or the forces necessary to make a change have actually built up. As a result, statements regarding change in the Internet should be carefully evaluated on their merits from both technical and ecosystem perspective.

Secondly, we often consider characteristics from a too narrow viewpoint. In the case of latency, it is easy to focus on a particular protocol or link, whereas from the user perspective latency is a property of the system, not a property of an individual component.

For instance, improvements on the performance of one link on a communications path can be insignificant, if the other parts make up a significant fraction of the system-level latency. That may seem obvious, but many applications are highly dependent on communications between a number of different parties which may reside in different places. For instance, a third party may perform authentication for a cloud-based service that also interacts with user's devices and a number of different sensors and actuators.

We cannot change the speed of light, and a single exchange with another part of the world may result in a 100ms delay, or about 200 times longer than the expected 5G radio link delay for critical applications. It is clear that designing applications from a system perspective is very important.

5.2. Implications

This section discusses a selected set of architectural effects and design choices within applications that desire low latency communications.

5.2.1. Service Distribution

As noted above, low-latency applications need to pay particular attention to the placement of services in the global network. Operations that are on the critical path for the low-latency aspects of an application are unlikely to work well if those communications need to traverse half of the Internet.

Many widely used services are already distributed and replicated throughout the world, to minimise communications latency. But many other services are not distributed in this manner. For low-latency applications such distribution becomes necessary. Hosting a global service in one location is not feasible due to latency, even when from a scale perspective a single server might otherwise suffice for the service. All major public cloud providers offer CDN services to their customers - AWS's CloudFront, Google's Cloud CDN and Azure's CDN to mention a few.

Content-Delivery Networks (CDNs) and similar arrangements are likely to flourish because of this. These arrangements can bring content close to end-users, and have a significant impact on latency. Typical CDN arrangements provide services that are on a global scale nearby, e.g., in the same country or even at the ISP's datacenter.

Today's CDNs are of course just one form of distributed service implementation. Previous generations, such as web caching, have existed as well, and it is likely that the current arrangements will evolve in the future. CDN evolution is also naturally affected not only by the need to provide services closer to the user, but also through the fine-grained control and visibility mechanisms that it gives to the content owners. Such factors continue to affect also future evolution, e.g., any information-centric networking solutions that might emerge.

5.2.2. Edge Computing

Recent advances in "edge computing" take the more traditional type service like CDN as well as a new class of services that require "local compute" capabilities placement even further by providing services near the users. This would enable more extreme use cases where latency from, say, ISP datacenter to the users is considered too high. An important consideration is what is considered an edge, however. From Internet perspective edge usually refers to the IP point of presence or the first IP hop. But given the centralised nature of many access networks, some of the discussions around the use of edge computing also involve components at the edge that are much closer to user than the first IP hop. Special arrangements are needed to enable direct IP connectivity from the user equipment to these components.

5.2.3. Routing and tunnels

How the communications are routed also matters. For instance, architectures based on tunneling to a central point may incur extra delay. One way to address this pressure is to use SDN- and virtualisation-based networks that can be provisioned in the desired

manner, so that, for instance, instances of tunnel servers can be placed in the topologically optimal place for a particular application.

5.2.4. Alternative Paths and Control Tension

Recent developments in multipath transport protocols [RFC6824] also provide application- and service-level control of some of the networking behaviour. Similar choices among alternative paths also exist in simpler techniques, ranging from server selection algorithms to IPv6 "Happy Eyeballs" algorithms [RFC6555]. In all of these cases an application makes some observations of the environment and decides to use an alternative path or target that is perceived to be best suited for the application's needs.

In all of these multipath and alternative selection techniques there is tension between application control (often by content providers) and network control (often by network operators).

One special case where that tension has appeared in the past is whether there should be ways to provide information from applications to networks on how packets should be treated. This was extensively discussed during the discussion stemming from implications of increased use of encryption in the Internet, and how that affects operators [I-D.nrooney-marnew-report].

Another case where there is tension is between mechanisms designed for a single link or network vs. end-to-end mechanisms. Many of the stated requirements for low-latency applications are explicitly about end-to-end characteristics and capabilities. Yet, the two mechanisms are very different, and most of the deployment difficulties reported in [CC2015] relate to end-to-end mechanisms.

Note that some of the multipath techniques can be used either by endpoints or by the network. Proxy-based Multipath TCP is one example of this [I-D.boucadair-mptcp-plain-mode].

5.2.5. Quality-of-Service

Existing approaches have not necessarily proven to be technical deficient in any way, but it seems that it is reasonable to draw some conclusions about the lack of feasibility for deploying mechanisms that require a high degree of coordination among multiple parties. Without significant changes in the marketplace or other conditions, these types of solutions do not seem likely to get traction.

The other observation that may be worth noting is that many networks have focused on providing highly tuned services for a relatively

small fractions of traffic. While this may be justifiable from the perspective of special applications needing that support, this does not seem to produce a good bang-for-the-buck ratio. There's relatively small amount of work on mechanisms that would help a large fraction of applications. For instance, in many access networks the over-the-top content is by far the biggest source of traffic. Solutions that are business-wise deployable for such traffic would seem preferable.

5.2.6. Cross-Layer Optimisations

In the search for even faster connection setup times one obvious technique is cross-layer optimisation. We have seen some of this in the IETF in the rethinking of the layers for transport, transport layer security, and application framework protocols. By taking into account the protocol layer interactions or even bundling the protocol design together, it is relatively easy to optimise the connection setup time, as evidenced by recent efforts to look for "0-RTT" designs in various protocols.

But while cross-layer optimisation can bring benefits, it also has downsides. In particular, it connects different parts of the stack in additional ways. This can lead to difficulties in further evolution of the technology, if done wrong.

In the case of the IETF transport protocol evolution, significant improvements were made to ensure better evolvability of the protocols than what we've experienced with TCP, starting from an ability to implement the new protocols in applications rather than in the kernel.

While the connection setup is an obvious example, cross-layer optimisations are not limited to them. Interfaces between application, transport, networking, and link layers can provide information and set parameters that improve latency. For instance, setting DSCP values or requesting a specialised L2 service for a particular application. Cross-Layer optimisations between lower layers will be discussed in the upcoming versions of the draft.

The effects of badly designed cross-layer optimisation are a particular form of Internet ossification. The general networking trend, however, is for greater flexibility and programmability. Arguably, the ease at which networks can evolve is probably even more important than their specific characteristics.

These comments about cross-layer optimisation should not be interpreted to mean that protocol design should not take into account how other layers behave. The IETF has a long tradition of discussing

link layer design implications for Internet communications (see, e.g., the results of the PILC working group [RFC3819]).

5.3. Recommendations for Further Work

Low-latency applications continue to be a hot topic in networking. The following topics in particular deserve further work from an architectural point of view:

- o Application architectures for globally connected but low-latency services.
- o What are the issues with inter-domain Quality-of-Service mechanisms? Are there approaches that would offer progress on this field? Work on Quality-of-Service mechanisms that are deployable for common cases, and without excessive need for technical and non-technical coordination across the ecosystem.
- o Network architectures that employ tunneling, and mitigations against the delay impacts of tunnels (such as tunnel server placement or "local breakout" techniques). Low latency often implies high reliability, special care is to be taken of network convergence, and other, relevant characteristics of the underlying infrastructure.
- o The emergence of cross-layer optimisations and how that affects the Internet architecture and its future evolution.
- o Inter-organisational matters, e.g., to what extent different standards organisations need to talk about low latency effects and ongoing work, to promote system-level understanding?

Overall, this memo stresses the importance of the system-level understanding of Internet applications and their latency issues. Efforts to address specific sub-issues are unlikely to be fruitful without a holistic plan.

In the authors' opinion, the most extreme use cases (e.g., 1ms or smaller latencies) are not worth building general-purpose networks for. But having the necessary tools so that networks can be flexible for the more general cases is very useful, as there are many applications that can benefit from the added flexibility. The key tools for this include ability to manage network function placement and topology.

6. Acknowledgements

The author would like to thank Brian Trammell, Mirja Kuehlewind, Linda Dunbar, Goran Rune, Ari Keranen, James Kempf, Stephen Farrell, Mohamed Boucadair, Kumar Balachandran, Jani-Pekka Kainulainen, Goran AP Eriksson, and many others for interesting discussions in this problem space.

The author would also like to acknowledge the important contribution that [I-D.dunbar-e2e-latency-arch-view-and-gaps] made in this topic.

7. Informative References

- [CC2015] claffy, kc. and D. Clark, "Adding Enhanced Services to the Internet: Lessons from History", September 2015 (https://www.caida.org/publications/papers/2015/adding_enhanced_services_internet/adding_enhanced_services_internet.pdf).
- [DETNET] "Deterministic Networking (DETNET) Working Group", March 2016 (<https://tools.ietf.org/wg/detnet/charters>).
- [ER2015] Yilmaz, O., "5G Radio Access for Ultra-Reliable and Low-Latency Communications", Ericsson Research Blog, May 2015 (<https://www.ericsson.com/research-blog/5g/5g-radio-access-for-ultra-reliable-and-low-latency-communications/>).
- [HU2015] "5G Vision: 100 Billion connections, 1 ms Latency, and 10 Gbps Throughput", Huawei 2015 (<http://www.huawei.com/minisite/5g/en/defining-5g.html>).
- [I-D.boucadair-mptcp-plain-mode] Boucadair, M., Jacquenet, C., Bonaventure, O., Behaghel, D., stefano.secci@lip6.fr, s., Henderickx, W., Skog, R., Vinapamula, S., Seo, S., Cloetens, W., Meyer, U., Contreras, L., and B. Peirens, "Extensions for Network-Assisted MPTCP Deployment Models", draft-boucadair-mptcp-plain-mode-10 (work in progress), March 2017.
- [I-D.dunbar-e2e-latency-arch-view-and-gaps] Dunbar, L., "Architectural View of E2E Latency and Gaps", draft-dunbar-e2e-latency-arch-view-and-gaps-01 (work in progress), March 2017.

- [I-D.ietf-quick-transport]
Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quick-transport-07 (work in progress), October 2017.
- [I-D.ietf-sfc-nsh]
Quinn, P., Elzur, U., and C. Pignataro, "Network Service Header (NSH)", draft-ietf-sfc-nsh-27 (work in progress), October 2017.
- [I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-21 (work in progress), July 2017.
- [I-D.nrooney-marnew-report]
Rooney, N., "IAB Workshop on Managing Radio Networks in an Encrypted World (MaRNEW) Report", draft-nrooney-marnew-report-03 (work in progress), June 2017.
- [IMT2020] "Framework and overall objectives of the future development of IMT for 2020 and beyond", ITU Recommendation M.2083-0, September 2015 (<http://www.itu.int/rec/R-REC-M.2083-0-201509-I/en>).
- [L4S] "Low Latency Low Loss Scalable throughput (L4S) Birds-of-Feather Session", July 2016 (<https://datatracker.ietf.org/wg/l4s/charter/>).
- [NFV2012] "Network Functions Virtualisation - Introductory White Paper", ETSI, http://portal.etsi.org/NFV/NFV_White_Paper.pdf, October 2012.
- [NGMN2015] "5G White Paper", NGMN Alliance, February 2015 (https://www.ngmn.org/fileadmin/ngmn/content/downloads/Technical/2015/NGMN_5G_White_Paper_V1_0.pdf).
- [NO2015] Doppler, K., "5G the next major wireless standard", DREAMS Seminar, January 2015 (https://chess.eecs.berkeley.edu/pubs/1084/doppler-DREAMS_5G_jan15.pdf).

- [OF2008] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks", ACM SIGCOMM Computer Communication Review, Volume 38, Issue 2, pp. 69-74 2008.
- [QU2016] "Leading the world to 5G", Qualcomm, February 2016 (<https://www.qualcomm.com/media/documents/files/qualcomm-5g-vision-presentation.pdf>).
- [RFC3819] Karn, P., Ed., Bormann, C., Fairhurst, G., Grossman, D., Ludwig, R., Mahdavi, J., Montenegro, G., Touch, J., and L. Wood, "Advice for Internet Subnetwork Designers", BCP 89, RFC 3819, DOI 10.17487/RFC3819, July 2004, <<https://www.rfc-editor.org/info/rfc3819>>.
- [RFC6555] Wing, D. and A. Yourtchenko, "Happy Eyeballs: Success with Dual-Stack Hosts", RFC 6555, DOI 10.17487/RFC6555, April 2012, <<https://www.rfc-editor.org/info/rfc6555>>.
- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <<https://www.rfc-editor.org/info/rfc6824>>.
- [RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, DOI 10.17487/RFC7323, September 2014, <<https://www.rfc-editor.org/info/rfc7323>>.
- [RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <<https://www.rfc-editor.org/info/rfc7413>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<https://www.rfc-editor.org/info/rfc7540>>.
- [RFC7665] Halpern, J., Ed. and C. Pignataro, Ed., "Service Function Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <<https://www.rfc-editor.org/info/rfc7665>>.

- [TS38913] "3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Study on Scenarios and Requirements for Next Generation Access Technologies; (Release 14)", 3GPP Technical Report TR 38.913 V14.2.0, March 2017 (<https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2996>).
- [TSN8021] "Time-Sensitive Networking Task Group", IEEE (<http://www.ieee802.org/1/pages/tsn.html>).

Authors' Addresses

Jari Arkko
Ericsson
Kauniainen 02700
Finland

Email: jari.arkko@piuha.net

Jeff Tantsura
Futurewei, Future Networks

Email: jefftant.ietf@gmail.com

Network Working Group
Internet Draft
Intended status: Standard Track
Expires: April 2019

A. Bashandy
Arrcus
C. Filsfils
Cisco Systems
Bruno Decraene
Stephane Litkowski
Orange
Pierre Francois
INSA Lyon
D. Voyer
Bell Canada
Francois Clad
Pablo Camarillo
Cisco Systems
October 4, 2018

Topology Independent Fast Reroute using Segment Routing
draft-bashandy-rtgwg-segment-routing-ti-lfa-05

Abstract

This document presents Topology Independent Loop-free Alternate Fast Re-route (TI-LFA), aimed at providing protection of node and adjacency segments within the Segment Routing (SR) framework. This Fast Re-route (FRR) behavior builds on proven IP-FRR concepts being LFAs, remote LFAs (RLFA), and remote LFAs with directed forwarding (DLFA). It extends these concepts to provide guaranteed coverage in any IGP network. A key aspect of TI-LFA is the FRR path selection approach establishing protection over post-convergence paths from the point of local repair, dramatically reducing the operational need to control the tie-breaks among various FRR options.

Status of this Memo

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

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

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/ietf/lid-abstracts.txt>

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

This Internet-Draft will expire on April 4, 2019.

Copyright Notice

Copyright (c) 2018 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. Conventions used in this document.....	5
2. Terminology.....	5
3. Intersecting P-Space and Q-Space with post-convergence paths...	6
3.1. P-Space property computation for a resource X.....	6
3.2. Q-Space property computation for a link S-F, over post-convergence paths.....	6
3.3. Q-Space property computation for a set of links adjacent to S, over post-convergence paths.....	7
3.4. Q-Space property computation for a node F, over post-convergence paths.....	7
4. TI-LFA Repair Tunnel.....	7
4.1. The repair node is a direct neighbor.....	7

4.2. The repair node is a PQ node.....	8
4.3. The repair is a Q node, neighbor of the last P node.....	8
4.4. Connecting distant P and Q nodes along post-convergence paths.....	8
5. Protecting segments.....	8
5.1. The active segment is a node segment.....	8
5.2. The active segment is an adjacency segment.....	9
5.2.1. Protecting [Adjacency, Adjacency] segment lists.....	9
5.2.2. Protecting [Adjacency, Node] segment lists.....	9
5.3. Protecting SR policy midpoints against node failure.....	10
5.3.1. Protecting {F, T, D} or {S->F, T, D}.....	10
5.3.2. Protecting {F, F->T, D} or {S->F, F->T, D}.....	10
6. Measurements on Real Networks.....	11
7. Security Considerations.....	17
8. IANA Considerations.....	17
9. Conclusions.....	17
10. References.....	17
10.1. Normative References.....	17
10.2. Informative References.....	17
11. Acknowledgments.....	18

1. Introduction

Segment Routing aims at supporting services with tight SLA guarantees [1]. By relying on segment routing this document provides a local repair mechanism for standard IGP shortest path capable of restoring end-to-end connectivity in the case of a sudden directly connected failure of a network component. Non-SR mechanisms for local repair are beyond the scope of this document. Non-local failures are addressed in a separate document [6].

The term topology independent (Ti) refers to the ability to provide a loop free backup path irrespective of the topologies prior the failure and after the failure.

For each destination in the network, TI-LFA prepares a data-plane switch-over to be activated upon detection of the failure of a link used to reach the destination. TI-LFA provides protection in the event of any one of the following: single link failure, single node failure, or single local SRLG failure. In link failure mode, the destination is protected assuming the failure of the link. In node protection mode, the destination is protected assuming that the neighbor connected to the primary link has failed. In local SRLG protecting mode, the destination is protected assuming that a configured set of links sharing fate with the primary link has failed (e.g. a linecard).

Protection techniques outlined in this document are limited to protecting links, nodes, and local SRLGs that are within a routing

domain. Protecting domain exit routers and/or links attached to another routing domains are beyond the scope of this document

Using segment routing, there is no need to establish TLDP sessions with remote nodes in order to take advantage of the applicability of remote LFAs (RLFA) [4][5] or remote LFAs with directed forwarding (DLFA)[2]. As a result, preferring LFAs over RLFAs or DLFAs, as well as minimizing the number of RLFA or DLFA repair nodes is not required

Using SR, there is no need to create state in the network in order to enforce an explicit FRR path thereby relieving the nodes from the extra state and the operator from having to deploy an extra protocol just to enhance FRR coverage.

The FRR behavior suggested in this document tailors the repair paths over the post-convergence path from the PLR to the protected destination, given the enabled protection mode for the interface. Using the post-convergence path in TI-LFA resolves some of operational issues with LFA selection that are mentioned in Section 3 of [5] (e.g. using PE routers to protect against core failures, or selecting links with low BW while links with high BW are available), because these issues presumably have been taken care of by the network operator as part of its original network engineering. Hence traffic that permanently uses the PLR after the failure achieves maximum benefits. Traffic that does not use the PLR prior to and after the failure remains unaffected. Traffic that temporarily continues to use the PLR after the failure benefits from the quick switching to the backup path by minimizing traffic loss until remote node(s) reacts.

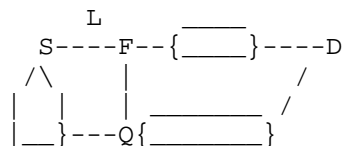


Figure 1 TI-LFA Protection

We use Figure 1 to illustrate the TI-LFA approach.

The Point of Local Repair (PLR), S, needs to find a node Q (a repair node) that is capable of safely forwarding the traffic to a destination D affected by the failure of the protected link L, a set of adjacent links including L (local SRLG), or the node F itself. The PLR also needs to find a way to reach Q without being affected by the convergence state of the nodes over the paths it wants to use to reach Q.

In Section 2 we define the main notations used in the document. They are in line with [2].

In Section 3, we suggest to compute the P-Space and Q-Space properties defined in Section 2, for the specific case of nodes lying over the post-convergence paths towards the protected destinations.

Using the properties defined in Section 3, Section 4 describes how to compute protection lists that encode a loopfree post-convergence path towards the destination.

Section 5 defines the segment operations to be applied by the PLR to ensure consistency with the forwarding state of the repair node.

By applying the algorithms specified in this document to actual service providers and large enterprise networks, we provide real life measurements for the number of SIDs used by repair paths. Section 6 summarizes these measurements.

1.1. 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 RFC-2119

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 RFC-2119 significance.

2. Terminology

We define the main notations used in this document as the following.

We refer to "old" and "new" topologies as the LSDB state before and after the considered failure.

SPT_old(R) is the Shortest Path Tree rooted at node R in the initial state of the network.

SPT_new(R, X) is the Shortest Path Tree rooted at node R in the state of the network after the resource X has failed.

Dist_old(A,B) is the shortest distance from node A to node B in SPT_old(A).

Dist_new(A,B, X) is the shortest distance from node A to node B in SPT_new(A,X).

PLR stands for "Point of Local Repair". It is the router that applies fast traffic restoration after detecting failure in a directly attached link, set of links, and/or node.

Similar to [4], we use the concept of P-Space and Q-Space for TI-LFA.

The P-Space $P(R,X)$ of a node R w.r.t. a resource X (e.g. a link $S-F$, a node F , or a local SRLG) is the set of nodes that are reachable from R without passing through X . It is the set of nodes that are not downstream of X in $SPT_old(R)$.

The Extended P-Space $P'(R,X)$ of a node R w.r.t. a resource X is the set of nodes that are reachable from R or a neighbor of R , without passing through X .

The Q-Space $Q(D,X)$ of a destination node D w.r.t. a resource X is the set of nodes which do not use X to reach D in the initial state of the network. In other words, it is the set of nodes which have D in their P-Space w.r.t. $S-F$, F , or a set of links adjacent to S .

A symmetric network is a network such that the IGP metric of each link is the same in both directions of the link.

3. Intersecting P-Space and Q-Space with post-convergence paths

In this section, we suggest to determine the P-Space and Q-Space properties of the nodes along the post-convergence paths from the PLR to the protected destination and compute an SR-based explicit path from P to Q when they are not adjacent. Such properties will be used in Section 4 to compute the TI-LFA repair list.

3.1. P-Space property computation for a resource X

A node N is in $P(R, X)$ if it is not downstream of X in $SPT_old(R)$. X can be a link, a node, or a set of links adjacent to the PLR. A node N is in $P'(R,X)$ if it is not downstream of X in $SPT_old(N)$, for at least one neighbor N of R .

3.2. Q-Space property computation for a link $S-F$, over post-convergence paths

We want to determine which nodes on the post-convergence path from the PLR to the destination D are in the Q-Space of destination D w.r.t. link $S-F$.

This can be found by intersecting the post-convergence path to D , assuming the failure of $S-F$, with $Q(D, S-F)$.

3.3. Q-Space property computation for a set of links adjacent to S, over post-convergence paths

We want to determine which nodes on the post-convergence path from the PLR to the destination D are in the Q-Space of destination D w.r.t. a set of links adjacent to S (S being the PLR). That is, we aim to find the set of nodes on the post-convergence path that use none of the members of the protected set of links, to reach D.

This can be found by intersecting the post-convergence path to D, assuming the failure of the set of links, with the intersection among $Q(D, S \rightarrow X)$ for all $S \rightarrow X$ belonging to the set of links.

3.4. Q-Space property computation for a node F, over post-convergence paths

We want to determine which nodes on the post-convergence from the PLR to the destination D are in the Q-Space of destination D w.r.t. node F.

This can be found by intersecting the post-convergence path to D, assuming the failure of F, with $Q(D, F)$.

4. TI-LFA Repair Tunnel

The TI-LFA repair tunnel consists of an outgoing interface and a list of segments (repair list) to insert on the SR header. The repair list encodes the explicit post-convergence path to the destination, which avoids the protected resource X and, at the same time, is guaranteed to be loop free irrespective of the state of FIBs along the nodes belonging to the explicit path. Thus there is no need for any co-ordination or message exchange between the PLR and any other router in the network.

The TI-LFA repair tunnel is found by intersecting $P(S, X)$ and $Q(D, X)$ with the post-convergence path to D and computing the explicit SR-based path $EP(P, Q)$ from P to Q when these nodes are not adjacent along the post convergence path. The TI-LFA repair list is expressed generally as $(Node_SID(P), EP(P, Q))$.

Most often, the TI-LFA repair list has a simpler form, as described in the following sections. Section 6 provides statistics for the number of SIDs in the explicit path to protect against various failures.

4.1. The repair node is a direct neighbor

When the repair node is a direct neighbor, the outgoing interface is set to that neighbor and the repair segment list is empty.

This is comparable to a post-convergence LFA FRR repair.

4.2. The repair node is a PQ node

When the repair node is in $P(S,X)$, the repair list is made of a single node segment to the repair node.

This is comparable to a post-convergence RLFA repair tunnel.

4.3. The repair is a Q node, neighbor of the last P node

When the repair node is adjacent to $P(S,X)$, the repair list is made of two segments: A node segment to the adjacent P node, and an adjacency segment from that node to the repair node.

This is comparable to a post-convergence DLFA repair tunnel.

4.4. Connecting distant P and Q nodes along post-convergence paths

In some cases, there is no adjacent P and Q node along the post-convergence path. However, the PLR can perform additional computations to compute a list of segments that represent a loopfree path from P to Q.

5. Protecting segments

In this section, we explain how a protecting router S processes the active segment of a packet upon the failure of its primary outgoing interface for the packet, S-F.

The behavior depends on the type of active segment to be protected.

5.1. The active segment is a node segment

The active segment is kept on the SR header, unchanged (1). The repair list is inserted at the head of the list. The active segment becomes the first segment of the inserted repair list.

Note (1): If SR-MPLS is being used and the SRGB at the repair node is different from the SRGB at the PLR, then the active segment MUST be updated to fit the SRGB of the repair node.

In Section 5.3, we describe the node protection behavior of PLR S, for the specific case where the active segment is a prefix segment for the neighbor F itself.

5.2. The active segment is an adjacency segment

We define hereafter the FRR behavior applied by S for any packet received with an active adjacency segment S-F for which protection was enabled. We distinguish the case where this active segment is followed by another adjacency segment from the case where it is followed by a node segment.

5.2.1. Protecting [Adjacency, Adjacency] segment lists

If the next segment in the list is an Adjacency segment, then the packet has to be conveyed to F.

To do so, S applies a "NEXT" operation on Adj(S-F) and then two consecutive "PUSH" operations: first it pushes a node segment for F, and then it pushes a protection list allowing to reach F while bypassing S-F. For details on the "NEXT" and "PUSH" operations, refer to [7].

Upon failure of S-F, a packet reaching S with a segment list matching [adj(S-F),adj(M),...] will thus leave S with a segment list matching [RT(F),node(F),adj(M)], where RT(F) is the repair tunnel for destination F.

In Section 5.3.2, we describe the TI-LFA behavior of PLR S when node protection is applied and the two first segments are Adjacency Segments.

5.2.2. Protecting [Adjacency, Node] segment lists

If the next segment in the stack is a node segment, say for node T, the packet segment list matches [adj(S-F),node(T),...].

A first solution would consist in steering the packet back to F while avoiding S-F. To do so, S applies a "NEXT" operation on Adj(S-F) and then two consecutive "PUSH" operations: first it pushes a node segment for F, and then it pushes a repair list allowing to reach F while bypassing S-F.

Upon failure of S-F, a packet reaching S with a segment list matching [adj(S-F),node(T),...] will thus leave S with a segment list matching [RT(F),node(F),node(T)].

Another solution is to not steer the packet back via F but rather follow the new shortest path to T. In this case, S just needs to apply a "NEXT" operation on the Adjacency segment related to S-F, and push a repair list redirecting the traffic to a node Q, whose path to node segment T is not affected by the failure.

Upon failure of S-F, packets reaching S with a segment list matching [adj(L), node(T), ...], would leave S with a segment list matching [RT(Q), node(T), ...]. Note that this second behavior is the one followed for node protection, as described in Section 5.3.1.

5.3. Protecting SR policy midpoints against node failure

In this section, we describe the behavior of a node S configured to interpret the failure of link S->F as the node failure of F, in the specific case where the active segment of the packet received by S is a Prefix SID of F represented as "F", or an Adjacency SID for the link S-F (represented as "S->F").

5.3.1. Protecting {F, T, D} or {S->F, T, D}

This section describes the protection behavior of S when all of the following conditions are true:

1. the active segment is a prefix SID for a neighbor F, or an adjacency segment S->F
2. the primary interface used to forward the packet failed
3. the segment following the active segment is a prefix SID (for node T)
4. node protection is active for that interface.

The TILFA Node FRR behavior becomes equivalent to:

1. Pop; the segment F or S->F is removed
2. Confirm that the next segment is in the SRGB of F, meaning that the next segment is a prefix segment, e.g. for node T
3. Identify T (as per the SRGB of F)
4. Pop the next segment and push T's segment based on the local SRGB
5. forward the packet according to T.

5.3.2. Protecting {F, F->T, D} or {S->F, F->T, D}

This section describes the protection behavior of S when all of the following conditions are true:

1. the active segment is a prefix SID for a neighbor F, or an adjacency segment S->F

2. the primary interface used to forward the packet failed
3. the segment following the active segment is an adjacency SID (F->T)
4. node protection is active for that interface.

The TILFA Node FRR behavior becomes equivalent to:

1. Pop; the segment F or S->F is removed
2. Confirm that the next segment is an adjacency SID of F, say F->T
3. Identify T (as per the set of Adjacency Segments of F)
4. Pop the next segment and push T's segment based on the local SRGB
5. forward the packet according to T.

It is noteworthy to mention that node "S" in the procedures described in Sections 5.3.1 and 5.3.2 can always determine whether the segment after popping the top segment is an adjacency SID or a prefix-SID of the next-hop "F" as follows:

1. In a link state environment, the node "S" knows the SRGB and the adj-SIDs of the neighboring node "F"
2. If the new segment after popping the top segment is within the SRGB or the adj-SIDs of "F", then node "S" is certain that the failure of node "F" is a midpoint failure and hence node "S" applies the procedures specified in Sections 5.3.1 or 5.3.2, respectively.
3. Otherwise the failure is not a midpoint failure and hence the node "S" may apply other protection techniques that are beyond the scope of this document or simply drop the packet and wait for normal protocol conversion.

6. Measurements on Real Networks

This section presents measurements performed on real service provider and large enterprise networks. The objective of the measurements is to assess the number of SIDs required in an explicit path when the mechanism described in this document are used to

protect against the failure scenarios within the scope of this document. The number of segments described in this section are applicable to instantiating segment routing over the MPLS forwarding plane.

The measurements below indicate that for link and local SRLG protection, a 1 SID repair path delivers more than 99% coverage. For node protection a 2 SIDs repair path yields 99% coverage.

Table 1 below lists the characteristics of the networks used in our measurements. The measurements are carried out as follows

- o For each network, the algorithms described in this document are applied to protect all prefixes against link, node, and local SRLG failure
- o For each prefix, the number of SIDs used by the repair path is recorded
- o The percentage of number of SIDs are listed in Tables 2A/B, 3A/B, and 4A/B

The measurements listed in the tables indicate that for link and local SRLG protection, 1 SID repair paths are sufficient to protect more than 99% of the prefix in almost all cases. For node protection 2 SIDs repair paths yield 99% coverage.

Network	Nodes	Circuits	Node-to-Link Ratio	SRLG info?
T1	408	665	1 : 63	Yes
T2	587	1083	1 : 84	No
T3	93	401	4 : 31	Yes
T4	247	393	1 : 59	Yes
T5	34	96	2 : 82	Yes
T6	50	78	1 : 56	No
T7	82	293	3 : 57	No
T8	35	41	1 : 17	Yes
T9	177	1371	7 : 74	Yes

Table 1: Data Set Definition

The rest of this section presents the measurements done on the actual topologies. The convention that we use is as follows

- o 0 SIDs: the calculated repair path starts with a directly connected neighbor that is also a loop free alternate, in which case there is no need to explicitly route the traffic using additional SIDs. This scenario is described in Section 4.1.
- o 1 SIDs: the repair node is a PQ node, in which case only 1 SID is needed to guarantee loop-freeness. This scenario is covered in Section 4.2.
- o 2 or more SIDs: The repair path consists of 2 or more SIDs as described in Sections 4.3 and 4.4. We do not cover the case for 2 SIDs (Section 4.3) separately because there was no granularity in the result. Also we treat the node-SID+adj-SID and node-SID + node-SID the same because they do not differ from the data plane point of view.

Table 2A and 2B below summarize the measurements on the number of SIDs needed for link protection

Network	0 SIDs	1 SID	2 SIDs	3 SIDs
T1	74.227%	25.256%	0.517%	0.001%
T2	81.097%	18.738%	0.165%	0.0%
T3	95.878%	4.067%	0.056%	0.0%
T4	62.547%	35.666%	1.788%	0.0%
T5	85.733%	14.267%	0.0%	0.0%
T6	81.252%	18.714%	0.033%	0.0%
T7	98,857%	1.143%	0.0%	0.0%
T8	94,118%	5.882%	0.0%	0.0%
T9	98.950%	1.050%	0.0%	0.0%

Table 2A: Link protection (repair size distribution)

Network	0 SIDs	1 SID	2 SIDs	3 SIDs
T1	74.227%	99.482%	99.999%	100.0%
T2	81.097%	99.835%	100.0%	100.0%
T3	95.878%	99.944%	100.0%	100.0%
T4	62.547%	98.212%	100.0%	100.0%
T5	85.733%	100.000%	100.0%	100.0%
T6	81.252%	99.967%	100.0%	100.0%
T7	98,857%	100.000%	100.0%	100.0%
T8	94,118%	100.000%	100.0%	100.0%
T9	98,950%	100.000%	100.0%	100.0%

Table 2B: Link protection repair size cumulative distribution

Table 3A and 3B summarize the measurements on the number of SIDs needed for local SRLG protection.

Network	0 SIDs	1 SID	2 SIDs	3 SIDs
T1	74.177%	25.306%	0.517%	0.001%
T2	No SRLG Information			
T3	93.650%	6.301%	0.049%	0.0%
T4	62,547%	35.666%	1.788%	0.0%
T5	83.139%	16.861%	0.0%	0.0%
T6	No SRLG Information			
T7	No SRLG Information			
T8	85.185%	14.815%	0.0%	0.0%
T9	98,940%	1.060%	0.0%	0.0%

Table 3A: Local SRLG protection repair size distribution

Network	0 SIDs	1 SID	2 SIDs	3 SIDs
T1	74.177%	99.482%	99.999%	100.001%
T2	No SRLG Information			
T3	93.650%	99.951%	100.000%	0.0%
T4	62,547%	98.212%	100.000%	100.0%
T5	83.139%	100.000%	100.0%	100.0%
T6	No SRLG Information			
T7	No SRLG Information			
T8	85.185%	100,000%	100.000%	100.0%
T9	98,940%	100,000%	100.000%	100.0%

Table 3B: Local SRLG protection repair size Cumulative distribution

The remaining two tables summarize the measurements on the number of SIDs needed for node protection.

Network	0 SIDs	1 SID	2 SIDs	3 SIDs	4 SIDs
T1	49.771%	47.902%	2.156%	0.148%	0.023%
T2	36,528%	59.625%	3.628%	0.194%	0.025%
T3	73,287%	25,574%	1,128%	0.010%	0%
T4	36.112%	57.350%	6.329%	0.199%	0.010%
T5	73.185%	26.815%	0%	0%	0%
T6	78.362%	21.320%	0.318%	0%	0%
T7	66.106%	32.813%	1.082%	0%	0%
T8	59.712%	40.288%	0%	0%	0%
T9	98.950%	1.050%	0%	0%	0%

Table 4A: Node protection (repair size distribution)

Network	0 SIDs	1 SID	2 SIDs	3 SIDs	4 SIDs
T1	49.771%	97.673%	99.829%	99.977%	100%
T2	36,528%	96.153%	99.781%	99.975%	100%
T3	73,287%	98.862%	99.990%	100.0%	100%
T4	36.112%	93.461%	99.791%	99.990%	100%
T5	73.185%	100.0%	100.0%	100.0%	100%
T6	78.362%	99.682%	100.0%	100.0%	100%
T7	66.106%	98,918%	100.0%	100.0%	100%
T8	59.712%	100.0%	100.0%	100.0%	100%
T9	98.950%	100.0%	100.0%	100.0%	100%

Table 4B: Node protection (repair size cumulative distribution)

7. Security Considerations

The techniques described in this document is internal functionality to a router that result in the ability to guarantee an upper bound on the time taken to restore traffic flow upon the failure of a directly connected link or node. As these techniques steer traffic to the post-convergence path as quickly as possible, this serves to minimize the disruption associated with a local failure which can be seen as a modest security enhancement. The protection mechanisms does not protect external destinations, but rather provides quick restoration for destination that are internal to a routing domain.

8. IANA Considerations

No requirements for IANA

9. Conclusions

This document proposes a mechanism that is able to pre-calculate a backup path for every primary path so as to be able to protect against the failure of a directly connected link, node, or SRLG. The mechanism is able to calculate the backup path irrespective of the topology as long as the topology is sufficiently redundant.

10. References

10.1. Normative References

10.2. Informative References

- [1] Filsfils, C., Previdi, S., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", draft-ietf-spring-segment-routing-08 (work in progress), May 2016.
- [2] Shand, M. and S. Bryant, "IP Fast Reroute Framework", RFC 5714, January 2010.
- [3] Filsfils, C., Francois, P., Shand, M., Decraene, B., Uttaro, J., Leymann, N., and M. Horneffer, "Loop-Free Alternate (LFA) Applicability in Service Provider (SP) Networks", RFC 6571, June 2012.

- [4] 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>>.
- [5] Litkowski, S., Ed., Decraene, B., Filsfils, C., Raza, K., Horneffer, M., and P. Sarkar, "Operational Management of Loop-Free Alternates", RFC 7916, DOI 10.17487/RFC7916, July 2016, <<https://www.rfc-editor.org/info/rfc7916>>.
- [6] Bashandy, A., Filsfils, C., and Litkowski, S., " Loop avoidance using Segment Routing", draft-bashandy-rtgwg-segment-routing-uloop-00, (work in progress), May 2017
- [7] Filsfils, C., Previdi, S., Decraene, B., Litkowski, S., and Shakir, R, "Segment Routing Architecture", draft-ietf-spring-segment-routing-11 (work in progress), February 2017

11. Acknowledgments

We would like to give Les Ginsberg special thanks for the valuable comments and contribution

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Pierre Francois
INSA Lyon
Email: pierre.francois@insa-lyon.fr

Ahmed Bashandy
Arrcus
Email: abashandy.ietf@gmail.com

Clarence Filsfils
Cisco Systems
Brussels, Belgium
Email: cfilsfil@cisco.com

Bruno Decraene
Orange
Issy-les-Moulineaux
FR
Email: bruno.decraene@orange.com

Stephane Litkowski
Orange
FR
Email: stephane.litkowski@orange.com

Daniel Voyer
Bell Canada
Canada
Email: daniel.voyer@bell.ca

Pablo Camarillo
Cisco Systems
Email: pcamaril@cisco.com

Francois Clad
Cisco Systems
Email: fclad@cisco.com

Network Working Group
Internet Draft
Intended status: Standards Track
Expires: June 2022

Ahmed Bashandy
Individual
Clarence Filsfils
Stephane Litkowski
Cisco Systems, Inc.
Bruno Decraene
Orange
Pierre Francois
INSA Lyon
Peter Psenak
Cisco Systems
December 22, 2021

Loop avoidance using Segment Routing
draft-bashandy-rtgwg-segment-routing-uloop-12

Abstract

This document presents a mechanism aimed at providing loop avoidance in the case of IGP network convergence event. The solution relies on the temporary use of SR policies ensuring loop-freeness over the post-convergence paths from the converging node to the destination.

Status of this Memo

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

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

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/ietf/lid-abstracts.txt>

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

This Internet-Draft will expire on June 22, 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 (<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.

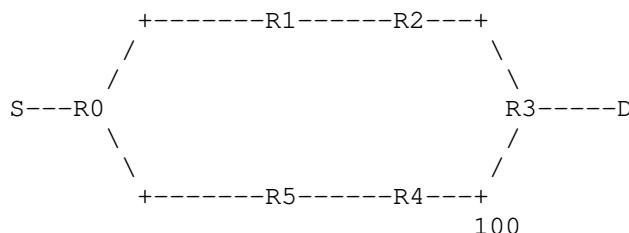
Table of Contents

1. Introduction.....	2
1.1. Conventions used in this document.....	4
2. Loop-free two-stage convergence process.....	4
3. Computing loop-avoiding SR policies.....	5
4. Analysis.....	5
4.1. Incremental Deployment.....	5
4.2. No impact on capacity planning.....	5
5. Security Considerations.....	6
6. IANA Considerations.....	6
7. Contributors.....	6
8. References.....	6
8.1. Normative References.....	6
8.2. Informative References.....	6
9. Acknowledgments.....	6

1. Introduction

Forwarding loops happen during the convergence of the IGP, as a result of transient inconsistency among forwarding states of the nodes of the network.

We use Figure 1 to illustrate the mechanism. In this scenario, all the IGP link metrics are 1, excepted R3-R4 whose metric is 100, and all links have symmetric metrics. We consider the traffic from S to D.



When the link between R2 and R3 fails, traffic sent from S to D, initially flowing along S-R0-R1-R2-R3-D is subject to transient forwarding loops while routers update their forwarding state for destination D. For example, if R0 updates its FIB before R5, packets for D may loop between R0 and R5. If R5 updates its FIB before R4, packets for D may loop between R5 and R4.

In our example, R0 can temporarily steer traffic destined to D over SR path [NodeSID(R4), AdjSID(R4->R3), D]. By doing so, packets for D will be forwarded by R5 as per NodeSID(R4), and by R4 as per AdjSID(R4->R3). From R3 on, the packet is forwarded as per destination D. As a result, traffic follows the desired path, regardless of the forwarding state for destination D at R5 and R4. After some time, the normal forwarding behavior (without using an SR policy) can be applied; routers will converge to their final forwarding state, still consistently forwarding along the post-convergence paths across the network.

1.1. 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 RFC-2119

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 RFC-2119 significance.

2. Loop-free two-stage convergence process

Upon a topology change, when a node R converging for destination D does not trust the loop-freeness of its post-convergence path for destination D, it applies the following two-stage convergence process for destination D.

Stage 1: After computing the new path to D, for a predetermined amount of time C, R installs a FIB entry for D that steers packets to D via a loop-free SR path. C should be greater than or equal to the worst-case convergence time of a node, network-wide. The determination of "C" is outside the scope of this document. The SR path is computed when the event occurs.

Stage 2: After C elapses, R installs the normal post-convergence FIB entry for D, i.e. without any additional segments inserted that ensure the loop-free property.

Loop-freeness is ensured during this process, because:

1. Paths made of non up-to-date routers are loop-free.

Routers which forward as per the initial state of the network are consistent.

2. A packet reaching a node in stage 1 is ensured to reach its destination.

When a packet reaches a router in stage 1, it is steered on a SR path ensuring a loop-free post-convergence path, whatever the state of other routers on the path.

3. Paths made of a mix of routers in stage 1 and stage 2 are consistent.

After C milliseconds, all routers are forwarding as per their post-convergence paths, either expressed classically or as a loop-free SR path.

In our example, when R2-R3 fails, R0 forwards traffic for destination D over SR Path [NodeSID(R4), AdjSID(R4->R3), D], for C milliseconds. During that period, packets sent by R0 to D are loop-free as per the application of the policy. When C elapses, R0 now uses its normal post-convergence path to the destination, forwarding packets for D as is to R5.

R5 also implements loop avoidance, and has thus temporarily used a loop-avoiding SR policy for D. This policy is [AdjSID(R4->R3), D], oif R5->R4. If R5 is still applying the stage 1 behavior, the packet will be forwarded using this policy, and will thus safely reach the destination. If R5 also had moved to stage 2, it forwards the packet as per its normal post-convergence path, via R4. The forwarding state of R4 for D at stage 1 and stage 2 are the same: oif R4->R3, as forwarding packets for destination D as is to R3 ensures a loop-free post-convergence path.

3. Computing loop-avoiding SR policies

The computation to turn a post-convergence path into a loop-free list of segments is outside the scope of this document. It is a local behavior at a node.

In a future revision of this document, we may provide a reference approach to compute loop-avoiding policies for link up, link metric increase, link down, link metric decrease, node up, and node down events. TI-LFA Repair Tunnel

4. Analysis

In this section, we review the main characteristics of the proposed solution. These characteristics are illustrated in [3].

4.1. Incremental Deployment

There is no requirement for a full network upgrade to get benefits from the solution.

(1) Nodes that are upgraded bring benefit for the traffic passing through them.

(2) Nodes that are not upgraded to support SR-based loop-avoidance will cause the micro-loops that they were causing before, unless they get avoided by the local behavior of a node supporting the behavior.

4.2. No impact on capacity planning

By ensuring loop-free post-convergence paths, the behavior remains in line with the natural expected convergence process of the IGP.

Enabling SR-based loop-avoidance hence does not require consideration for capacity planning, compared to any loop avoidance mechanism that lets traffic follow a different path than the post-convergence one. The behavior is local. Nothing is expected from remote nodes except the basic support of Prefix and Adjacency SID's.

5. Security Considerations

The behavior described in this document is internal functionality to a router that result in the ability to explicitly steer traffic over the post convergence path after a remote topology change in a manner that guarantees loop freeness. Because the behavior serves to minimize the disruption associated with a topology changes, it can be seen as a modest security enhancement.

6. IANA Considerations

No requirements for IANA

7. Contributors

Additional contributors: Bruno Decraene and Peter Psenak.

8. References

8.1. Normative References

8.2. Informative References

- [1] Filsfils, C., Previdi, S., Decraene, B., Litkowski, S., and Shakir, R., "Segment Routing Architecture", draft-ietf-spring-segment-routing-11 (work in progress), November 2016.
- [2] Shand, M. and Bryant, S., "IP Fast Reroute Framework", RFC 5714, January 2010.
- [3] Litkowski, S., "Avoiding micro-loops using Segment Routing", MPLS World Congress , 2016.

9. Acknowledgments

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Ahmed Bashandy
Individual
Email: abashandy.ietf@gmail.com

Clarence Filsfils
Cisco Systems
Brussels, Belgium
Email: cfilsfil@cisco.com

Stephane Litkowski
Cisco
Email: slitkows.ietf@gmail.com

Bruno Decraene
Orange
Email: bruno.decraene@orange.com

Pierre Francois
INSA Lyon
Email: pierre.francois@insa-lyon.fr

Peter Psenak
Cisco System
Email: ppsenak@cisco.com

Routing Area Working Group
Internet-Draft
Intended status: Informational
Expires: September 6, 2018

S. Bryant
J. Dong
Huawei
Z. Li
China Mobile
T. Miyasaka
KDDI Corporation
March 05, 2018

Enhanced Virtual Private Networks (VPN+)
draft-bryant-rtgwg-enhanced-vpn-02

Abstract

This draft describes a number of enhancements that need to be made to virtual private networks (VPNs) to support the needs of new applications, particularly applications that are associated with 5G services. A network enhanced with these properties may form the underpin of network slicing, but will also be of use in its own right.

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 September 6, 2018.

Copyright Notice

Copyright (c) 2018 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	2
2. Requirements Language	4
3. Overview of the Requirements	4
3.1. Isolation between Virtual Networks	4
3.2. Diverse Performance Guarantees	6
3.3. A Pragmatic Approach to Isolation	7
3.4. Integration	8
3.5. Dynamic Configuration	8
3.6. Customized Control Plane	9
4. Architecture and Components of VPN+	9
4.1. Communications Layering	9
4.2. Multi-Point to Multi-point	10
4.3. Candidate Underlay Technologies	10
4.3.1. FlexE	11
4.3.2. Dedicated Queues	12
4.3.3. Time Sensitive Networking	12
4.3.4. Deterministic Networking	12
4.3.5. MPLS Traffic Engineering (MPLS-TE)	13
4.3.6. Segment Routing	13
4.4. Control Plane Considerations	16
4.5. Application Specific Network Types	17
4.6. Integration with Service Functions	17
5. Scalability Considerations	17
5.1. Maximum Stack Depth	18
5.2. RSVP scalability	18
6. OAM and Instrumentation	19
7. Enhanced Resiliency	19
8. Security Considerations	20
9. IANA Considerations	20
10. References	21
10.1. Normative References	21
10.2. Informative References	21
Authors' Addresses	22

1. Introduction

Virtual networks, often referred to as virtual private networks (VPNs) have served the industry well as a means of providing different groups of users with logically isolated access to a common network. The common or base network that is used to provide the VPNs

is often referred to as the underlay, and the VPN is often called an overlay.

Driven largely by needs surfacing from 5G, the concept of network slicing has gained traction. There is a need to create a VPN with enhanced characteristics. Specifically there is a need for a transport network supporting a set of virtual networks each of which provides the client with dedicated (private) networking, computing and storage resources drawn from a shared pool. The tenant of such a network can require a degree of isolation and performance that previously could only be satisfied by dedicated networks. Additionally the tenant may ask for some level of control of their virtual network e.g. to customize the service paths in the network slice.

These properties cannot be met with pure overlay networks, as they require tighter coordination and integration between the underlay and the overlay network. This document introduces a new network service called enhanced VPN (VPN+). VPN+ refers to a virtual network which has dedicated network resources allocated from the underlay network. Unlike traditional VPN, an enhanced VPN can achieve greater isolation and guaranteed performance.

These new network layer properties, which have general applicability, may also be of interest as part of a network slicing solution.

This document specifies a framework for using the existing, modified and potential new networking technologies as components to provide an enhanced VPN (VPN+) service. Specifically we are concerned with:

- o The design of the enhanced VPN data-plane
- o The necessary protocols in both, underlay and the overlay of enhanced VPN, and
- o The mechanisms to achieve integration between overlay and underlay
- o The necessary method of monitoring an enhanced VPN
- o The methods of instrumenting an enhanced VPN to ensure that the required tenant Service Level Agreement (SLA) is maintained

The required layer structure necessary to achieve this is shown in Section 4.1.

One use for enhanced VPNs is to create network slices with different isolation requirements. Such slices may be used to provide different tenants of vertical industrial markets with their own virtual network

with the explicit characteristics required. These slices may be "hard" slices providing a high degree of confidence that the VPN+ characteristics will be maintained over the slice life cycle, or they may be "soft" slices in which case some degree of interaction may be experienced.

2. Requirements Language

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

3. Overview of the Requirements

In this section we provide an overview of the requirements of an enhanced VPN.

3.1. Isolation between Virtual Networks

The requirement is to provide both hard and soft isolation between the tenants/applications using one enhanced VPN and the tenants/applications using another enhanced VPN. Hard isolation is needed so that applications with exacting requirements can function correctly despite a flash demand being created on another VPN competing for the underlying resources. An example might be a network supporting both emergency services and public broadband multi-media services.

During a major incident the VPNs supporting these services would both be expected to experience high data volumes, and it is important that both make progress in the transmission of their data. In these circumstances the VPNs would require an appropriate degree of isolation to be able to continue to operate acceptably.

We introduce the terms hard (static) and soft (dynamic) isolation to cover cases such as the above. A VPN has soft isolation if the traffic of one VPN cannot be inspected by the traffic of another. Both IP and MPLS VPNs are examples of soft isolated VPNs because the network delivers the traffic only to the required VPN endpoints. However the traffic from one or more VPNs and regular network traffic may congest the network resulting in delays for other VPNs operating normally. The ability for a VPN to be sheltered from this effect is called hard isolation, and this property is required by some critical applications. Although these isolation requirements are triggered by the needs of 5G networks, they have general utility. In the remainder of this section we explore how isolation may be achieved in packet networks.

It is of course possible to achieve high degrees of isolation in the optical layer. However this is done at the cost of allocating resources on a long term basis and end-to-end basis. Such an arrangement means that the full cost of the resources must be borne by the service that is allocated the resources. On the other hand, isolation at the packet layer allows the resources to be shared amongst many services and only dedicated to a service on a temporary basis. This allows greater statistical multiplexing of network resources and amortizes the cost over many services, leading to better economy. However, the degree of isolation required by network slicing cannot easily be met with MPLS-TE packet LSPs as they guarantee long-term bandwidth, but not latency.

Thus some trade-off between the two approaches needs to be considered to provide the required isolation between virtual networks while still allows reasonable sharing inside each VPN.

The work of the IEEE project on Time Sensitive Networking is introducing the concept of packet scheduling where a high priority packet stream may be given a scheduled time slot thereby guaranteeing that it experiences no queuing delay and hence a reduced latency. However where no scheduled packet arrives its reserved time-slot is handed over to best effort traffic, thereby improving the economics of the network. Such a scheduling mechanism may be usable directly, or with extension to achieve isolation between multiple VPNs.

One of the key areas in which isolation needs to be provided is at the interfaces. If nothing is done the system falls back to the router queuing system in which the ingress places it on a selected output queue. Modern routers have quite sophisticated output queuing systems, traditionally these have not provided the type of scheduling system needed to support the levels of isolation needed for the applications that are the target of VPN+ networks. However some of the more modern approaches to queuing allow the construction of logical virtual channelized sub-interfaces (VCSI). With VCSIs there is only one physical interface, and routing sees a single adjacency, but the queuing system is used to provide virtual interfaces at various priorities. Sophisticated queuing systems of this type may be used to provide end-to-end virtual isolation between tenant's traffic in an otherwise homogeneous network.

[FLEXE] provides the ability to multiplex multiple channels over an Ethernet link in a way that provides hard isolation. However it is a only a link technology. When packets are received by the downstream node they need to be processed in a way that preserves that isolation. This in turn requires a queuing and forwarding implementation that preserves the isolation, such as a sliced hardware system, or an LVI system of the type described above.

3.2. Diverse Performance Guarantees

There are several aspects to guaranteed performance, guaranteed maximum packet loss, guaranteed maximum delay and guaranteed delay variation.

Guaranteed maximum packet loss is a common parameter, and is usually addressed by setting the packet priorities, queue size and discard policy. However this becomes more difficult when the requirement is combine with the latency requirement. The limiting case is zero congestion loss, and than is the goal of the Deterministic Networking work that the IETF and IEEE are pursuing. In modern optical networks loss due to transmission errors is already asymptotic to zero due, but there is always the possibility of failure of the interface and the fiber itself. This can only be addressed by some form of packet duplication and transmission over diverse paths.

Guaranteed maximum latency is required in a number of applications particularly real-time control applications and some types of virtual reality applications. The work of the IETF Deterministic Networking (DetNet) Working Group is relevant, however the scope needs to be extended to methods of enhancing the underlay to better support the delay guarantee, and to integrate these enhancements with the overall service provision.

Guaranteed maximum delay variation is a service that may also be needed. Time transfer is one example of a service that needs this, although the fungible nature of time means that it might be delivered by the underlay as a shared service and not provided through different virtual networks. Alternatively a dedicated virtual network may be used to provide this as a shared service. The need for guaranteed maximum delay variation as a general requirement is for further study.

This leads to the concept that there is a spectrum of grades of service guarantee that need to be considered when deploying and enhanced VPN. As a guide to understanding the design requirements we can consider four types:

- o Guaranteed latency,
- o Enhanced delivery
- o Assured bandwidth,
- o Best effort

In Section 3.1 we considered the work of the IEEE Time Sensitive Networking (TSN) project and the work of the IETF DetNet Working group in the context of isolation. However this work is of greater relevance in assuring end-to-end packet latency. It is also of importance in considering enhanced delivery.

A service that is guaranteed latency has a latency upper bound provided by the network. It is important to note that assuring the upper bound is more important than achieving the minimum latency.

A service that is offered enhanced delivery is one in which the network (at layer 3) attempts to deliver the packet through multiple paths in the hope of avoiding transient congestion [I-D.ietf-detnet-dp-sol].

A useful mechanism to provide these guarantees is to use Flex Ethernet [FLEXE] as the underlay. This is a method of bonding Ethernets together and of providing time-slot based channelization over an Ethernet bearer. Such channels are fully isolated from other channels running over the same Ethernet bearer. As noted elsewhere this produces hard isolation but at the cost of making the reclamation of unused bandwidth harder.

These approaches can usefully be used in tandem. It is possible to use FlexE to provide tenant isolation, and then to use the TSN approach over FlexE to provide service performance guarantee inside the a slice/tenant VPN.

3.3. A Pragmatic Approach to Isolation

A key question to consider is whether whether it is possible to achieve hard isolation in packet networks? Packet networks were never designed to support hard isolation, just the opposite, they were designed to provide a high degree of statistical multiplexing and hence a significant economic advantage when compared to a dedicated, or a Time Division Multiplexing (TDM) network. However the key thing to bear in mind is that the concept of hard isolation needs to be viewed from the perspective of the application, and there is no need to provide any harder isolation than is required by the application. From a historical perspective it is good to think about pseudowires [RFC3985] which emulate services that in many would have had hard isolation in their native form. However experience has shown that in most cases an approximation to this requirement is sufficient for most uses.

Thus, for example, using FlexE or channelized sub-interface, together with packet scheduling as interface slicing, and optionally, also together with the slicing of node resources (Network Processor Unit

(NPU), etc.), it may be possible to provide a type of hard isolation that is adequate for many applications. Other applications may be satisfied with a classical VPN and reserved bandwidth, but yet others may require dedicated point to point fiber. The requirement is thus to qualify the needs of each application and provide an economic solution that satisfies those needs without over-engineering.

3.4. Integration

A solution to the enhanced VPN problem will need to provide seamless integration of both Overlay VPN and the underlay network resources. This needs to be done in a flexible and scalable way so that it can be widely deployed in operator networks. Given the targeting of both this technology and service function chaining at mobile networks and in particular 5G the co-integration of service functions is a likely requirement.

3.5. Dynamic Configuration

It is necessary that new enhanced VPNs can be introduced to the network, modified, and removed from the network according to service demand. In doing so due regard must be given to the impact of other enhanced VPNs that are operational. An enhanced VPN that requires hard isolation must not be disrupted by the installation or modification of another enhanced VPN.

Whether modification of an enhanced VPN can be disruptive to that VPN, and in particular the traffic in flight is to be determined, but is likely to be a difficult problem to address.

The data-plane aspect of this are discussed further in Section 4.3.

The control-plane and management-plane aspects of this, particularly the garbage collection are likely to be challenging and are for further study.

As well as managing dynamic changes to the VPN in a seamless way, dynamic changes to the underlay and its transport network need to be managed in order to avoid disruption to sensitive services.

In addition to non-disruptively managing the network as a result of gross change such as the inclusion of a new VPN endpoint or a change to a link, consideration has to be given to the need to move VPN traffic as a result of traffic volume changes.

3.6. Customized Control Plane

In some cases it is desirable that an enhanced VPN has a custom control-plane, so that the tenant of the enhanced VPN can have some control to the resources and functions partitioned for this VPN. Each enhanced VPN may have its own dedicated controller, it may be provided with an interface to a control-plane that is shared with a set of other tenants, or it may be provided with an interface to the control-plane of the underlay provided by the underlay network operator.

Further detail on this requirement will be provided in a future version of the draft.

4. Architecture and Components of VPN+

Normally a number of enhanced VPN services will be provided by a common network infrastructure. Each enhanced VPN consists of both the overlay and a specific set of dedicated network resources and functions allocated in the underlay to satisfy the needs of the VPN tenant. The integration between overlay and underlay ensures the isolation and between different enhanced VPNs, and facilitates the guaranteed performance for different services.

An enhanced VPN needs to be designed with consideration given to:

- o Isolation of enhanced VPN data plane.
- o A scalable control plane to match the data plane isolation.
- o The amount of state in the packet vs the amount of state in the control plane.
- o Mechanism for diverse performance guarantee within an enhanced VPN
- o Support of the required integration between network functions and service functions.

4.1. Communications Layering

The communications layering model use to build an enhanced VPN is shown in Figure 1.

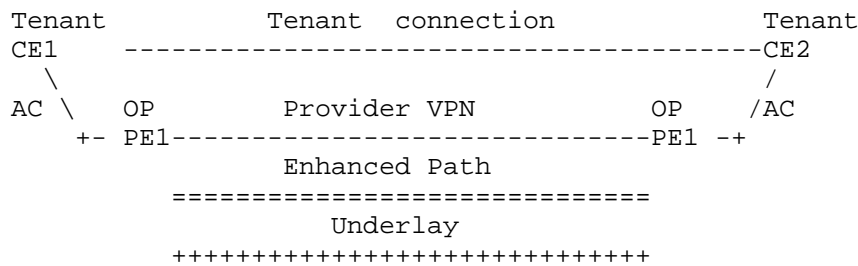


Figure 1: Communication Layering

The network operator is required to provide a tenant connection between the tenant's Customer Equipment (CE) (CE1 and CE2). These CEs attach to the Operator's Provider Edge Equipments (PE) (PE1 and PE2 respectively). The attachment circuits (AC) are outside the scope of this document other than to note that they obviously need to provide a connection of sufficient quality in terms of isolation, latency etc so as to satisfy the needs of the user. The subtlety to be aware of is that the ACs are often provided by a network rather than a fixed point to point connection and thus the considerations in this document may apply to the network that provides the AC.

A provider VPN is constructed between PE1 and PE2 to carry tenant traffic. This is a normal VPN, and provides one stage of isolation between tenants.

An enhanced path is constructed to carry the provider VPN using dedicated resources drawn from the underlay.

4.2. Multi-Point to Multi-point

At a VPN level connections are frequently multi-point-to-multi-point (MP2MP). As far as such services are concerned the underlay is also an abstract MP2MP medium. However when service guarantees are provided, such as with an enhanced VPN, each point to point path through the underlay needs to be specifically engineered to meet the required performance guarantees.

4.3. Candidate Underlay Technologies

A VPN is a network created by applying a multiplexing technique to the underlying network (the underlay) in order to distinguish the traffic of one VPN from that of another. A VPN path that travels by other than the shortest path through the underlay normally requires state in the underlay to specify that path. State is normally applied to the underlay through the use of the RSVP Signaling protocol, or directly through the use of an SDN controller, although

other techniques may emerge as this problem is studied. This state gets harder to manage as the number of VPN paths increases. Furthermore, as we increase the coupling between the underlay and the overlay to support the VPN which requires enhanced VPN service, this state will increase further.

In an enhanced VPN different subsets of the underlay resources are dedicated to different VPNs. Any enhanced VPN solution thus needs tighter coupling with underlay than is the case with classical VPNs. We cannot for example share the tunnel between enhanced VPNs which require hard isolation.

In the following sections we consider a number of candidate underlay solutions for proving the required VPN separation.

- o FlexE
- o Time Sensitive Networking
- o Deterministic Networking
- o Dedicated Queues

We then consider the problem of slice differentiation and resource representation. Candidate technologies are:

- o MPLS
- o MPLS-SR
- o Segment Routing over IPv6 (SRv6)

4.3.1. FlexE

FlexE [FLEXE] is a method of creating a point-to-point Ethernet with a specific fixed bandwidth. FlexE supports the bonding of multiple links, which supports creating larger links out of multiple slower links in a more efficient way than traditional link aggregation. FlexE also supports the sub-rating of links, which allows an operator to only use a portion of a link. FlexE also supports the channelization of links, which allows one link to carry several lower-speed or sub-rated links from different sources.

If different FlexE channels are used for different services, then no sharing is possible between the services. This in turn means that it is not possible to dynamically re-distribute unused bandwidth to lower priority services increasing the cost of operation of the network. FlexE can on the other hand be used to provide hard

isolation between different tenants by providing hard isolation on an interface. The tenant can then use other methods to manage the relative priority of their own traffic.

Methods of dynamically re-sizing FlexE channels and the implication for enhanced VPN are under study.

4.3.2. Dedicated Queues

In an enhanced VPN providing multiple isolated virtual networks the conventional Diff-Serv based queuing system is insufficient for our purposes due to the limited number of queues which cannot differentiate between traffic of different VPNs and the range of service classes that each need to provide their tenants. This problem is particularly acute with an MPLS underlay due to the small number of traffic class services available. In order to address this problem and thus reduce the interference between VPNs, it is likely to be necessary to steer traffic of VPNs to dedicated input and output queues.

4.3.3. Time Sensitive Networking

Time Sensitive Networking (TSN) is an IEEE project that is designing a method of carrying time sensitive information over Ethernet. As Ethernet this can obviously be tunneled over a Layer 3 network in a pseudowire. However the TSN payload would be opaque to the underlay and thus not treated specifically as time sensitive data. The preferred method of carrying TSN over a layer 3 network is through the use of deterministic networking as explained in the following section of this document.

The mechanisms defined in TSN can be used to meet the requirements of time sensitive services of an enhanced VPN.

4.3.4. Deterministic Networking

Deterministic Networking (DetNet) [I-D.ietf-detnet-architecture] is a technique being developed in the IETF to enhance the ability of layer 3 networks to deliver packets more reliably and with greater control over the delay. The design cannot use classical re-transmission techniques such as TCP since can add delay that is above the maximum tolerated by the applications. Even the delay improvements that are achieved with SCTP-PR are outside the bounds set by application demands. The approach is to pre-emptively send copies of the packet over various paths in the expectation that this minimizes the chance of all packets being lost, but to trim duplicate packets to prevent excessive flooding of the network and to prevent multiple packets being delivered to the destination. It also seeks to set an upper

bound on latency. Note that it is not the goal to minimize latency, and the optimum upper bound paths may not be the minimum latency paths.

DetNet is based on flows. It currently makes no comment on the underlay, and so at this stage must be assumed to use the base topology. To be of use in this application DetNet there needs to be a description of how to deal with the concept of flows within an enhanced VPN.

How we use DetNet in a multi-tenant (VPN) network, and how to improve the scalability of DetNet in a multi-tenant (VPN) network is for further study.

4.3.5. MPLS Traffic Engineering (MPLS-TE)

Normal MPLS runs on the base topology and has the concepts of reserving end to end bandwidth for an LSP, and of creating VPNs. VPN traffic can be run over RSVP-TE tunnels to provide reserved bandwidth for a specific VPN connection. This is rarely deployed in practice due to scaling and management overhead concerns.

4.3.6. Segment Routing

Segment Routing [I-D.ietf-spring-segment-routing] is a method that prepends instructions to packets at entry and sometimes at various points as it passes through the network. These instructions allow packets to be routed on paths other than the shortest path for various traffic engineering reasons. These paths can be strict or loose paths, depending on the compactness required of the instruction list and the degree of autonomy granted to the network (for example to support ECMP).

With SR, a path needs to be dynamically created through a set of resources by simply specifying the Segment IDs (SIDs), i.e. instructions rooted at a particular point in the network. Thus if a path is to be provisioned from some ingress point A to some egress point B in the underlay, A is provided with the A..B SID list and instructions on how to identify the packets to which the SID list is to be prepended.

By encoding the state in the packet, as is done in Segment Routing, state is transitioned out of the network.

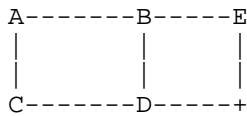


Figure 2: An SR Network Fragment

Consider the network fragment shown in Figure 2. To send a packet from A to E via B, D & E: Node A prepends the ordered list of SIDs: D, E to the packet and pushes the packet to B. SID list {B, D, E} can be used as a VPN path. Thus, to create a VPN, a set of SID Lists is created and provided to each ingress node of the VPN together with packet selection criteria. In this way it is possible to create a VPN with no state in the core. However this is at the expense of creating a larger packet with possible MTU and hardware restriction limits that need to be overcome.

Note in the above if A and E support multiple VPN an additional VPN identifier will need to be added to the packet, but this is omitted from this text for simplicity.

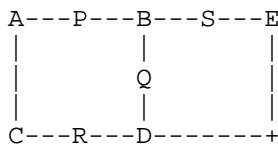


Figure 3: Another SR Network Fragment

Consider a further network fragment shown in Figure 3, and further consider VPN A+D+E.

A has lists: {P, B, Q, D}, {P, B, S, E}
 D has lists: {Q, B, P, A}, {E}
 E has lists: {S, B, P, A}, {D}

To create a new VPN C+D+B the following list are introduced:

C lists: {R, D}, {A, P, B}
 D lists: {R, C}, {Q, B}
 B lists: {Q, D}, {P, A, C}

Thus VPN C+D+B was created without touching the settings of the core routers, indeed it is possible to add endpoints to the VPNs, and move the paths around simply by providing new lists to the affected endpoints.

There are a number of limitations in SR as it is currently defined that limit its applicability to enhanced VPNs:

- o Segments are shared between different VPNs,
- o There is no reservation of bandwidth,
- o There is limited differentiation in the data plane.

Thus some extensions to SR are needed to provide isolation between different enhanced VPNs. This can be achieved by including a finer granularity of state in the core in anticipation of its future use by authorized services. We therefore need to evaluate the balance between this additional state and the performance delivered by the network.

Both MPLS Segment Routing and SRv6 Segment Routing are candidate technologies for enhanced VPN.

With current segment routing, the instructions are used to specify the nodes and links to be traversed. However, in order to achieve the required isolation between different services, new instructions can be created which can be prepended to a packet to steer it through specific dedicated network resources and functions, e.g. links, queues, processors, services etc.

Clearly we can use traditional constructs to create a VPN, but there are advantages to the use of other constructs such as Segment Routing (SR) in the creation of virtual networks with enhanced properties.

Traditionally a traffic engineered path operates with a granularity of a link with hints about priority provided through the use of the traffic class field in the header. However to achieve the latency and isolation characteristics that are sought by VPN+ users, steering packets through specific queues resources will likely be required. The extent to which these needs can be satisfied through existing QoS mechanisms is to be determined. What is clear is that a fine control of which services wait for which, with a fine granularity of queue management policy is needed. Note that the concept of a queue is a useful abstraction for many types of underlay mechanism that may be used to provide enhanced latency support. From the perspective of the control plane and from the perspective of the segment routing the method of steering a packet to a queue that provides the required properties is a universal construct. How the queue satisfies the requirement is outside the scope of these aspect of the enhanced VPN system. Thus for example a FlexE channel, or time sensitive networking packet scheduling slot are abstracted to the same concept and bound to the data plane in a common manner.

We can introduce the specification of finer, deterministic, granularity to path selection through extensions to traditional path construction techniques such as RSVP-TE and MPLS-TP.

We can also introduce it by specifying the queue through an SR instruction list. Thus new SR instructions may be created to specify not only which resources are traversed, but in some cases how they are traversed. For example, it may be possible to specify not only the queue to be used but the policy to be applied when enqueueing and dequeuing.

This concept can be further generalized, since as well as queuing to the output port of a router, it is possible to queue to any resource, for example:

- o A network processor unit (NPU)
- o A Central Processing Unit (CPU) Core
- o A Look-up engine such as TCAMs

4.4. Control Plane Considerations

It is expected that VPN+ would be based on a hybrid control mechanism, which takes advantage of the logically centralized controller for on-demand provisioning and global optimization, whilst still relies on distributed control plane to provide scalability, high reliability, fast reaction, automatic failure recovery etc. Extension and optimization to the distributed control plane is needed to support the enhanced properties of VPN+.

Where SR is used as a the data-plane construct it needs to be noted that it does not have the capability of reserving resources along the path nor do its currently specified distributed control plane (the link state routing protocols). An SDN controller can clearly do this, from the controllers point of view, and no resource reservation is done on the device. Thus if a distributed control plane is needed either in place of an SDN controller or as an assistant to it, the design of the control system needs to ensure that resources are uniquely allocated to the correct service, and no allocated to multiple services causing unintended resource conflict. This needs further study.

On the other hand an advantage of using an SR approach is that it provides a way of efficiently binding the network underlay and the enhanced VPN overlay. With a technology such as RSVP-TE LSPs, each virtual path in the VPN is bound to the underlay with a dedicated TE-LSP.

RSVP-TE could be enhanced to bind the VPN to specific resources within the underlay, but as noted elsewhere in this document there are concerns as to the scalability of this approach. With an SR-based approach to resource reservation (per-slice reservation), it is straightforward to create dedicated SR network slices, and the VPN can be bound to a particular SR network slice.

4.5. Application Specific Network Types

Although a lot of the traffic that will be carried over the enhanced VPN will likely be IPv4 or IPv6, the design has to be capable of carrying other traffic types. In particular the design SHOULD be capable of carrying Ethernet traffic. This is easily accomplished through the various pseudowire (PW) techniques [RFC3985]. Where the underlay is MPLS Ethernet can be carried over the enhanced VPN encapsulated according to the method specified in [RFC4448]. Where the underlay is IP Layer Two Tunneling Protocol - Version 3 (L2TPv3) [RFC3931] can be used with Ethernet traffic carried according to [RFC4719]. Encapsulations have been defined for most of the common layer two type for both PW over MPLS and for L2TPv3.

4.6. Integration with Service Functions

There is a significant overlap between the problem of routing a packet through a set of network resources and the problem of routing a packet through a set of compute resources. Service Function Chain technology is designed to forward a packet through a set of compute resources.

A future version of this document will discuss this further.

5. Scalability Considerations

For a packet to transit a network, other than on a best effort, shortest path basis, it is necessary to introduce additional state, either in the packet, or in the network or some combination of both.

There are at least three ways of doing this:

- o Introduce the complete state into the packet. That is how SR does this, and this allows the controller to specify the precise series of forwarding and processing instructions that will happen to the packet as it transits the network. The cost of this is an increase in the packet header size. The cost is also that systems will have capabilities enabled in case they are called upon by a service. This is a type of latent state, and increases as we more precisely specify the path and resources that need to be exclusively available to a VPN.

- o Introduce the state to the network. This is normally done by creating a path using RSVP-TE, which can be extended to introduce any element that needs to be specified along the path, for example explicitly specifying queuing policy. It is of course possible to use other methods to introduce path state, such as via a Software Defined Network (SDN) controller, or possibly by modifying a routing protocol. With this approach there is state per path per path characteristic that needs to be maintained over its life-cycle. This is more state than is needed using SR, but the packet are shorter.
- o Provide a hybrid approach based on using binding SIDs to create path fragments, and bind them together with SR.

Dynamic creation of a VPN path using SR requires less state maintenance in the network core at the expense of larger VPN headers on the packet. The scaling properties will reduce roughly from a function of $(N/2)^2$ to a function of N , where N is the VPN path length in intervention points (hops plus network functions). Reducing the state in the network is important to VPN+, as VPN+ requires the overlay to be more closely integrated with the underlay than with traditional VPNs. This tighter coupling would normally mean that significant state needed to be created and maintained in the core. However, a segment routed approach allows much of this state to be spread amongst the network ingress nodes, and transiently carried in the packets as SIDs.

These approaches are for further study.

5.1. Maximum Stack Depth

One of the challenges with SR is the stack depth that nodes are able to impose on packets. This leads to a difficult balance between adding state to the network and minimizing stack depth, or minimizing state and increasing the stack depth.

5.2. RSVP scalability

The traditional method of creating a resource allocated path through an MPLS network is to use the RSVP protocol. However there have been concerns that this requires significant continuous state maintenance in the network. There are ongoing works to improve the scalability of RSVP-TE LSPs in the control plane [I-D.ietf-teas-rsvp-te-scaling-rec]. This will be considered further in a future version of this document.

There is also concern at the scalability of the forwarder footprint of RSVP as the number of paths through an LSR grows

[I-D.sitaraman-mpls-rsvp-shared-labels] proposes to address this by employing SR within a tunnel established by RSVP-TE. This work will be considered in a future version of this document.

6. OAM and Instrumentation

A study of OAM in SR networks has been documented in [I-D.ietf-spring-oam-usecase].

The enhanced VPN OAM design needs to consider the following requirements:

- o Instrumentation of the underlay so that the network operator can be sure that the resources committed to a tenant are operating correctly and delivering the required performance.
- o Instrumentation of the overlay by the tenant. This is likely to be transparent to the network operator and to use existing methods. Particular consideration needs to be given to the need to verify the isolation and the various committed performance characteristics.
- o Instrumentation of the overlay by the network provider to proactively demonstrate that the committed performance is being delivered. This needs to be done in a non-intrusive manner, particularly when the tenant is deploying a performance sensitive application
- o Verification of the conformity of the path to the service requirement. This may need to be done as part of a commissioning test.

These issues will be discussed in a future version of this document.

7. Enhanced Resiliency

Each enhanced VPN, of necessity, has a life-cycle, and needs modification during deployment as the needs of its user change. Additionally as the network as a whole evolves there will need to be garbage collection performed to consolidate resources into usable quanta.

Systems in which the path is imposed such as SR, or some form of explicit routing tend to do well in these applications because it is possible to perform an atomic transition from one path to another. However implementations and the monitoring protocols need to make sure that the new path is up before traffic is transitioned to it.

There are however two manifestations of the latency problem that are for further study in any of these approaches:

- o The problem of packets overtaking one and other if a path latency reduces during a transition.
- o The problem of the latency transient in either direction as a path migrates.

There is also the matter of what happens during failure in the underlay infrastructure. Fast reroute is one approach, but that still produces a transient loss with a normal goal of rectifying this within 50ms. An alternative is some form of N+1 delivery such as has been used for many years to support protection from service disruption. This may be taken to a different level using the techniques proposed by the IETF deterministic network work with multiple in-network replication and the culling of later packets.

In addition to the approach used to protect high priority packets, consideration has to be given to the impact of best effort traffic on the high priority packets during a transient. Specifically if a conventional re-convergence process is used there will inevitably be micro-loops and whilst some form of explicit routing will protect the high priority traffic, lower priority traffic on best effort shortest paths will micro-loop without the use of a loop prevention technology. To provide the highest quality of service to high priority traffic, either this traffic must be shielded from the micro-loops, or micro-loops must be prevented.

8. Security Considerations

All types of virtual network require special consideration to be given to the isolation between the tenants. However in an enhanced virtual network service hard isolation needs to be considered. If a service requires a specific latency then it can be damaged by simply delaying the packet through the activities of another tenant. In a network with virtual functions, depriving a function used by another tenant of compute resources can be just as damaging as delaying transmission of a packet in the network.

9. IANA Considerations

There are no requested IANA actions.

10. References

10.1. 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, <<https://www.rfc-editor.org/info/rfc2119>>.

10.2. Informative References

- [FLEXE] "Flex Ethernet Implementation Agreement", March 2016, <<http://www.oiforum.com/wp-content/uploads/OIF-FLEXE-01.0.pdf>>.
- [I-D.ietf-detnet-architecture] Finn, N., Thubert, P., Varga, B., and J. Farkas, "Deterministic Networking Architecture", draft-ietf-detnet-architecture-04 (work in progress), October 2017.
- [I-D.ietf-detnet-dp-sol] Korhonen, J., Andersson, L., Jiang, Y., Finn, N., Varga, B., Farkas, J., Bernardos, C., Mizrahi, T., and L. Berger, "DetNet Data Plane Encapsulation", draft-ietf-detnet-dp-sol-01 (work in progress), January 2018.
- [I-D.ietf-spring-oam-usecase] Geib, R., Filsfils, C., Pignataro, C., and N. Kumar, "A Scalable and Topology-Aware MPLS Dataplane Monitoring System", draft-ietf-spring-oam-usecase-10 (work in progress), December 2017.
- [I-D.ietf-spring-segment-routing] Filsfils, C., Previdi, S., Ginsberg, L., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", draft-ietf-spring-segment-routing-15 (work in progress), January 2018.
- [I-D.ietf-teas-rsvp-te-scaling-rec] Beeram, V., Minei, I., Shakir, R., Pacella, D., and T. Saad, "Techniques to Improve the Scalability of RSVP Traffic Engineering Deployments", draft-ietf-teas-rsvp-te-scaling-rec-09 (work in progress), February 2018.

- [I-D.sitaraman-mpls-rsvp-shared-labels]
Sitaraman, H., Beeram, V., Parikh, T., and T. Saad,
"Signaling RSVP-TE tunnels on a shared MPLS forwarding
plane", draft-sitaraman-mpls-rsvp-shared-labels-03 (work
in progress), December 2017.
- [NETCALC] "Applicability of Network Calculus to DetNet", November
2017, <[https://datatracker.ietf.org/meeting/100/materials/
slides-100-detnet-applicability-of-network-calculus-to-
detnet](https://datatracker.ietf.org/meeting/100/materials/slides-100-detnet-applicability-of-network-calculus-to-detnet)>.
- [RFC3931] Lau, J., Ed., Townsley, M., Ed., and I. Goyret, Ed.,
"Layer Two Tunneling Protocol - Version 3 (L2TPv3)",
RFC 3931, DOI 10.17487/RFC3931, March 2005,
<<https://www.rfc-editor.org/info/rfc3931>>.
- [RFC3985] Bryant, S., Ed. and P. Pate, Ed., "Pseudo Wire Emulation
Edge-to-Edge (PWE3) Architecture", RFC 3985,
DOI 10.17487/RFC3985, March 2005,
<<https://www.rfc-editor.org/info/rfc3985>>.
- [RFC4448] Martini, L., Ed., Rosen, E., El-Aawar, N., and G. Heron,
"Encapsulation Methods for Transport of Ethernet over MPLS
Networks", RFC 4448, DOI 10.17487/RFC4448, April 2006,
<<https://www.rfc-editor.org/info/rfc4448>>.
- [RFC4719] Aggarwal, R., Ed., Townsley, M., Ed., and M. Dos Santos,
Ed., "Transport of Ethernet Frames over Layer 2 Tunneling
Protocol Version 3 (L2TPv3)", RFC 4719,
DOI 10.17487/RFC4719, November 2006,
<<https://www.rfc-editor.org/info/rfc4719>>.

Authors' Addresses

Stewart Bryant
Huawei

Email: stewart.bryant@gmail.com

Jie Dong
Huawei

Email: jie.dong@huawei.com

Zhenqiang Li
China Mobile

Email: lizhenqiang@chinamobile.com

Takuya Miyasaka
KDDI Corporation

Email: ta-miyasaka@kddi.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: November 10, 2017

M. Bjorklund
Tail-f Systems
J. Schoenwaelder
Jacobs University
P. Shafer
K. Watsen
Juniper Networks
R. Wilton
Cisco Systems
May 9, 2017

Guidelines for YANG Module Authors (NMDA)
draft-dsdt-nmda-guidelines-01

Abstract

The "Network Management Datastore Architecture" (NMDA) adds the ability to inspect the current operational values for configuration, allowing clients to use identical paths for retrieving the configured values and the operational values. This change will simplify models and help modelers, but will create a period of transition as NMDA becomes a standard and is widely implemented. During that interim, the guidelines given in this document should help modelers find an optimal path forward.

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 November 10, 2017.

Copyright Notice

Copyright (c) 2017 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. Keywords	2
1.2. Terminology	2
1.3. Executive Summary	3
1.4. Background	3
1.5. Network Management Datastores Architecture	5
2. Guidelines for YANG Modelers	5
3. IANA Considerations	8
4. Security Considerations	8
5. Informative References	8
Authors' Addresses	9

1. Introduction

This document provides advice and guidelines to help modelers plan for the emerging "Network Management Datastore Architecture" (NMDA) [I-D.ietf-netmod-revised-datastores]. This architecture provides an architectural framework for datastores as they are used by network management protocols such as NETCONF [RFC6241], RESTCONF [RFC8040] and the YANG [RFC7950] data modeling language. Datastores are a fundamental concept binding network management data models to network management protocols, enabling data models to be written in a network management protocol agnostic way.

1.1. Keywords

The keywords "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].

1.2. Terminology

This document uses the terminology defined by the NMDA [I-D.ietf-netmod-revised-datastores].

1.3. Executive Summary

The Network Management Datastore Architecture (NMDA) addresses the so called "OpState problem" that has been the subject of much discussion in the IETF. NMDA is still in development and there will be a transition period before NMDA solutions are universally available.

These guidelines are aimed to enable the creation of models that can take advantage of the NMDA, while pragmatically allowing those models to be used with the existing network configuration protocol implementations.

It is the strong recommendation that models SHOULD move as quickly as possible to the NMDA. The specific approach to be taken for models being developed now and during the NMDA transition period should be based on both the expected usage and the maturity of the data model.

1. New models and models that are not concerned with the operational state of configuration information SHOULD immediately be structured to be NMDA-compatible.
2. Models that require immediate support for "in use" and "system created" information SHOULD be structured for NMDA. A non-NMDA version of these models SHOULD also be published, using either an existing model or a model created either by hand or with suitable tools that support current modeling strategies. Both the NMDA and the non-NMDA modules SHOULD be published in the same document, with NMDA modules in the document main body and the non-NMDA modules in a non-normative appendix. The use of the non-NMDA model will allow temporary bridging of the time period until NMDA implementations are available.

Additional details on these guidelines can be found below, notably in Section 2.

1.4. Background

NETCONF ([RFC6241]) was developed with a focus on configuration data, and has unfortunate gaps in its treatment of operational data. The <get-config> operation can return configuration data (defined as nodes with "config true") stored in <running>. This data is typically the only data created by CLI users and NETCONF clients. The <get> operation is defined as returning all the data on the device, including the contents of <running>, as well as any operational state data. While the NETCONF design envisioned models merging "config false" nodes with the contents of running, there are two issues involved.

First, the desire of clients to see the true operational ("in use") value of configuration data resulted in the need for data models to have two distinct leafs, one to show the configured value and the other to show the operational value. An example would be the speed of an interface, where the configured value may not be the value that is currently used.

Second, devices often have "system created" resources that exist as operational data even when there is no corresponding configuration data. An example would be built-in networking interfaces that always appear in operational data.

A similar situation to the second issue discussed above exists while the device is processing configuration data changes. When configuration data is deleted, the operational data will continue to exist during the time period in which the device is releasing resources associated with the data. An example would be deleting a BGP peer, where the peer continues to exist in operational data until the connection is closed and any other resources are released.

To address these issues without requiring a protocol modification, two distinct strategies have been adopted in YANG model design:

The first strategy makes two distinct top-level containers, one for configuration and one for state. These are sometimes referred to as `"/foo"` and `"/foo-state"`. An example would be the interface model defined in [RFC7223]. These models require two completely distinct set of nodes, with repetition of both the interior containers, lists, and key nodes, but also repetition of many other nodes to allow visibility of the operational values of configured nodes. This leads to over-use of YANG groupings in ways that affect the readability of the models, as well as creating opportunities to incorrectly mirror the model's hierarchies. Also this "stitching together" of data from the two trees is merely a convention, not a formal relationship.

The second strategy uses two sibling containers, named `"config"` and `"state"`, placed deeper within the model node hierarchy. The `"config"` container holds the configured values, while the `"state"` container holds the operational values. The duplication of interior nodes in the hierarchies is removed, but the duplication of leafs representing configuration remains. Groupings can be used to avoid the repetition of the leafs in the YANG file, but at the expense of readability. In addition, this strategy does not handle the existence of operational data for which there is no configuration data, such as the system-created data and deleted peers scenarios discussed above.

1.5. Network Management Datastores Architecture

The Network Management Datastores Architecture (NMDA) addresses the problems mentioned above by creating an architectural framework which includes a distinct datastore for operational data, called `<operational>`. This datastore is defined as containing both config true and config false nodes, with the formal understanding that the "in use" values are returned for the config true nodes. This allows modelers to use a single hierarchy for all configuration and operational data, which both improves readability and reduces the possibility of modeling inconsistencies that might impact programmatic access.

In addition, another datastore named `<intended>` is defined to provide a complete view of the configuration data, even in the presence of device-specific features that expand or remove configuration data. While such mechanisms are currently non-standard, the NMDA recognizes they exist and need to be handled appropriately. In the future, such mechanisms may become standardized.

The NMDA allows the deprecation of NETCONF's `<get>` operation, removing the source of these issues. The new operations `<get-data>` and `<edit-data>` will support a parameter indicating the target datastore. Similar changes are planned for RESTCONF ([RFC8040]).

2. Guidelines for YANG Modelers

The following guidelines are meant to help modelers develop YANG models that will maximize the utility of the model with both current implementations and NMDA-capable implementations. Any questions regarding these guidelines can be sent to yang-doctors@ietf.org.

The direction taken should be based on both the maturity of the data model, along with the number of concrete implementations of the model. The intent is not to destabilize the IETF modeling community, but to create models that can take advantage of the NMDA, while pragmatically allowing those models to be used with the existing network configuration protocol implementations.

It is the strong recommendation that models SHOULD move as quickly as possible to the NMDA. This is key to the future of these models. The NETMOD WG will rework existing models to this architecture. Given the permanence and gravity of work published by the IETF, creating future-proof data models is vital.

The two current strategies ("`/foo-state`" and "`config/state`" containers) mix data retrieval details into the data model, complicating the models and impairing their readability. Rather than

maintain these details inside the data model, models can be post-processed to add this derivative information, either manually or using tools.

Tools can automatically produce the required derived modules. The suggested approach is to produce a "state" version of the module with a distinct namespace, rather than using the "/foo-state" top-level container. Since the contents are identical, constraints in the data model such as "must" statements should not need to change. Only the model name, namespace, and prefix should need to change. This simplifies the tooling needed to generate the derived model, as well as reducing changes needed in client applications when transitioning to the NMDA model.

These derived models use distinct module names and namespaces, allowing servers to announce their support for the base or derived models.

Consider the following trivial model:

```
module example-thermostat {
  namespace "tag:ietf:example:thermostat";
  prefix "thermo";

  container thermostat {
    leaf high-temperature {
      description "High temperature threshold";
      type int;
    }
    leaf low-temperature {
      description "Low temperature threshold";
      type int;
    }
    leaf current-temperature {
      description "Current temperature reading";
      type int;
      config false;
    }
  }
}
```

In the derived model, the contents mirror the NMDA data model, but are marked as "config false", and the module name and namespace values have a "-state" suffix:


```
module example-thermostat-state {
  namespace "tag:ietf:example:thermostat-state";
  prefix "thermo-state";

  container thermostat {
    config false;
    leaf high-temperature {
      description "High temperature threshold";
      type int;
    }
    leaf low-temperature {
      description "Low temperature threshold";
      type int;
    }
    leaf current-temperature {
      description "Current temperature reading";
      type int;
    }
  }
}
```

By adopting a tools-based solution for supporting models that are currently under development, models can be quickly restructured to be NMDA-compatible while giving continuity to their community of developers. When NMDA-capable implementations become available, the base data models can be used directly.

Modelers and reviewers can view the simple data model, published in the body of document. Tools can generate any required derived models, and those models can be published in a non-normative appendix to allow interoperability.

It is critical to consider the following guidelines, understanding that our goal is to make models that will see continued use in the long term, balancing short term needs against a desire for consistent, usable models in the future:

(a) New models and models that are not concerned with the operational state of configuration information SHOULD immediately be structured to be NMDA-compatible.

(b) Models that require immediate support for "in use" and "system created" information SHOULD be structured for NMDA. A non-NMDA version of these models SHOULD exist, either an existing model or a model created either by hand or with suitable tools that mirror the current modeling strategies. Both the NMDA and the non-NMDA modules SHOULD be published in the same document, with NMDA modules in the document main body and the non-NMDA modules in a non-normative

appendix. The use of the non-NMDA model will allow temporary bridging of the time period until NMDA implementations are available.

(c) For published models, the model should be republished with an NMDA-compatible structure, deprecating non-NMDA constructs. For example, the "ietf-interfaces" model in [RFC7223] will be restructured as an NMDA-compatible model. The "/interfaces-state" hierarchy will be marked "status deprecated". Models that mark their "/foo-state" hierarchy with "status deprecated" will allow NMDA-capable implementations to avoid the cost of duplicating the state nodes, while enabling non-NMDA-capable implementations to utilize them for access to the operational values.

(d) For models that augment models which have not been structured with the NMDA, the modeler will have to consider the structure of the base model and the guidelines listed above. Where possible, such models should move to new revisions of the base model that are NMDA-compatible. When that is not possible, augmenting "state" containers SHOULD be avoided, with the expectation that the base model will be re-released with the state containers marked as deprecated. It is RECOMMENDED to augment only the "/foo" hierarchy of the base model. Where this recommendation cannot be followed, then any new "state" elements SHOULD be included in their own module.

3. IANA Considerations

This document has no actions for IANA.

4. Security Considerations

This document has no security considerations.

5. Informative References

[I-D.ietf-netmod-revised-datastores]

Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture", draft-ietf-netmod-revised-datastores-01 (work in progress), March 2017.

[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>>.

- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<http://www.rfc-editor.org/info/rfc7223>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<http://www.rfc-editor.org/info/rfc8040>>.

Authors' Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Juergen Schoenwaelder
Jacobs University

Email: j.schoenwaelder@jacobs-university.de

Phil Shafer
Juniper Networks

Email: phil@juniper.net

Kent Watsen
Juniper Networks

Email: kwatsen@juniper.net

Rob Wilton
Cisco Systems

Email: rwilton@cisco.com

NFVRG
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2018

Fangwei. Hu
RongRong. Hua
ZTE Corporation
Shujun. Hu
Lu. Huang
China Mobile
July 2, 2017

YANG Data Model for Configuration Interface of Control-Plane and User-
Plane separation BNG
draft-hu-opsawg-cu-separation-yang-model-00.txt

Abstract

This document defines the YANG data model for operation management of Control-Plane and User-Plane separation BNG.

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 3, 2018.

Copyright Notice

Copyright (c) 2017 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. Concept and Terminology	4
2.1. Terminology	4
3. Information model	4
3.1. overview	4
3.2. vBNG interface configuration	4
3.3. Control channel configuration	5
3.4. Acl Configuration	5
3.5. QoS Configuration	6
4. vBNG YANG Data Model	6
5. Security Considerations	11
6. Acknowledgements	11
7. IANA Considerations	11
8. References	11
8.1. Normative References	11
8.2. Informative References	11
Authors' Addresses	12

1. Introduction

Cloud-based BNG with C/U separated conception is raised by [I-D.gu-nfvrg-cloud-bng-architecture]. The main idea of Control-Plane and User-Plane separation method is to extract and centralize the user management functions of multiple BNG devices, forming an unified and centralized control plane (CP), while the traditional router's Control Plane and forwarding plane are both preserved on BNG devices in the form of a user plane (UP).

The architecture of C/U separated BNG is shown as the following figure[I-D.huang-nvo3-vxlan-extension-for-vbras].

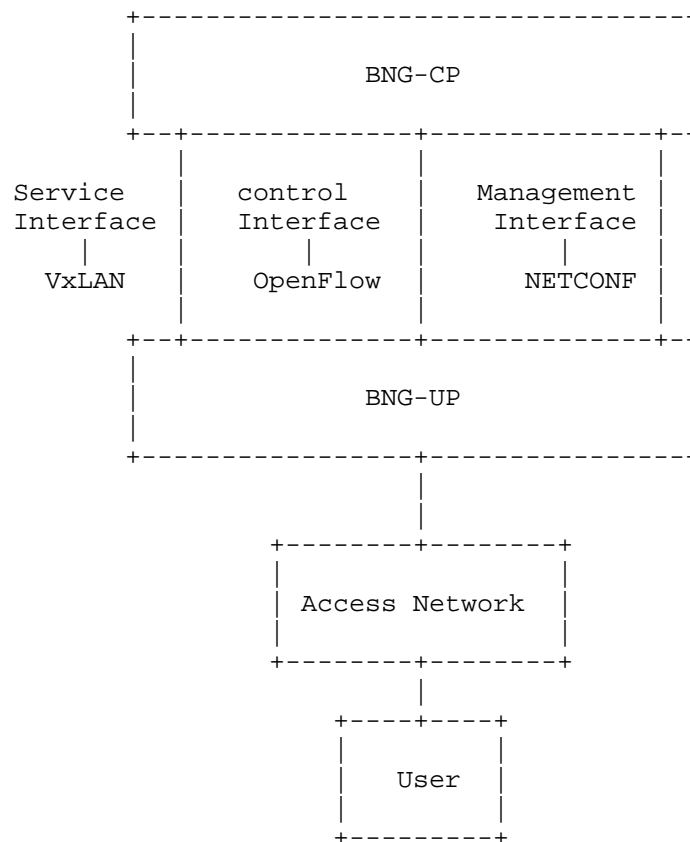


Figure 1: Architecture of C/U separated vBNG

There are three interfaces between BNG-CP and BNG-UP: Service interface, control interface and management interface. The service interface is used to carry PPPoE/IPoE dialup packets between user plane and control plane. The requirement and possible solution is defined in the [I-D.huang-nvo3-vxlan-extension-for-vbras]. Control interface is used for setting forwarding entries of user plane through OpenFlow or other protocols [I-D.wcg-i2rs-cu-separation-infor-model]. Management interface is used by CP to carry out basic configurations of user plane through NETCONF. The YANG data model about the configuration information is defined in this document.

Though BNG-CP and BNG-UP are connected with network management, most of the configuration information for BNG-UP are through the BNG-CP by netconf protocol[RFC6241], which simplifies the implementation of BNG-UP in the C/U separated BNG architecture.

Very few configuration parameters (such as IP address and port number for netconf protocol) for BNG-UP are configured through the network management directly.

2. Concept and 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 [RFC2119].

2.1. Terminology

BNG: Broadband Network Gateway. A broadband remote access server routes traffic to and from broadband remote access devices such as digital subscriber line access multiplexers (DSLAM) on an Internet service provider's (ISP) network.

CP: Control Plane. The CP is a user control management component which support to manage UP's resources such as the user entry and forwarding policy.

UP: User Plane. UP is a network edge and user policy implementation component.

3. Information model

3.1. overview

The vBNG UP or CP part can be a physical or logical network element. We augment [I-D.ietf-rtgwg-lne-model] to define the information model for vBNG CP and UP.

```
module: ietf-vbng
  augment /lne:logical-network-elements/lne:logical-network-element:
    +--rw ietf-vbng
      +--rw vbng-name?          string
      +--rw enable ?           boolean
```

3.2. vBNG interface configuration

The vBNG interface configuration is to configure the basic interface informations of vBNG UP element, such as interface name, the VLAN parameters for the sub-interface.

The tree structure for vBNG interface configuration is as following:

```

+--rw interfaces
|   +--rw interface* [name]
|       +--rw name          if:interface-ref
|       +--rw ethernet
|       |   +--rw lacp?      boolean
|       +--rw mac-offset?    uint32
|       +--rw vlans
|           +--rw tag* [index]
|               +--rw index      uint8
|               +--rw tag
|                   +--rw tag-type?    string
|                   +--rw vlan-id?     vlan-id

```

3.3. Control channel configuration

The control channel configuration is to configure the OpenFlow channel parameters and the VXLAN tunnel parameters.

The OpenFlow channel parameters include: ofls-name, dpid, of-port. The tree structure for OpenFlow channel configuration parameters are as following:

```

+--rw openflow-channel
|   +--rw ofls-name?    string
|   +--rw dpid?         uint32
|   +--rw of-port?      uint32

```

The static VXLAN tunnel is suggested to be used for vBNG CP and UP. The VXLAN tunnel parameters include: tunnel-source-ip, tunnel-destination-ip, vxlan-id, vxlan-tunnel-id, vxlan-tunnel-name, etc.

```

+--rw vxlan-channel* [vxlan-tunnel-id]
|   +--rw vxlan-tunnel-id      uint32
|   +--rw vxlan-tunnel-name?    string
|   +--rw address-family* [af]
|       +--rw af                                address-family-type
|       +--rw tunnel-source-ip?                  address-family-type
|       +--rw tunnel-destination-ip?              address-family-type
|       +--rw bind-vxlan-id* [vxlan-id]
|           +--rw vxlan-id      vxlan-id

```

3.4. Acl Configuration

The acl information for BNG-UP is configured through netconf from BNG-CP. The ACL information includes ipv4-acl, ipv6-acl, link-acl, etc. The YANG data model for ACL can refer to [I-D.ietf-netmod-acl-model]

3.5. QoS Configuration

The QoS information for BNG-UP is also configured through netconf from BNG-CP. The vBNG support QoS information includes IP-DSCP, MPLS, VPLS, VPWS etc. The YANG data model for QoS refer to [I-D.asechoud-rtgwg-qos-model]

4. vBNG YANG Data Model

```
<CODE BEGINS> file "ietf-vbng@2017-06-29.yang"
module ietf-vbng{
  namespace "urn:ietf:params:xml:ns:yang:ietf-vbng";
  prefix "vbng";

  import ietf-interfaces {
    prefix if;
  }

  import ietf-logical-network-element {
    prefix lne;
  }

/*
  import ietf-yang-types {
    prefix yang;
  }
*/

  organization
    "IETF NETCONF Working Group";

  contact
    "
      WG List:  <mailto:netconf@ietf.org>

      Editor:   Fangwei Hu
                <mailto:hu.fangwei@zte.com.cn>

    ";

  description
    "The YANG module defines a generic configuration
      model for vbng";

  revision 2017-06-29 {
    description "Initial revision";
```

```
reference
  "draft-hu-opsawg-cu-separation-yang-model-00";
}

/* Typedefs */

typedef vlan-id {
  type uint16 {
    range "0..4094";
  }
  description
    "Typedef for VLAN ID.";
}

typedef vxlan-id {
  type uint32;
  description
    "Typedef for VxLAN ID.";
}

typedef address-family-type {
  type enumeration {
    enum ipv4 {
      description
        "IPv4";
    }
    enum ipv6 {
      description
        "IPv6";
    }
  }
  description
    "Typedef for address family type.";
}

/* Configuration Data */

augment /lne:logical-network-elements/lne:logical-network-element {
  container ietf-vbng{
    leaf vbng-name {
      type string;
      description "configure vbng name";
    }

    leaf enable {
      type boolean;
    }
  }
}
```

```

        description "'true' to support vbng control plane and user pla
ne separation";
    }

    container interfaces {
        list interface {
            key name;
            leaf name {
                type if:interface-ref;
                description "interface name";
            }
            container ethernet {
                leaf lacp {
                    type boolean;
                    description "enable lacp function";
                }
                description "configure ethernet interface";
            }
            leaf mac-offset {
                type uint32;
                description "configure mac offset";
            }

            container vlans {
                list tag {
                    key index;
                    max-elements 2;

                    leaf index {
                        type uint8 {
                            range "0..1";
                        }

                        must ". = 0 or
count(..../tag[index = 0]/index) > 0" {
                            error-message "An inner tag can only be specified if an
                                outer tag has also been specified";
                            description "Ensure that an inner tag cannot be
                                specified without an outer tag'";
                        }
                    }

                    description "The index into the tag stack, outermost tag
                        assigned index 0";
                }

                container tag{
                    leaf tag-type {
                        type string;
                        description "tag type";
                    }
                }
            }
        }
    }

```

```

        leaf vlan-id {
            type vlan-id;
            description "vlan id value";
        }

        description "tag";
    }
    description "tag list";
}
description "vlans";
}
description "interfaces list";
}
description "interface container";
}

    container openflow-channel {
        leaf ofls-name {
            type string;
            description "openflow logical name";
        }
        leaf dpid {
            type uint32;
            description "dpid value";
        }
        leaf of-port {
            type uint32;
            description "openflow channel udp port number";
        }
        description "configure openflow channel value";
    }

list vxlan-channel{
    key vxlan-tunnel-id;
    leaf vxlan-tunnel-id {
        type uint32;
        description
            "Static VxLAN tunnel ID.";
    }

    leaf vxlan-tunnel-name {
        type string;
        description
            "Name of the static VxLAN tunnel.";
    }

    list address-family {
        key "af";
    }
}
```

```
    leaf af {
      type address-family-type;
      description
        "Address family type value.";
    }

    leaf tunnel-source-ip {
      type address-family-type;
      description
        "Source IP address for the static VxLAN tunnel";
    }

    leaf tunnel-destination-ip {
      type address-family-type;
      description
        "Destination IP address for the static VxLAN tunnel";
    }

    list bind-vxlan-id {
      key vxlan-id;
      leaf vxlan-id {
        type vxlan-id;
        description
          "VxLAN ID.";
      }
      description
        "VxLAN ID list for the VTEP.";
    }

    description
      "Per-af params.";
  }
  description
    "Configure the VxLAN channel";
}

description "ietf-vbng configuration!";
}
description "augment lne model";
}
}
<CODE ENDS>
```

5. Security Considerations

6. Acknowledgements

7. IANA Considerations

This document requires no IANA Actions. Please remove this section before RFC publication.

8. References

8.1. Normative References

[I-D.asechoud-rtgwg-qos-model]

Choudhary, A., Jethanandani, M., Strahle, N., Aries, E., and I. Chen, "YANG Model for QoS", draft-asechoud-rtgwg-qos-model-02 (work in progress), June 2017.

[I-D.ietf-netmod-acl-model]

Bogdanovic, D., Jethanandani, M., Huang, L., Agarwal, S., and D. Blair, "Network Access Control List (ACL) YANG Data Model", draft-ietf-netmod-acl-model-11 (work in progress), June 2017.

[I-D.ietf-rtgwg-lne-model]

Berger, L., Hopps, C., Lindem, A., and D. Bogdanovic, "YANG Logical Network Elements", draft-ietf-rtgwg-lne-model-02 (work in progress), March 2017.

[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>>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.

8.2. Informative References

[I-D.gu-nfvrg-cloud-bng-architecture]

Gu, R. and S. Hu, "Control and User Plane Separation Architecture of Cloud based BNG", draft-gu-nfvrg-cloud-bng-architecture-00 (work in progress), February 2017.

[I-D.huang-nvo3-vxlan-extension-for-vbras]

Huang, L. and S. Hu, "VxLAN Extension Requirement for Signaling Exchange Between Control and User Plane of vBras", draft-huang-nvo3-vxlan-extension-for-vbras-00 (work in progress), March 2017.

[I-D.wcg-i2rs-cu-separation-infor-model]

Wang, Z., iqjie@mail.ustc.edu.cn, i., and R. Gu, "Information Model of Control-Plane and User-Plane separation BNG", draft-wcg-i2rs-cu-separation-infor-model-00 (work in progress), March 2017.

Authors' Addresses

Fangwei Hu
ZTE Corporation
No.889 Bibo Rd
Shanghai 201203
China

Phone: +86 21 68896273
Email: hu.fangwei@zte.com.cn

RongRong Hua
ZTE Corporation
No.50 Software Avenue,Yuhuatai District
Nanjing, Jiangsu Province 210012
China

Email: hua.rongrong@zte.com.cn

Shujun Hu
China Mobile
32 Xuanwumen West Ave, Xicheng District
Beijing 100053
China

Email: 13488683482@139.com

Lu Huang
China Mobile
32 Xuanwumen West Ave, Xicheng District
Beijing 100053
China

Email: hlisname@yahoo.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 20, 2018

L. Berger
LabN Consulting, L.L.C.
C. Hopps
Deutsche Telekom
A. Lindem
Cisco Systems
D. Bogdanovic

X. Liu
Jabil
March 19, 2018

YANG Model for Logical Network Elements
draft-ietf-rtgwg-lne-model-10

Abstract

This document defines a logical network element YANG module. This module can be used to manage the logical resource partitioning that may be present on a network device. Examples of common industry terms for logical resource partitioning are Logical Systems or Logical Routers. The YANG model in this document conforms to the Network Management Datastore Architecture as defined in RFCXXXX.

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 September 20, 2018.

Copyright Notice

Copyright (c) 2018 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	2
1.1. Terminology	3
2. Overview	4
3. Logical Network Elements	5
3.1. LNE Instantiation and Resource Assignment	6
3.2. LNE Management - LNE View	7
3.3. LNE Management - Host Network Device View	7
4. Security Considerations	8
5. IANA Considerations	9
6. Logical Network Element Model	10
7. References	14
7.1. Normative References	14
7.2. Informative References	15
Appendix A. Acknowledgments	16
Appendix B. Examples	17
B.1. Example: Host Device Managed LNE	17
B.1.1. Configuration Data	20
B.1.2. State Data	24
B.2. Example: Self Managed LNE	33
B.2.1. Configuration Data	36
B.2.2. State Data	39
Authors' Addresses	48

1. Introduction

This document defines a YANG [RFC6020] module to support the creation of logical network elements on a network device. A logical network element (LNE) is an independently managed virtual device made up of resources allocated to it from the host or parent network device. An LNE running on a host network device conceptually parallels a virtual machine running on a host system. Using host-virtualization terminology one could refer to an LNE as a "Guest", and the containing network-device as the "Host". While LNEs may be implemented via host-virtualization technologies this is not a requirement. The YANG model in this document conforms to the Network

Management Datastore Architecture defined in the [I-D.ietf-netmod-revised-datastores].

This document also defines the necessary augmentations for allocating host resources to a given LNE. As the interface management model [I-D.ietf-netmod-rfc7223bis] is the only a module that currently defines host resources, this document currently defines only a single augmentation to cover the assignment of interfaces to an LNE. Future modules that define support for the control of host device resources are expected to, where appropriate, provide parallel support for the assignment of controlled resources to LNEs.

As each LNE is an independently managed device, each will have its own set of YANG modeled data that is independent of the host device and other LNEs. For example, multiple LNEs may all have their own "Tunnel0" interface defined which will not conflict with each other and will not exist in the host's interface model. An LNE will have its own management interfaces possibly including independent instances of netconf/restconf/etc servers to support configuration of their YANG models. As an example of this independence, an implementation may choose to completely rename assigned interfaces, so on the host the assigned interface might be called "Ethernet0/1" while within the LNE it might be called "eth1".

In addition to standard management interfaces, a host device implementation may support accessing LNE configuration and operational YANG models directly from the host system. When supported, such access is accomplished through a yang-schema-mount mount point [I-D.ietf-netmod-schema-mount] under which the root level LNE YANG models may be accessed.

Examples of vendor terminology for an LNE include logical system or logical router, and virtual switch, chassis, or fabric.

1.1. Terminology

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.

Readers are expected to be familiar with terms and concepts of YANG [RFC7950] and YANG Schema Mount [I-D.ietf-netmod-schema-mount].

This document uses the graphical representation of data models defined in [I-D.ietf-netmod-yang-tree-diagrams].

2. Overview

In this document, we consider network devices that support protocols and functions defined within the IETF Routing Area, e.g., routers, firewalls, and hosts. Such devices may be physical or virtual, e.g., a classic router with custom hardware or one residing within a server-based virtual machine implementing a virtual network function (VNF). Each device may sub-divide their resources into logical network elements (LNEs), each of which provides a managed logical device. Examples of vendor terminology for an LNE include logical system or logical router, and virtual switch, chassis, or fabric. Each LNE may also support virtual routing and forwarding (VRF) and virtual switching instance (VSI) functions, which are referred to below as a network instances (NIs). This breakdown is represented in Figure 1.

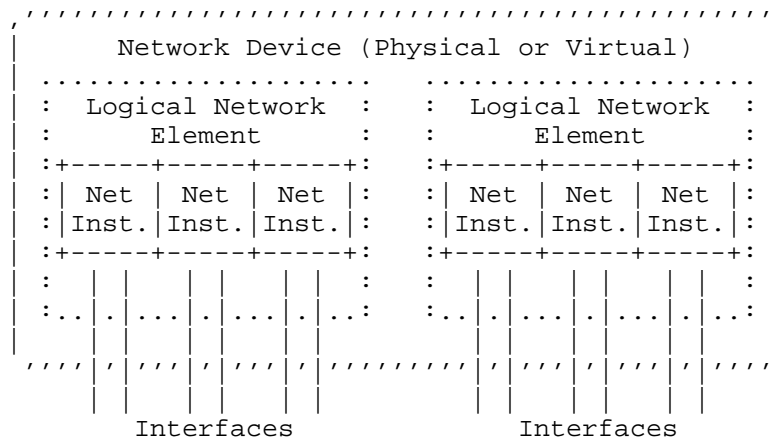


Figure 1: Module Element Relationships

A model for LNEs is described in Section 3 and the model for NIs is covered in [I-D.ietf-rtgwg-ni-model].

The interface management model [I-D.ietf-netmod-rfc7223bis] is an existing model that is impacted by the definition of LNEs and network instances. This document and [I-D.ietf-rtgwg-ni-model] define augmentations to the interface module to support LNEs and NIs. Similar elements, although perhaps only for LNEs, may also need to be included as part of the definition of the future hardware and QoS modules.

Interfaces are a crucial part of any network device's configuration and operational state. They generally include a combination of raw

physical interfaces, link-layer interfaces, addressing configuration, and logical interfaces that may not be tied to any physical interface. Several system services, and layer 2 and layer 3 protocols may also associate configuration or operational state data with different types of interfaces (these relationships are not shown for simplicity). The interface management model is defined by [I-D.ietf-netmod-rfc7223bis]. The logical-network-element module augments existing interface management model by adding an identifier which is used on interfaces to identify an associated LNE.

The interface related augmentation is as follows:

```
module: ietf-logical-network-element
  augment /if:interfaces/if:interface:
    +---rw bind-lne-name?    ->
      /logical-network-elements/logical-network-element/name
```

The interface model defined in [I-D.ietf-netmod-rfc7223bis] is structured to include all interfaces in a flat list, without regard to logical assignment of resources supported on the device. The bind-lne-name leaf provides the association between an interface and its associated LNE. Note that as currently defined, to assign an interface to both an LNE and NI, the interface would first be assigned to the LNE and then within that LNE's interface module, the LNE's representation of that interface would be assigned to an NI using the mechanisms defined in [I-D.ietf-rtgwg-ni-model].

3. Logical Network Elements

Logical network elements support the ability of some devices to partition resources into independent logical routers and/or switches. Device support for multiple logical network elements is implementation specific. Systems without such capabilities need not include support for the logical-network-element module. In physical devices, some hardware features are shared across partitions, but control plane (e.g., routing) protocol instances, tables, and configuration are managed separately. For example, in logical routers or VNFs, this may correspond to establishing multiple logical instances using a single software installation. The model supports configuration of multiple instances on a single device by creating a list of logical network elements, each with their own configuration and operational state related to routing and switching protocols.

The LNE model can be represented as:

```

module: ietf-logical-network-element
  +--rw logical-network-elements
    +--rw logical-network-element* [name]
      +--rw name                string
      +--rw managed?            boolean
      +--rw description?        string
      +--mp root
  augment /if:interfaces/if:interface:
    +--rw bind-lne-name?
      -> /logical-network-elements/logical-network-element/name

  notifications:
    +---n bind-lne-name-failed
      +--ro name                -> /if:interfaces/interface/name
      +--ro bind-lne-name
      |   -> /if:interfaces/interface/lne:bind-lne-name
      +--ro error-info?         string

```

'name' identifies the logical network element. 'managed' indicates if the server providing the host network device will provide the client LNE information via the 'root' structure. The root of an LNE's specific data is the schema mount point 'root'. bind-lne-name is used to associated an interface with an LNE and bind-lne-name-failed is used in certain failure cases.

An LNE root MUST contain at least the YANG library [RFC7895] and Interfaces [I-D.ietf-netmod-rfc7223bis] modules.

3.1. LNE Instantiation and Resource Assignment

Logical network elements may be controlled by clients using existing list operations. When list entries are created, a new LNE is instantiated. The models mounted under an LNE root are expected to be dependent on the server implementation. When a list entry is deleted, an existing LNE is destroyed. For more information, see [RFC7950] Section 7.8.6.

Once instantiated, host network device resources can be associated with the new LNE. As previously mentioned, this document augments ietf-interfaces with the bind-lne-name leaf to support such associations for interfaces. When an bind-lne-name is set to a valid LNE name, an implementation MUST take whatever steps are internally necessary to assign the interface to the LNE or provide an error message (defined below) with an indication of why the assignment failed. It is possible for the assignment to fail while processing the set, or after asynchronous processing. Error notification in the latter case is supported via a notification.

On a successful interface assignment to an LNE, an implementation MUST also make the resource available to the LNE by providing a system created interface to the LNE. The name of the system created interface is a local matter and may be identical or completely different, and mapped from and to, the name used in the context of the host device. The system created interface SHOULD be exposed via the LNE-specific instance of the interfaces module [I-D.ietf-netmod-rfc7223bis].

3.2. LNE Management - LNE View

Each LNE instance is expected to support management functions from within the context of the LNE root, via a server that provides information with the LNE's root exposed as device root. Management functions operating within the context of an LNE are accessed through the LNE's standard management interfaces, e.g., NETCONF and SNMP. Initial configuration, much like the initial configuration of the host device, is a local implementation matter.

When accessing an LNE via the LNE's management interface, a network-device representation will be presented, but its scope will be limited to the specific LNE. Normal YANG/NETCONF mechanisms, together with the required YANG library [RFC7895] instance, can be used to identify the available modules. Each supported module will be presented as a top level module. Only LNE associated resources will be reflected in resource related modules, e.g., interfaces, hardware, and perhaps QoS. From the management perspective, there will be no difference between the available LNE view (information) and a physical network device.

3.3. LNE Management - Host Network Device View

There are multiple implementation approaches possible to enable a network device to support the logical-network-element module and multiple LNEs. Some approaches will allow the management functions operating at network device level to access LNE configuration and operational information, while others will not. Similarly, even when LNE management from the network device is supported by the implementation, it may be prohibited by user policy.

Independent of the method selected by an implementation, the 'managed' boolean mentioned above is used to indicate when LNE management from the network device context is possible. When the 'managed' boolean is 'false', the LNE cannot be managed by the host system and can only be managed from within the context of the LNE as described in the previous section, Section 3.2. Attempts to access information below a root node whose associated 'managed' boolean is set to 'false' MUST result in the error message indicated below. In

some implementations, it may not be possible to change this value. For example, when an LNE is implemented using virtual machine and traditional hypervisor technologies, it is likely that this value will be restricted to a 'false' value.

It is an implementation choice if the information can be accessed and modified from within the context of the LNE, or even the context of the host device. When the 'managed' boolean is 'true', LNE information SHALL be accessible from the context of the host device. When the associated schema-mount definition has the 'config' leaf set to 'true', then LNE information SHALL also be modifiable from the context of the host device. When LNE information is available from both the host device and from within the context of the LNE, the same information MUST be made available via the 'root' element, with paths modified as described in [I-D.ietf-netmod-schema-mount].

An implementation MAY represent an LNE's schema using either the 'inline' or 'shared-schema' approaches defined in [I-D.ietf-netmod-schema-mount]. The choice of which to use is completely an implementation choice. The inline case is anticipated to be generally used in the cases where the 'managed' will always be 'false'. The 'shared-schema' approach is expected to be most useful in the case where all LNEs share the same schema. When 'shared-schema' is used with an LNE mount point, the YANG library rooted in the LNE's mount point MUST match the associated schema defined according to the ietf-yang-schema-mount module.

Beyond the two modules that will always be present for an LNE, as an LNE is a network device itself, all modules that may be present at the top level network device MAY also be present for the LNE. The list of available modules is expected to be implementation dependent. As is the method used by an implementation to support LNEs. Appendix B provide example uses of LNEs.

4. Security Considerations

The YANG modules specified in this document define a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

LNE information represents device and network configuration information. As such, the security of this information is important, but it is fundamentally no different than any other interface or device configuration information that has already been covered in other documents such as [I-D.ietf-netmod-rfc7223bis], [RFC7317] and [I-D.ietf-netmod-rfc8022bis].

The vulnerable "config true" parameters and subtrees are the following:

/logical-network-elements/logical-network-element: This list specifies the logical network element and the related logical device configuration.

/logical-network-elements/logical-network-element/managed: While this leaf is contained in the previous list, it is worth particular attention as it controls whether information under the LNE mount point is accessible by both the host device and within the LNE context. There may be extra sensitivity to this leaf in environments where an LNE is managed by a different party than the host device, and that party does not wish to share LNE information with the operator of the host device.

/if:interfaces/if:interface/bind-lne-name: This leaf indicates the LNE instance to which an interface is assigned. Implementations should pay particular attention to when changes to this leaf are permitted as removal of an interface from an LNE can have major impact on the LNEs operation as it is similar to physically removing an interface from the device. Implementations can reject an reassignment using the previously described error message generation.

Unauthorized access to any of these lists can adversely affect the security of both the local device and the network. This may lead to network malfunctions, delivery of packets to inappropriate destinations, and other problems.

5. IANA Considerations

This document registers a URI in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration is requested to be made.

URI: urn:ietf:params:xml:ns:yang:ietf-logical-network-element

Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.

This document registers a YANG module in the YANG Module Names registry [RFC6020].

```
name:      ietf-logical-network-element
namespace: urn:ietf:params:xml:ns:yang:ietf-logical-network-element
prefix:    lne
reference:  RFC XXXX
```

6. Logical Network Element Model

The structure of the model defined in this document is described by the YANG module below.

```
<CODE BEGINS> file "ietf-logical-network-element@2018-03-20.yang"
module ietf-logical-network-element {
  yang-version 1.1;

  // namespace

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-logical-network-element";
  prefix lne;

  // import some basic types

  import ietf-interfaces {
    prefix if;
    reference "draft-ietf-netmod-rfc7223bis:
              A YANG Data Model for Interface Management";
  }
  import ietf-yang-schema-mount {
    prefix yangmnt;
    reference "draft-ietf-netmod-schema-mount: YANG Schema Mount";
    // RFC Ed.: Please replace this draft name with the corresponding
    // RFC number
  }

  organization
    "IETF Routing Area (rtgwg) Working Group";
  contact
    "WG Web:  <http://tools.ietf.org/wg/rtgwg/>
    WG List:  <mailto:rtgwg@ietf.org>

    Author:   Lou Berger
              <mailto:lberger@labn.net>
    Author:   Christan Hopps
              <mailto:chopps@chopps.org>
    Author:   Acee Lindem
```

```

    <mailto:acee@cisco.com>
  Author:   Dean Bogdanovic
            <mailto:ivandean@gmail.com>";
description
  "This module is used to support multiple logical network
  elements on a single physical or virtual system.

  Copyright (c) 2017 IETF Trust and the persons
  identified as authors of the code.  All rights reserved.

  Redistribution and use in source and binary forms, with or
  without modification, is permitted pursuant to, and subject
  to the license terms contained in, the Simplified BSD
  License set forth in Section 4.c of the IETF Trust's Legal
  Provisions Relating to IETF Documents
  (http://trustee.ietf.org/license-info).

  This version of this YANG module is part of RFC XXXX; see
  the RFC itself for full legal notices.";

// RFC Ed.: replace XXXX with actual RFC number and remove
// this note
// RFC Ed.: please update TBD

revision 2018-03-20 {
  description
    "Initial revision.";
  reference "RFC XXXX";
}

// top level device definition statements

container logical-network-elements {
  description
    "Allows a network device to support multiple logical
    network element (device) instances.";
  list logical-network-element {
    key "name";
    description
      "List of logical network elements.";
    leaf name {
      type string;
      description
        "Device-wide unique identifier for the
        logical network element.";
    }
    leaf managed {
      type boolean;
    }
  }
}
```

```
    default "true";
    description
        "True if the host can access LNE information
         using the root mount point. This value
         my not be modifiable in all implementations.";
}
leaf description {
    type string;
    description
        "Description of the logical network element.";
}
container "root" {
    description
        "Container for mount point.";
    yangmnt:mount-point "root" {
        description
            "Root for models supported per logical
             network element. This mount point may or may not
             be inline based on the server implementation. It
             SHALL always contain a YANG library and interfaces
             instance.

            When the associated 'managed' leaf is 'false' any
            operation that attempts to access information below
            the root SHALL fail with an error-tag of
            'access-denied' and an error-app-tag of
            'lne-not-managed'.";
    }
}
}
}

// augment statements

augment "/if:interfaces/if:interface" {
    description
        "Add a node for the identification of the logical network
         element associated with an interface. Applies to
         interfaces that can be assigned on a per logical network
         element basis.

        Note that a standard error will be returned if the
        identified leafref isn't present. If an interfaces
        cannot be assigned for any other reason, the operation
        SHALL fail with an error-tag of 'operation-failed' and an
        error-app-tag of 'lne-assignment-failed'. A meaningful
        error-info that indicates the source of the assignment
        failure SHOULD also be provided.";
```

```
    leaf bind-lne-name {
      type leafref {
        path
          "/logical-network-elements/logical-network-element/name";
      }
      description
        "Logical network element ID to which interface is bound.";
    }
  }

  // notification statements

  notification bind-lne-name-failed {
    description
      "Indicates an error in the association of an interface to an
       LNE. Only generated after success is initially returned when
       bind-lne-name is set.";
    leaf name {
      type leafref {
        path "/if:interfaces/if:interface/if:name";
      }
      mandatory true;
      description
        "Contains the interface name associated with the
         failure.";
    }
    leaf bind-lne-name {
      type leafref {
        path "/if:interfaces/if:interface/lne:bind-lne-name";
      }
      mandatory true;
      description
        "Contains the bind-lne-name associated with the
         failure.";
    }
    leaf error-info {
      type string;
      description
        "Optionally, indicates the source of the assignment
         failure.";
    }
  }
}
<CODE ENDS>
```

7. References

7.1. Normative References

- [I-D.ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K.,
and R. Wilton, "Network Management Datastore
Architecture", draft-ietf-netmod-revised-datastores-10
(work in progress), January 2018.
- [I-D.ietf-netmod-rfc7223bis]
Bjorklund, M., "A YANG Data Model for Interface
Management", draft-ietf-netmod-rfc7223bis-03 (work in
progress), January 2018.
- [I-D.ietf-netmod-schema-mount]
Bjorklund, M. and L. Lhotka, "YANG Schema Mount", draft-
ietf-netmod-schema-mount-08 (work in progress), October
2017.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,
<<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688,
DOI 10.17487/RFC3688, January 2004,
<<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
(TLS) Protocol Version 1.2", RFC 5246,
DOI 10.17487/RFC5246, August 2008,
<<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for
the Network Configuration Protocol (NETCONF)", RFC 6020,
DOI 10.17487/RFC6020, October 2010,
<<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed.,
and A. Bierman, Ed., "Network Configuration Protocol
(NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011,
<<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure
Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011,
<<https://www.rfc-editor.org/info/rfc6242>>.

- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<https://www.rfc-editor.org/info/rfc6536>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

7.2. Informative References

- [I-D.ietf-netmod-entity]
Bierman, A., Bjorklund, M., Dong, J., and D. Romascanu, "A YANG Data Model for Hardware Management", draft-ietf-netmod-entity-08 (work in progress), January 2018.
- [I-D.ietf-netmod-rfc8022bis]
Lhotka, L., Lindem, A., and Y. Qu, "A YANG Data Model for Routing Management (NMDA Version)", draft-ietf-netmod-rfc8022bis-11 (work in progress), January 2018.
- [I-D.ietf-netmod-yang-tree-diagrams]
Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-06 (work in progress), February 2018.
- [I-D.ietf-rtgwg-device-model]
Lindem, A., Berger, L., Bogdanovic, D., and C. Hopps, "Network Device YANG Logical Organization", draft-ietf-rtgwg-device-model-02 (work in progress), March 2017.
- [I-D.ietf-rtgwg-ni-model]
Berger, L., Hopps, C., Lindem, A., Bogdanovic, D., and X. Liu, "YANG Model for Network Instances", draft-ietf-rtgwg-ni-model-11 (work in progress), March 2018.
- [RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<https://www.rfc-editor.org/info/rfc7317>>.
- [RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <<https://www.rfc-editor.org/info/rfc7895>>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.

Appendix A. Acknowledgments

The Routing Area Yang Architecture design team members included Acee Lindem, Anees Shaikh, Christian Hopps, Dean Bogdanovic, Lou Berger, Qin Wu, Rob Shakir, Stephane Litkowski, and Yan Gang. Useful review comments were also received by Martin Bjorklund, John Scudder, Dan Romascanu and Taylor Yu.

This document was motivated by, and derived from, [I-D.ietf-rtgwg-device-model].

The RFC text was produced using Marshall Rose's xml2rfc tool.

Thanks to Alvaro Retana for IESG review.

Appendix B. Examples

The following subsections provide example uses of LNEs.

B.1. Example: Host Device Managed LNE

This section describes an example of the LNE model using schema mount to achieve the parent management. An example device supports multiple instances of LNEs (logical routers), each of which supports features of layer 2 and layer 3 interfaces [I-D.ietf-netmod-rfc7223bis], routing information base [I-D.ietf-netmod-rfc8022bis], and OSPF protocol. Each of these features is specified by a YANG model, and they are combined using YANG Schema Mount as shown below. Not all possible mounted modules are shown. For example implementations could also mount the model defined in [I-D.ietf-netmod-entity].

```

module: ietf-logical-network-element
  +--rw logical-network-elements
    +--rw logical-network-element* [name]
      +--rw name string
      +--mp root
        +--ro yanglib:modules-state/
          |   +--ro module-set-id string
          |   +--ro module* [name revision]
          |     +--ro name yang:yang-identifier
        +--rw sys:system/
          |   +--rw contact? string
          |   +--rw hostname? inet:domain-name
          |   +--rw authentication {authentication}?
          |     +--rw user-authentication-order* identityref
          |     +--rw user* [name] {local-users}?
          |       +--rw name string
          |       +--rw password? ianach:crypt-hash
          |       +--rw authorized-key* [name]
          |         +--rw name string
          |         +--rw algorithm string
          |         +--rw key-data binary
        +--ro sys:system-state/
          |   ...
        +--rw rt:routing/
          |   +--rw router-id? yang:dotted-quad
          |   +--rw control-plane-protocols
          |     +--rw control-plane-protocol* [type name]
          |       +--rw ospf:ospf/
          |         +--rw areas

```

```

|           |--rw area* [area-id]
|           |--rw interfaces
|             |--rw interface* [name]
|               |--rw name if:interface-ref
|               |--rw cost?   uint16
|--rw if:interfaces/
  |--rw interface* [name]
    |--rw name          string
    |--rw ip:ipv4!/
    |   |--rw address* [ip]
    |   ...
module: ietf-interfaces
  |--rw interfaces
    |--rw interface* [name]
      |--rw name          string
      |--rw lne:bind-lne-name?  string
      |--ro oper-status    enumeration
module: ietf-yang-library
  |--ro modules-state
    |--ro module-set-id    string
    |--ro module* [name revision]
      |--ro name          yang:yang-identifier
module: ietf-system
  |--rw system
    |--rw contact?        string
    |--rw hostname?       inet:domain-name
    |--rw authentication {authentication}?
      |--rw user-authentication-order*  identityref
      |--rw user* [name] {local-users}?
        |--rw name          string
        |--rw password?     ianach:crypt-hash
        |--rw authorized-key* [name]
          |--rw name          string
          |--rw algorithm    string
          |--rw key-data     binary
  |--ro system-state
    |--ro platform
      |--ro os-name?        string
      |--ro os-release?     string

```

To realize the above schema, the example device implements the following schema mount instance:

```
"ietf-yang-schema-mount:schema-mounts": {  
  "mount-point": [  
    {  
      "module": "ietf-logical-network-element",  
      "label": "root",  
      "shared-schema": {}  
    }  
  ]  
}
```

By using the implementation of the YANG schema mount, an operator can create instances of logical routers. An interface can be assigned to a logical router, so that the logical router has the permission to access this interface. The OSPF protocol can then be enabled on this assigned interface.

For this implementation, a parent management session has access to the schemas of both the parent hosting system and the child logical routers. In addition, each child logical router can grant its own management sessions, which have the following schema:

```

module: ietf-yang-library
  +--ro modules-state
    +--ro module-set-id      string
    +--ro module* [name revision]
      +--ro name                yang:yang-identifier

module: ietf-system
  +--rw system
    | +--rw contact?          string
    | +--rw hostname?         inet:domain-name
    | +--rw authentication {authentication}?
    | | +--rw user-authentication-order* identityref
    | | +--rw user* [name] {local-users}?
    | | | +--rw name          string
    | | | +--rw password?     ianach:crypt-hash
    | | | +--rw authorized-key* [name]
    | | | | +--rw name        string
    | | | | +--rw algorithm   string
    | | | | +--rw key-data    binary
    | +--ro system-state
    | +--ro platform
    | | +--ro os-name?        string
    | | +--ro os-release?     string

module: ietf-routing
  rw-- routing
    +--rw router-id?          yang:dotted-quad
    +--rw control-plane-protocols
    | +--rw control-plane-protocol* [type name]
    | | +--rw ospf:ospf/
    | | | +--rw areas
    | | | | +--rw area* [area-id]
    | | | | +--rw interfaces
    | | | | | +--rw interface* [name]
    | | | | | | +--rw name      if:interface-ref
    | | | | | | +--rw cost?    uint16

module: ietf-interfaces
  +--rw interfaces
    +--rw interface* [name]
    | +--rw name                string
    +--ro oper-status           enumeration

```

B.1.1.1. Configuration Data

The following shows an example where two customer specific LNEs are configured:

```

{
  "ietf-logical-network-element:logical-network-elements": {
    "logical-network-element": [
      {
        "name": "cust1",
        "root": {
          "ietf-system:system": {
            "authentication": {
              "user": [
                {
                  "name": "john",
                  "password": "$0$password"
                }
              ]
            }
          },
          "ietf-routing:routing": {
            "router-id": "192.0.2.1",
            "control-plane-protocols": {
              "control-plane-protocol": [
                {
                  "type": "ietf-routing:ospf",
                  "name": "1",
                  "ietf-ospf:ospf": {
                    "af": "ipv4",
                    "areas": {
                      "area": [
                        {
                          "area-id": "203.0.113.1",
                          "interfaces": {
                            "interface": [
                              {
                                "name": "eth1",
                                "cost": 10
                              }
                            ]
                          }
                        }
                      ]
                    }
                  }
                }
              ]
            }
          },
          "ietf-interfaces:interfaces": {
            "interfaces": {
              "interface": [

```

```

    {
      "name": "eth1",
      "ip:ipv4": {
        "address": [
          {
            "ip": "192.0.2.11",
            "prefix-length": 24,
          }
        ]
      },
      "ip:ipv6": {
        "address": [
          {
            "ip": "2001:db8:0:2::11",
            "prefix-length": 64,
          }
        ]
      }
    }
  ]
}
},
{
  "name": "cust2",
  "root": {
    "ietf-system:system": {
      "authentication": {
        "user": [
          {
            "name": "john",
            "password": "$0$password"
          }
        ]
      }
    }
  }
  "ietf-routing:routing": {
    "router-id": "192.0.2.2",
    "control-plane-protocols": {
      "control-plane-protocol": [
        {
          "type": "ietf-routing:ospf",
          "name": "1",
          "ietf-ospf:ospf": {
            "af": "ipv4",
            "areas": {
              "area": [

```

[illegible]

```

    }
  ]
},
"ip:ipv6": {
  "address": [
    {
      "ip": "2001:db8:0:2::10",
      "prefix-length": 64,
    }
  ]
}
},
{
  "name": "cust1:eth1",
  "lne:bind-lne-name": "cust1"
},
{
  "name": "cust2:eth1",
  "lne:bind-lne-name": "cust2"
}
]
}
},
"ietf-system:system": {
  "authentication": {
    "user": [
      {
        "name": "root",
        "password": "$0$password"
      }
    ]
  }
}
}
}

```

B.1.2. State Data

The following shows possible state data associated the above configuration data:

```

{
  "ietf-logical-network-element:logical-network-elements": {
    "logical-network-element": [
      {
        "name": "cust1",
        "root": {
          "ietf-yang-library:modules-state": {

```



```
"module-set-id": "123e4567-e89b-12d3-a456-426655440000",
"module": [
  {
    "name": "iana-if-type",
    "revision": "2014-05-08",
    "namespace":
      "urn:ietf:params:xml:ns:yang:iana-if-type",
    "conformance-type": "import"
  },
  {
    "name": "ietf-inet-types",
    "revision": "2013-07-15",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-inet-types",
    "conformance-type": "import"
  },
  {
    "name": "ietf-interfaces",
    "revision": "2014-05-08",
    "feature": [
      "arbitrary-names",
      "pre-provisioning"
    ],
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-interfaces",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-ip",
    "revision": "2014-06-16",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-ip",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-ospf",
    "revision": "2018-03-03",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-ospf",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-routing",
    "revision": "2018-03-13",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-routing",
    "conformance-type": "implement"
  },
]
```

```
{
  "name": "ietf-system",
  "revision": "2014-08-06",
  "namespace":
    "urn:ietf:params:xml:ns:yang:ietf-system",
  "conformance-type": "implement"
},
{
  "name": "ietf-yang-library",
  "revision": "2016-06-21",
  "namespace":
    "urn:ietf:params:xml:ns:yang:ietf-yang-library",
  "conformance-type": "implement"
},
{
  "name": "ietf-yang-types",
  "revision": "2013-07-15",
  "namespace":
    "urn:ietf:params:xml:ns:yang:ietf-yang-types",
  "conformance-type": "import"
}
]
},
"ietf-system:system-state": {
  "ietf-system:system-state": {
    "platform": {
      "os-name": "NetworkOS"
    }
  }
},
"ietf-routing:routing": {
  "router-id": "192.0.2.1",
  "control-plane-protocols": {
    "control-plane-protocol": [
      {
        "type": "ietf-routing:ospf",
        "name": "1",
        "ietf-ospf:ospf": {
          "af": "ipv4",
          "areas": {
            "area": [
              {
                "area-id": "203.0.113.1",
                "interfaces": {
                  "interface": [
                    {
                      "name": "eth1",
                      "cost": 10
                    }
                  ]
                }
              }
            ]
          }
        }
      }
    ]
  }
}
```

```

    }
  ]
}
},
"ietf-interfaces:interfaces": {
  "interfaces": {
    "interface": [
      {
        "name": "eth1",
        "type": "iana-if-type:ethernetCsmacd",
        "oper-status": "up",
        "phys-address": "00:01:02:A1:B1:C1",
        "statistics": {
          "discontinuity-time": "2017-06-26T12:34:56-05:00"
        },
        "ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.11",
              "prefix-length": 24,
            }
          ]
        }
      }
    ]
  }
}
},
{
  "name": "cust2",
  "root": {
    "ietf-yang-library:modules-state": {
      "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
      "module": [
        {
          "name": "iana-if-type",
          "revision": "2014-05-08",
          "namespace":
            "urn:ietf:params:xml:ns:yang:iana-if-type",
          "conformance-type": "import"
        }
      ]
    }
  }
}

```

```
    },
    {
      "name": "ietf-inet-types",
      "revision": "2013-07-15",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-inet-types",
      "conformance-type": "import"
    },
    {
      "name": "ietf-interfaces",
      "revision": "2014-05-08",
      "feature": [
        "arbitrary-names",
        "pre-provisioning"
      ],
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-interfaces",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-ip",
      "revision": "2014-06-16",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-ip",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-ospf",
      "revision": "2018-03-03",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-ospf",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-routing",
      "revision": "2018-03-13",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-routing",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-system",
      "revision": "2014-08-06",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-system",
      "conformance-type": "implement"
    },
  ],
  {
```

```

        "name": "ietf-yang-library",
        "revision": "2016-06-21",
        "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-yang-library",
        "conformance-type": "implement"
    },
    {
        "name": "ietf-yang-types",
        "revision": "2013-07-15",
        "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-yang-types",
        "conformance-type": "import"
    }
]
}
"ietf-system:system-state": {
    "ietf-system:system-state": {
        "platform": {
            "os-name": "NetworkOS"
        }
    }
},
"ietf-routing:routing": {
    "router-id": "192.0.2.2",
    "control-plane-protocols": {
        "control-plane-protocol": [
            {
                "type": "ietf-routing:ospf",
                "name": "1",
                "ietf-ospf:ospf": {
                    "af": "ipv4",
                    "areas": {
                        "area": [
                            {
                                "area-id": "203.0.113.1",
                                "interfaces": {
                                    "interface": [
                                        {
                                            "name": "eth1",
                                            "cost": 10
                                        }
                                    ]
                                }
                            }
                        ]
                    }
                }
            }
        ]
    }
}

```

```

    ]
  }
}
"ietf-interfaces:interfaces": {
  "interfaces": {
    {
      "name": "eth1",
      "type": "iana-if-type:ethernetCsmacd",
      "oper-status": "up",
      "phys-address": "00:01:02:A1:B1:C2",
      "statistics": {
        "discontinuity-time": "2017-06-26T12:34:56-05:00"
      },
      "ip:ipv4": {
        "address": [
          {
            "ip": "192.0.2.11",
            "prefix-length": 24,
          }
        ]
      }
    }
  ]
}
}
]
},
"ietf-interfaces:interfaces": {
  "interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "oper-status": "up",
        "phys-address": "00:01:02:A1:B1:C0",
        "statistics": {
          "discontinuity-time": "2017-06-26T12:34:56-05:00"
        },
        "ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.10",
              "prefix-length": 24,
            }
          ]
        }
      }
    ]
  }
}

```

```
    },
    {
      "name": "cust1:eth1",
      "type": "iana-if-type:ethernetCsmacd",
      "oper-status": "up",
      "phys-address": "00:01:02:A1:B1:C1",
      "statistics": {
        "discontinuity-time": "2017-06-26T12:34:56-05:00"
      }
    },
    {
      "name": "cust2:eth1",
      "type": "iana-if-type:ethernetCsmacd",
      "oper-status": "up",
      "phys-address": "00:01:02:A1:B1:C2",
      "statistics": {
        "discontinuity-time": "2017-06-26T12:34:56-05:00"
      }
    }
  ]
}
},
"ietf-system:system-state": {
  "platform": {
    "os-name": "NetworkOS"
  }
},
"ietf-yang-library:modules-state": {
  "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
  "module": [
    {
      "name": "iana-if-type",
      "revision": "2014-05-08",
      "namespace":
        "urn:ietf:params:xml:ns:yang:iana-if-type",
      "conformance-type": "import"
    },
    {
      "name": "ietf-inet-types",
      "revision": "2013-07-15",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-inet-types",
      "conformance-type": "import"
    },
    {
      "name": "ietf-interfaces",
```

```
"revision": "2014-05-08",
"feature": [
  "arbitrary-names",
  "pre-provisioning"
],
"namespace":
"urn:ietf:params:xml:ns:yang:ietf-interfaces",
"conformance-type": "implement"
},
{
  "name": "ietf-ip",
  "revision": "2014-06-16",
  "namespace":
"urn:ietf:params:xml:ns:yang:ietf-ip",
  "conformance-type": "implement"
},
{
  "name": "ietf-logical-network-element",
  "revision": "2017-03-13",
  "feature": [
    "bind-lne-name"
  ],
  "namespace":
"urn:ietf:params:xml:ns:yang:ietf-logical-network-element",
  "conformance-type": "implement"
},
{
  "name": "ietf-ospf",
  "revision": "2018-03-03",
  "namespace":
"urn:ietf:params:xml:ns:yang:ietf-ospf",
  "conformance-type": "implement"
},
{
  "name": "ietf-routing",
  "revision": "2018-03-13",
  "namespace":
"urn:ietf:params:xml:ns:yang:ietf-routing",
  "conformance-type": "implement"
},
{
  "name": "ietf-system",
  "revision": "2014-08-06",
  "namespace":
"urn:ietf:params:xml:ns:yang:ietf-system",
  "conformance-type": "implement"
},
{
```



```

        "name": "ietf-yang-library",
        "revision": "2016-06-21",
        "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-yang-library",
        "conformance-type": "implement"
    },
    {
        "name": "ietf-yang-schema-mount",
        "revision": "2017-05-16",
        "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount",
        "conformance-type": "implement"
    },
    {
        "name": "ietf-yang-types",
        "revision": "2013-07-15",
        "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-yang-types",
        "conformance-type": "import"
    }
]
},
"ietf-yang-schema-mount:schema-mounts": {
    "mount-point": [
        {
            "module": "ietf-logical-network-element",
            "label": "root",
            "shared-schema": {}
        }
    ]
}
}

```

B.2. Example: Self Managed LNE

This section describes an example of the LNE model using schema mount to achieve child independent management. An example device supports multiple instances of LNEs (logical routers), each of them has the features of layer 2 and layer 3 interfaces [I-D.ietf-netmod-rfc7223bis], routing information base [I-D.ietf-netmod-rfc8022bis], and the OSPF protocol. Each of these features is specified by a YANG model, and they are put together by the YANG Schema Mount as following:

```

    module: ietf-logical-network-element
+--rw logical-network-elements
  +--rw logical-network-element* [name]
    +--rw name                string
    +--mp root
      // The internal modules of the LNE are not visible to
      // the parent management.
      // The child manages its modules, including ietf-routing
      // and ietf-interfaces

module: ietf-interfaces
+--rw interfaces
  +--rw interface* [name]
    +--rw name                string
    +--rw lne:bind-lne-name?  string
    +--ro oper-status         enumeration

module: ietf-yang-library
+--ro modules-state
  +--ro module-set-id        string
  +--ro module* [name revision]
    +--ro name                yang:yang-identifier

module: ietf-system
+--rw system
|   +--rw contact?           string
|   +--rw hostname?          inet:domain-name
|   +--rw authentication {authentication}?
|     +--rw user-authentication-order* identityref
|     +--rw user* [name] {local-users}?
|       +--rw name            string
|       +--rw password?       ianach:crypt-hash
|       +--rw authorized-key* [name]
|         +--rw name            string
|         +--rw algorithm      string
|         +--rw key-data       binary
+--ro system-state
  +--ro platform
    |   +--ro os-name?         string
    |   +--ro os-release?     string

```

To realize the above schema, the device implements the following schema mount instance:

```
"ietf-yang-schema-mount:schema-mounts": {  
  "mount-point": [  
    {  
      "module": "ietf-logical-network-element",  
      "label": "root",  
      "inline": {}  
    }  
  ]  
}
```

By using the implementation of the YANG schema mount, an operator can create instances of logical routers, each with their logical router specific in-line modules. An interface can be assigned to a logical router, so that the logical router has the permission to access this interface. The OSPF protocol can then be enabled on this assigned interface. Each logical router independently manages its own set of modules, which may or may not be the same as other logical routers. The following is an example of schema set implemented on one particular logical router:

```

module: ietf-yang-library
  +--ro modules-state
    +--ro module-set-id      string
    +--ro module* [name revision]
      +--ro name                yang:yang-identifier

module: ietf-system
  +--rw system
    | +--rw contact?          string
    | +--rw hostname?        inet:domain-name
    | +--rw authentication {authentication}?
    |   +--rw user-authentication-order* identityref
    |   +--rw user* [name] {local-users}?
    |     +--rw name          string
    |     +--rw password?     ianach:crypt-hash
    |     +--rw authorized-key* [name]
    |       +--rw name        string
    |       +--rw algorithm   string
    |       +--rw key-data    binary
    +--ro system-state
      +--ro platform
        | +--ro os-name?      string
        | +--ro os-release?   string

module: ietf-routing
  +--rw routing
    +--rw router-id?          yang:dotted-quad
    +--rw control-plane-protocols
      +--rw control-plane-protocol* [type name]
        +--rw ospf:ospf/
          +--rw areas
            +--rw area* [area-id]
              +--rw interfaces
                +--rw interface* [name]
                  +--rw name      if:interface-ref
                  +--rw cost?     uint16

module: ietf-interfaces
  +--rw interfaces
    +--rw interface* [name]
      +--rw name                string
      +--ro oper-status          enumeration

```

B.2.1. Configuration Data

Each of the child virtual routers is managed through its own sessions and configuration data.

B.2.1.1.1. Logical Network Element 'vnf1'

The following shows an example configuration data for the LNE name "vnf1":

```
{
  "ietf-system:system": {
    "authentication": {
      "user": [
        {
          "name": "john",
          "password": "$0$password"
        }
      ]
    }
  },
  "ietf-routing:routing": {
    "router-id": "192.0.2.1",
    "control-plane-protocols": {
      "control-plane-protocol": [
        {
          "type": "ietf-routing:ospf",
          "name": "1",
          "ietf-ospf:ospf": {
            "af": "ipv4",
            "areas": {
              "area": [
                {
                  "area-id": "203.0.113.1",
                  "interfaces": {
                    "interface": [
                      {
                        "name": "eth1",
                        "cost": 10
                      }
                    ]
                  }
                }
              ]
            }
          }
        }
      ]
    }
  },
  "ietf-interfaces:interfaces": {
    "interfaces": {
      "interface": [
```

```

    {
      "name": "eth1",
      "ip:ipv4": {
        "address": [
          {
            "ip": "192.0.2.11",
            "prefix-length": 24,
          }
        ]
      }
    }
  ]
}

```

B.2.1.2. Logical Network Element 'vnf2'

The following shows an example configuration data for the LNE name "vnf2":

```

{
  "ietf-system:system": {
    "authentication": {
      "user": [
        {
          "name": "john",
          "password": "$0$password"
        }
      ]
    }
  },
  "ietf-routing:routing": {
    "router-id": "192.0.2.2",
    "control-plane-protocols": {
      "control-plane-protocol": [
        {
          "type": "ietf-routing:ospf",
          "name": "1",
          "ietf-ospf:ospf": {
            "af": "ipv4",
            "areas": {
              "area": [
                {
                  "area-id": "203.0.113.1",
                  "interfaces": {
                    "interface": [
                      {

```

```

        "name": "eth1",
        "cost": 10
      }
    ]
  }
}
},
"ietf-interfaces:interfaces": {
  "interfaces": {
    "interface": [
      {
        "name": "eth1",
        "ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.11",
              "prefix-length": 24,
            }
          ]
        }
      }
    ]
  }
}
}

```

B.2.2. State Data

The following sections shows possible state data associated the above configuration data. Note that there are three views: the host device's, and each LNE's.

B.2.2.1. Host Device

The following shows state data for the device hosting the example LNEs:

```

{
  "ietf-logical-network-element:logical-network-elements": {
    "logical-network-element": [
      {
        "name": "vnf1",

```

```

        "root": {
        }
    },
    {
        "name": "vnf2",
        "root": {
        }
    }
]
},
"ietf-interfaces:interfaces": {
    "interfaces": {
        "interface": [
            {
                "name": "eth0",
                "type": "iana-if-type:ethernetCsmacd",
                "oper-status": "up",
                "phys-address": "00:01:02:A1:B1:C0",
                "statistics": {
                    "discontinuity-time": "2017-06-26T12:34:56-05:00"
                },
                "ip:ipv4": {
                    "address": [
                        {
                            "ip": "192.0.2.10",
                            "prefix-length": 24,
                        }
                    ]
                }
            }
        ],
        {
            "name": "vnf1:eth1",
            "type": "iana-if-type:ethernetCsmacd",
            "oper-status": "up",
            "phys-address": "00:01:02:A1:B1:C1",
            "statistics": {
                "discontinuity-time": "2017-06-26T12:34:56-05:00"
            }
        },
        {
            "name": "vnf2:eth2",
            "type": "iana-if-type:ethernetCsmacd",
            "oper-status": "up",
            "phys-address": "00:01:02:A1:B1:C2",
            "statistics": {
                "discontinuity-time": "2017-06-26T12:34:56-05:00"
            }
        }
    ]
}

```



```
    }
  ]
}
},

"ietf-system:system-state": {
  "platform": {
    "os-name": "NetworkOS"
  }
},

"ietf-yang-library:modules-state": {
  "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
  "module": [
    {
      "name": "iana-if-type",
      "revision": "2014-05-08",
      "namespace":
        "urn:ietf:params:xml:ns:yang:iana-if-type",
      "conformance-type": "import"
    },
    {
      "name": "ietf-inet-types",
      "revision": "2013-07-15",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-inet-types",
      "conformance-type": "import"
    },
    {
      "name": "ietf-interfaces",
      "revision": "2014-05-08",
      "feature": [
        "arbitrary-names",
        "pre-provisioning"
      ],
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-interfaces",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-ip",
      "revision": "2014-06-16",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-ip",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-logical-network-element",
```

```
    "revision": "2017-03-13",
    "feature": [
      "bind-lne-name"
    ],
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-logical-network-element",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-system",
    "revision": "2014-08-06",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-system",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-yang-library",
    "revision": "2016-06-21",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-yang-library",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-yang-schema-mount",
    "revision": "2017-05-16",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-yang-types",
    "revision": "2013-07-15",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-yang-types",
    "conformance-type": "import"
  }
]
},
"ietf-yang-schema-mount:schema-mounts": {
  "mount-point": [
    {
      "module": "ietf-logical-network-element",
      "label": "root",
      "inline": {}
    }
  ]
}
```

```
}
```

B.2.2.2. Logical Network Element 'vnf1'

The following shows state data for the example LNE with name "vnf1":

```
{
  "ietf-yang-library:modules-state": {
    "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
    "module": [
      {
        "name": "iana-if-type",
        "revision": "2014-05-08",
        "namespace":
          "urn:ietf:params:xml:ns:yang:iana-if-type",
        "conformance-type": "import"
      },
      {
        "name": "ietf-inet-types",
        "revision": "2013-07-15",
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-inet-types",
        "conformance-type": "import"
      },
      {
        "name": "ietf-interfaces",
        "revision": "2014-05-08",
        "feature": [
          "arbitrary-names",
          "pre-provisioning"
        ],
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-interfaces",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-ip",
        "revision": "2014-06-16",
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ip",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-ospf",
        "revision": "2018-03-03",
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ospf",
        "conformance-type": "implement"
      }
    ]
  }
}
```

```
    },
    {
      "name": "ietf-routing",
      "revision": "2018-03-13",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-routing",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-system",
      "revision": "2014-08-06",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-system",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-yang-library",
      "revision": "2016-06-21",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-yang-library",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-yang-types",
      "revision": "2013-07-15",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-yang-types",
      "conformance-type": "import"
    }
  ]
},

"ietf-system:system-state": {
  "platform": {
    "os-name": "NetworkOS"
  }
},

"ietf-routing:routing": {
  "router-id": "192.0.2.1",
  "control-plane-protocols": {
    "control-plane-protocol": [
      {
        "type": "ietf-routing:ospf",
        "name": "1",
        "ietf-ospf:ospf": {
          "af": "ipv4",
          "areas": {
```

```

        "area": [
            {
                "area-id": "203.0.113.1",
                "interfaces": {
                    "interface": [
                        {
                            "name": "eth1",
                            "cost": 10
                        }
                    ]
                }
            }
        ]
    },
    "ietf-interfaces:interfaces": {
        "interfaces": {
            "interface": [
                {
                    "name": "eth1",
                    "type": "iana-if-type:ethernetCsmacd",
                    "oper-status": "up",
                    "phys-address": "00:01:02:A1:B1:C1",
                    "statistics": {
                        "discontinuity-time": "2017-06-26T12:34:56-05:00"
                    },
                    "ip:ipv4": {
                        "address": [
                            {
                                "ip": "192.0.2.11",
                                "prefix-length": 24,
                            }
                        ]
                    }
                }
            ]
        }
    }
}

```

B.2.2.3. Logical Network Element 'vnf2'

The following shows state data for the example LNE with name "vnf2":

```
{
  "ietf-yang-library:modules-state": {
    "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
    "module": [
      {
        "name": "iana-if-type",
        "revision": "2014-05-08",
        "namespace":
          "urn:ietf:params:xml:ns:yang:iana-if-type",
        "conformance-type": "import"
      },
      {
        "name": "ietf-inet-types",
        "revision": "2013-07-15",
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-inet-types",
        "conformance-type": "import"
      },
      {
        "name": "ietf-interfaces",
        "revision": "2014-05-08",
        "feature": [
          "arbitrary-names",
          "pre-provisioning"
        ],
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-interfaces",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-ip",
        "revision": "2014-06-16",
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ip",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-ospf",
        "revision": "2018-03-03",
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ospf",
        "conformance-type": "implement"
      }
    ]
  }
}
```

```
    "name": "ietf-routing",
    "revision": "2018-03-13",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-routing",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-system",
    "revision": "2014-08-06",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-system",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-yang-library",
    "revision": "2016-06-21",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-yang-library",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-yang-types",
    "revision": "2013-07-15",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-yang-types",
    "conformance-type": "import"
  }
]
},
"ietf-system:system-state": {
  "platform": {
    "os-name": "NetworkOS"
  }
},
"ietf-routing:routing": {
  "router-id": "192.0.2.2",
  "control-plane-protocols": {
    "control-plane-protocol": [
      {
        "type": "ietf-routing:ospf",
        "name": "1",
        "ietf-ospf:ospf": {
          "af": "ipv4",
          "areas": {
            "area": [
              {
```

```

        "area-id": "203.0.113.1",
        "interfaces": {
            "interface": [
                {
                    "name": "eth1",
                    "cost": 10
                }
            ]
        }
    }
}
},
"ietf-interfaces:interfaces": {
    "interfaces": {
        "interface": [
            {
                "name": "eth1",
                "type": "iana-if-type:ethernetCsmacd",
                "oper-status": "up",
                "phys-address": "00:01:02:A1:B1:C2",
                "statistics": {
                    "discontinuity-time": "2017-06-26T12:34:56-05:00"
                },
                "ip:ipv4": {
                    "address": [
                        {
                            "ip": "192.0.2.11",
                            "prefix-length": 24,
                        }
                    ]
                }
            }
        ]
    }
}
}

```

Authors' Addresses

Lou Berger
LabN Consulting, L.L.C.

Email: lberger@labn.net

Christan Hopps
Deutsche Telekom

Email: chopps@chopps.org

Acee Lindem
Cisco Systems
301 Midenhall Way
Cary, NC 27513
USA

Email: acee@cisco.com

Dean Bogdanovic

Email: ivandean@gmail.com

Xufeng Liu
Jabil

Email: Xufeng_Liu@jabil.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 20, 2018

L. Berger
LabN Consulting, L.L.C.
C. Hopps
Deutsche Telekom
A. Lindem
Cisco Systems
D. Bogdanovic

X. Liu
Jabil
March 19, 2018

YANG Model for Network Instances
draft-ietf-rtgwg-ni-model-12

Abstract

This document defines a network instance module. This module can be used to manage the virtual resource partitioning that may be present on a network device. Examples of common industry terms for virtual resource partitioning are Virtual Routing and Forwarding (VRF) instances and Virtual Switch Instances (VSIs).

The YANG model in this document conforms to the Network Management Datastore Architecture defined in I-D.ietf-netmod-revised-datastores.

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 September 20, 2018.

Copyright Notice

Copyright (c) 2018 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	2
1.1. Terminology	3
2. Overview	4
3. Network Instances	5
3.1. NI Types and Mount Points	6
3.1.1. Well Known Mount Points	7
3.1.2. NI Type Example	8
3.2. NIs and Interfaces	9
3.3. Network Instance Management	10
3.4. Network Instance Instantiation	12
4. Security Considerations	13
5. IANA Considerations	14
6. Network Instance Model	14
7. References	20
7.1. Normative References	20
7.2. Informative References	22
Appendix A. Acknowledgments	23
Appendix B. Example NI usage	23
B.1. Configuration Data	23
B.2. State Data - Non-NMDA Version	27
B.3. State Data - NMDA Version	33
Authors' Addresses	42

1. Introduction

This document defines the second of two new modules that are defined to support the configuration and operation of network-devices that allow for the partitioning of resources from both, or either, management and networking perspectives. Both leverage the YANG functionality enabled by YANG Schema Mount [I-D.ietf-netmod-schema-mount].

The YANG model in this document conforms to the Network Management Datastore Architecture defined in the [I-D.ietf-netmod-revised-datastores].

The first form of resource partitioning provides a logical partitioning of a network device where each partition is separately managed as essentially an independent network element which is 'hosted' by the base network device. These hosted network elements are referred to as logical network elements, or LNEs, and are supported by the logical-network-element module defined in [I-D.ietf-rtgwg-lne-model]. That module is used to identify LNEs and associate resources from the network-device with each LNE. LNEs themselves are represented in YANG as independent network devices; each accessed independently. Examples of vendor terminology for an LNE include logical system or logical router, and virtual switch, chassis, or fabric.

The second form, which is defined in this document, provides support for what is commonly referred to as Virtual Routing and Forwarding (VRF) instances as well as Virtual Switch Instances (VSI), see [RFC4026] and [RFC4664]. In this form of resource partitioning, multiple control plane and forwarding/bridging instances are provided by and managed via a single (physical or logical) network device. This form of resource partitioning is referred to as a Network Instance and is supported by the network-instance module defined below. Configuration and operation of each network-instance is always via the network device and the network-instance module.

One notable difference between the LNE model and the NI model is that the NI model provides a framework for VRF and VSI management. This document envisions the separate definition of VRF and VSI, i.e., L3 and L2 VPN, technology specific models. An example of such can be found in the emerging L3VPN model defined in [I-D.ietf-bess-l3vpn-yang] and the examples discussed below.

1.1. Terminology

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.

Readers are expected to be familiar with terms and concepts of YANG [RFC7950] and YANG Schema Mount [I-D.ietf-netmod-schema-mount].

This document uses the graphical representation of data models defined in [I-D.ietf-netmod-yang-tree-diagrams].

2. Overview

In this document, we consider network devices that support protocols and functions defined within the IETF, e.g, routers, firewalls, and hosts. Such devices may be physical or virtual, e.g., a classic router with custom hardware or one residing within a server-based virtual machine implementing a virtual network function (VNF). Each device may sub-divide their resources into logical network elements (LNEs) each of which provides a managed logical device. Examples of vendor terminology for an LNE include logical system or logical router, and virtual switch, chassis, or fabric. Each LNE may also support virtual routing and forwarding (VRF) and virtual switching instance (VSI) functions, which are referred to below as a network instances (NIs). This breakdown is represented in Figure 1.

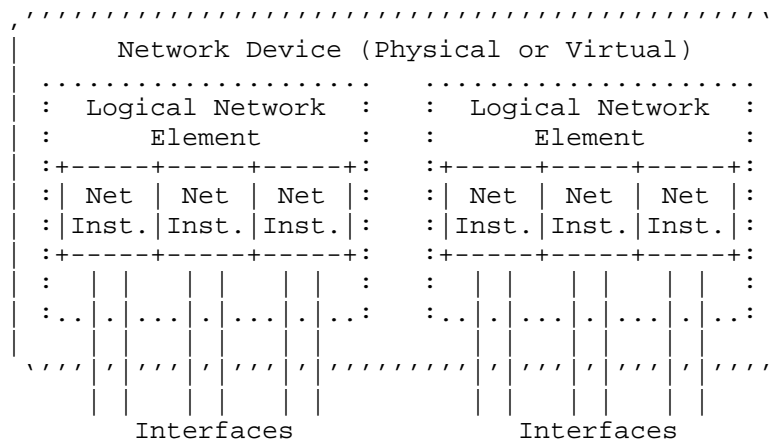


Figure 1: Module Element Relationships

A model for LNEs is described in [I-D.ietf-rtgwg-lne-model] and the model for NIs is covered in this document in Section 3.

The current interface management model [I-D.ietf-netmod-rfc7223bis] is impacted by the definition of LNEs and NIs. This document and [I-D.ietf-rtgwg-lne-model] define augmentations to the interface module to support LNEs and NIs.

The network instance model supports the configuration of VRFs and VSIs. Each instance is supported by information that relates to the device, for example the route target used when advertising VRF routes via the mechanisms defined in [RFC4364], and information that relates to the internal operation of the NI, for example for routing protocols [I-D.ietf-netmod-rfc8022bis] and OSPF [I-D.ietf-ospf-yang].

This document defines the network-instance module that provides a basis for the management of both types of information.

NI information that relates to the device, including the assignment of interfaces to NIs, is defined as part of this document. The defined module also provides a placeholder for the definition of NI-technology specific information both at the device level and for NI internal operation. Information related to NI internal operation is supported via schema mount [I-D.ietf-netmod-schema-mount] and mounting appropriate modules under the mount point. Well known mount points are defined for L3VPN, L2VPN, and L2+L3VPN NI types.

3. Network Instances

The network instance container is used to represent virtual routing and forwarding instances (VRFs) and virtual switching instances (VSIs). VRFs and VSIs are commonly used to isolate routing and switching domains, for example to create virtual private networks, each with their own active protocols and routing/switching policies. The model supports both core/provider and virtual instances. Core/provider instance information is accessible at the top level of the server, while virtual instance information is accessible under the root schema mount points.

```

module: ietf-network-instance
  +--rw network-instances
    +--rw network-instance* [name]
      +--rw name                string
      +--rw enabled?            boolean
      +--rw description?        string
      +--rw (ni-type)?
      +--rw (root-type)
        +--:(vrf-root)
          | +--mp vrf-root
        +--:(vsi-root)
          | +--mp vsi-root
        +--:(vv-root)
          +--mp vv-root
  augment /if:interfaces/if:interface:
    +--rw bind-ni-name? -> /network-instances/network-instance/name
  augment /if:interfaces/if:interface/ip:ipv4:
    +--rw bind-ni-name? -> /network-instances/network-instance/name
  augment /if:interfaces/if:interface/ip:ipv6:
    +--rw bind-ni-name? -> /network-instances/network-instance/name

notifications:
  +---n bind-ni-name-failed
    +--ro name                -> /if:interfaces/interface/name
    +--ro interface
      | +--ro bind-ni-name?
      | -> /if:interfaces/interface/ni:bind-ni-name
    +--ro ipv4
      | +--ro bind-ni-name?
      | -> /if:interfaces/interface/ip:ipv4/ni:bind-ni-name
    +--ro ipv6
      | +--ro bind-ni-name?
      | -> /if:interfaces/interface/ip:ipv6/ni:bind-ni-name
    +--ro error-info?        string

```

A network instance is identified by a 'name' string. This string is used both as an index within the network-instance module and to associate resources with a network instance as shown above in the interface augmentation. The ni-type and root-type choice statements are used to support different types of L2 and L3 VPN technologies. The bind-ni-name-failed notification is used in certain failure cases.

3.1. NI Types and Mount Points

The network-instance module is structured to facilitate the definition of information models for specific types of VRFs and VSIs using augmentations. For example, the information needed to support

VPLS, VxLAN and EVPN based L2VPNs are likely to be quite different. Example models under development that could be restructured to take advantage on NIs include, for L3VPNs [I-D.ietf-bess-l3vpn-yang] and for L2VPNs [I-D.ietf-bess-l2vpn-yang].

Documents defining new YANG models for the support of specific types of network instances should augment the network instance module. The basic structure that should be used for such augmentations include a case statement, with containers for configuration and state data and finally, when needed, a type specific mount point. Generally ni types, are expected to not need to define type specific mount points, but rather reuse one of the well known mount point, as defined in the next section. The following is an example type specific augmentation:

```
augment "/ni:network-instances/ni:network-instance/ni:ni-type" {  
  case l3vpn {  
    container l3vpn {  
      ...  
    }  
    container l3vpn-state {  
      ...  
    }  
  }  
}
```

3.1.1. Well Known Mount Points

YANG Schema Mount, [I-D.ietf-netmod-schema-mount], identifies mount points by name within a module. This definition allows for the definition of mount points whose schema can be shared across ni-types. As discussed above, ni-types largely differ in the configuration information needed in the core/top level instance to support the NI, rather than in the information represented within an NI. This allows the use of shared mount points across certain NI types.

The expectation is that there are actually very few different schema that need to be defined to support NIs on an implementation. In particular, it is expected that the following three forms of NI schema are needed, and each can be defined with a well known mount point that can be reused by future modules defining ni-types.

The three well known mount points are:

vrf-root
vrf-root is intended for use with L3VPN type ni-types.

vsi-root

vsi-root is intended for use with L2VPN type ni-types.

vv-root

vv-root is intended for use with ni-types that simultaneously support L2VPN bridging and L3VPN routing capabilities.

Future model definitions should use the above mount points whenever possible. When a well known mount point isn't appropriate, a model may define a type specific mount point via augmentation.

3.1.2. NI Type Example

The following is an example of an L3VPN VRF using a hypothetical augmentation to the networking instance schema defined in [I-D.ietf-bess-l3vpn-yang]. More detailed examples can be found in Appendix B.

```

module: ietf-network-instance
  +--rw network-instances
    +--rw network-instance* [name]
      +--rw name string
      +--rw enabled? boolean
      +--rw description? string
      +--rw (ni-type)?
        |   +--:(l3vpn)
        |     +--rw l3vpn:l3vpn
        |         |   ... // config data
        |         +--ro l3vpn:l3vpn-state
        |             |   ... // state data
        +--rw (root-type)
          +--:(vrf-root)
            +--mp vrf-root
              +--rw rt:routing/
                +--rw router-id? yang:dotted-quad
                +--rw control-plane-protocols
                  +--rw control-plane-protocol* [type name]
                  +--rw ospf:ospf/
                    +--rw area* [area-id]
                    +--rw interfaces
                      +--rw interface* [name]
                        +--rw name if:interface-ref
                        +--rw cost? uint16
              +--ro if:interfaces@
                |   ...

```

This shows YANG Routing Management [I-D.ietf-netmod-rfc8022bis] and YANG OSPF [I-D.ietf-ospf-yang] as mounted modules. The mounted

modules can reference interface information via a parent-reference to the containers defined in [I-D.ietf-netmod-rfc7223bis].

3.2. NIs and Interfaces

Interfaces are a crucial part of any network device's configuration and operational state. They generally include a combination of raw physical interfaces, link-layer interfaces, addressing configuration, and logical interfaces that may not be tied to any physical interface. Several system services, and layer 2 and layer 3 protocols may also associate configuration or operational state data with different types of interfaces (these relationships are not shown for simplicity). The interface management model is defined by [I-D.ietf-netmod-rfc7223bis].

As shown below, the network-instance module augments the existing interface management model by adding a name which is used on interface or sub-interface types to identify an associated network instance. Similarly, this name is also added for IPv4 and IPv6 types, as defined in [I-D.ietf-netmod-rfc7277bis].

The following is an example of envisioned usage. The interfaces container includes a number of commonly used components as examples:

```
module: ietf-interfaces
  +--rw interfaces
  |   +--rw interface* [name]
  |   |   +--rw name                                string
  |   |   +--rw ip:ipv4!
  |   |   |   +--rw ip:enabled?                      boolean
  |   |   |   +--rw ip:forwarding?                   boolean
  |   |   |   +--rw ip:mtu?                          uint16
  |   |   |   +--rw ip:address* [ip]
  |   |   |   |   +--rw ip:ip                        inet:ipv4-address-no-zone
  |   |   |   |   +--rw (ip:subnet)
  |   |   |   |   |   +--:(ip:prefix-length)
  |   |   |   |   |   |   +--rw ip:prefix-length?    uint8
  |   |   |   |   |   +--:(ip:netmask)
  |   |   |   |   |   |   +--rw ip:netmask?          yang:dotted-quad
  |   |   |   +--rw ip:neighbor* [ip]
  |   |   |   |   +--rw ip:ip                        inet:ipv4-address-no-zone
  |   |   |   |   +--rw ip:link-layer-address        yang:phys-address
  |   |   |   +--rw ni:bind-network-instance-name?  string
  |   +--rw ni:bind-network-instance-name?          string
```

The [I-D.ietf-netmod-rfc7223bis] defined interface model is structured to include all interfaces in a flat list, without regard to virtual instances (e.g., VRFs) supported on the device. The bind-

network-instance-name leaf provides the association between an interface and its associated NI (e.g., VRF or VSI). Note that as currently defined, to assign an interface to both an LNE and NI, the interface would first be assigned to the LNE using the mechanisms defined in [I-D.ietf-rtgwg-lne-model] and then within that LNE's interface module, the LNE's representation of that interface would be assigned to an NI.

3.3. Network Instance Management

Modules that may be used to represent network instance information will be available under the ni-type specific 'root' mount point. The "shared-schema" method defined in the "ietf-yang-schema-mount" module [I-D.ietf-netmod-schema-mount] MUST be used to identify accessible modules. A future version of this document could relax this requirement. Mounted modules SHOULD be defined with access, via the appropriate schema mount parent-references [I-D.ietf-netmod-schema-mount], to device resources such as interfaces. An implementation MAY choose to restrict parent referenced information to information related to a specific instance, e.g., only allowing references to interfaces that have a "bind-network-instance-name" which is identical to the instance's "name".

All modules that represent control-plane and data-plane information may be present at the 'root' mount point, and be accessible via paths modified per [I-D.ietf-netmod-schema-mount]. The list of available modules is expected to be implementation dependent, as is the method used by an implementation to support NIs.

For example, the following could be used to define the data organization of the example NI shown in Section 3.1.2:

```
"ietf-yang-schema-mount:schema-mounts": {
  "mount-point": [
    {
      "module": "ietf-network-instance",
      "label": "vrf-root",
      "shared-schema": {
        "parent-reference": [
          "/*[namespace-uri() = 'urn:ietf:...:ietf-interfaces']"
        ]
      }
    ]
  ]
}
```

Module data identified according to the ietf-yang-schema-mount module will be instantiated under the mount point identified under "mount-

point". These modules will be able to reference information for nodes belonging to top-level modules that are identified under "parent-reference". Parent referenced information is available to clients via their top level paths only, and not under the associated mount point.

To allow a client to understand the previously mentioned instance restrictions on parent referenced information, an implementation MAY represent such restrictions in the "parent-reference" leaf-list. For example:

```
"namespace": [
  {
    "prefix": "if",
    "uri": "urn:ietf:params:xml:ns:yang:ietf-interfaces"
  },
  {
    "prefix": "ni",
    "uri": "urn:ietf:params:xml:ns:yang:ietf-network-instance"
  }
],
"mount-point": [
  {
    "module": "ietf-network-instance",
    "label": "vrf-root",
    "shared-schema": {
      "parent-reference": [
        "/if:interfaces/if:interface
          [ni:bind-network-instance-name = current()/../ni:name]",
        "/if:interfaces/if:interface/ip:ipv4
          [ni:bind-network-instance-name = current()/../ni:name]",
        "/if:interfaces/if:interface/ip:ipv6
          [ni:bind-network-instance-name = current()/../ni:name]"
      ]
    }
  }
],
```

The same such "parent-reference" restrictions for non-NMDA implementations can be represented based on the [RFC7223] and [RFC7277] as:

```

"namespace": [
  {
    "prefix": "if",
    "uri": "urn:ietf:params:xml:ns:yang:ietf-interfaces"
  },
  {
    "prefix": "ni",
    "uri": "urn:ietf:params:xml:ns:yang:ietf-network-instance"
  }
],
"mount-point": [
  {
    "module": "ietf-network-instance",
    "label": "vrf-root",
    "shared-schema": {
      "parent-reference": [
        "/if:interfaces/if:interface
          [ni:bind-network-instance-name = current()/../ni:name]",
        "/if:interfaces-state/if:interface
          [if:name = /if:interfaces/if:interface
            [ni:bind-ni-name = current()/../ni:name]/if:name]",
        "/if:interfaces/if:interface/ip:ipv4
          [ni:bind-network-instance-name = current()/../ni:name]",
        "/if:interfaces-state/if:interface/ip:ipv4
          [if:name = /if:interfaces/if:interface/ip:ipv4
            [ni:bind-ni-name = current()/../ni:name]/if:name]",
        "/if:interfaces/if:interface/ip:ipv6
          [ni:bind-network-instance-name = current()/../ni:name]",
        "/if:interfaces-state/if:interface/ip:ipv6
          [if:name = /if:interfaces/if:interface/ip:ipv4
            [ni:bind-ni-name = current()/../ni:name]/if:name]"
      ]
    }
  }
],

```

3.4. Network Instance Instantiation

Network instances may be controlled by clients using existing list operations. When a list entry is created, a new instance is instantiated. The models mounted under an NI root are expected to be dependent on the server implementation. When a list entry is deleted, an existing network instance is destroyed. For more information, see [RFC7950] Section 7.8.6.

Once instantiated, host network device resources can be associated with the new NI. As previously mentioned, this document augments ietf-interfaces with the bind-ni-name leaf to support such

associations for interfaces. When a bind-ni-name is set to a valid NI name, an implementation MUST take whatever steps are internally necessary to assign the interface to the NI or provide an error message (defined below) with an indication of why the assignment failed. It is possible for the assignment to fail while processing the set operation, or after asynchronous processing. Error notification in the latter case is supported via a notification.

4. Security Considerations

The YANG modules specified in this document define a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

There are two different sets of security considerations to consider in the context of this document. One set is security related to information contained within mounted modules. The security considerations for mounted modules are not substantively changed based on the information being accessible within the context of an NI. For example, when considering the modules defined in [I-D.ietf-netmod-rfc8022bis], the security considerations identified in that document are equally applicable, whether those modules are accessed at a server's root or under an NI instance's root node.

The second area for consideration is information contained in the NI module itself. NI information represents network configuration and route distribution policy information. As such, the security of this information is important, but it is fundamentally no different than any other interface or routing configuration information that has already been covered in [I-D.ietf-netmod-rfc7223bis] and [I-D.ietf-netmod-rfc8022bis].

The vulnerable "config true" parameters and subtrees are the following:

/network-instances/network-instance: This list specifies the network instances and the related control plane protocols configured on a device.

/if:interfaces/if:interface/*/bind-network-instance-name: This leaf indicates the NI instance to which an interface is assigned.

Unauthorized access to any of these lists can adversely affect the routing subsystem of both the local device and the network. This may lead to network malfunctions, delivery of packets to inappropriate destinations and other problems.

5. IANA Considerations

This document registers a URI in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration is requested to be made.

URI: urn:ietf:params:xml:ns:yang:ietf-network-instance

Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.

This document registers a YANG module in the YANG Module Names registry [RFC6020].

name: ietf-network-instance
namespace: urn:ietf:params:xml:ns:yang:ietf-network-instance
prefix: ni
reference: RFC XXXX

6. Network Instance Model

The structure of the model defined in this document is described by the YANG module below.

```
<CODE BEGINS> file "ietf-network-instance@2018-03-20.yang"
module ietf-network-instance {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-network-instance";
  prefix ni;

  // import some basic types

  import ietf-interfaces {
    prefix if;
    reference "draft-ietf-netmod-rfc7223bis: A YANG Data Model
              for Interface Management";
  }
  import ietf-ip {
    prefix ip;
```

```
reference "draft-ietf-netmod-rfc7277bis: A YANG Data Model
        for IP Management";
}
import ietf-yang-schema-mount {
  prefix yangmnt;
  reference "draft-ietf-netmod-schema-mount: YANG Schema Mount";
  // RFC Ed.: Please replace this draft name with the
  // corresponding RFC number
}

organization
  "IETF Routing Area (rtgwg) Working Group";
contact
  "WG Web:    <http://tools.ietf.org/wg/rtgwg/>
   WG List:   <mailto:rtgwg@ietf.org>

   Author:    Lou Berger
               <mailto:lberger@labn.net>
   Author:    Christan Hopps
               <mailto:chopps@chopps.org>
   Author:    Acee Lindem
               <mailto:acee@cisco.com>
   Author:    Dean Bogdanovic
               <mailto:ivandean@gmail.com>";

description
  "This module is used to support multiple network instances
   within a single physical or virtual device.  Network
   instances are commonly known as VRFs (virtual routing
   and forwarding) and VSIs (virtual switching instances).

   Copyright (c) 2017 IETF Trust and the persons
   identified as authors of the code.  All rights reserved.

   Redistribution and use in source and binary forms, with or
   without modification, is permitted pursuant to, and subject
   to the license terms contained in, the Simplified BSD License
   set forth in Section 4.c of the IETF Trust's Legal Provisions
   Relating to IETF Documents
   (http://trustee.ietf.org/license-info).

   This version of this YANG module is part of RFC XXXX; see
   the RFC itself for full legal notices.";

// RFC Ed.: replace XXXX with actual RFC number and remove
// this note
// RFC Ed.: please update TBD

revision 2018-03-20 {
```



```
    description
      "Initial revision.";
    reference "RFC TBD";
  }

  // top level device definition statements

  container network-instances {
    description
      "Network instances each of which consists of a
      VRFs (virtual routing and forwarding) and/or
      VSIs (virtual switching instances).";
    reference "draft-ietf-rtgwg-rfc8022bis - A YANG Data Model
      for Routing Management";
    list network-instance {
      key "name";
      description
        "List of network-instances.";
      leaf name {
        type string;
        mandatory true;
        description
          "device scoped identifier for the network
          instance.";
      }
      leaf enabled {
        type boolean;
        default "true";
        description
          "Flag indicating whether or not the network
          instance is enabled.";
      }
      leaf description {
        type string;
        description
          "Description of the network instance
          and its intended purpose.";
      }
      choice ni-type {
        description
          "This node serves as an anchor point for different types
          of network instances. Each 'case' is expected to
          differ in terms of the information needed in the
          parent/core to support the NI, and may differ in their
          mounted schema definition. When the mounted schema is
          not expected to be the same for a specific type of NI
          a mount point should be defined.";
      }
    }
  }
```

```

choice root-type {
  mandatory true;
  description
    "Well known mount points.";
  container vrf-root {
    description
      "Container for mount point.";
    yangmnt:mount-point "vrf-root" {
      description
        "Root for L3VPN type models. This will typically
        not be an inline type mount point.";
    }
  }
  container vsi-root {
    description
      "Container for mount point.";
    yangmnt:mount-point "vsi-root" {
      description
        "Root for L2VPN type models. This will typically
        not be an inline type mount point.";
    }
  }
  container vv-root {
    description
      "Container for mount point.";
    yangmnt:mount-point "vv-root" {
      description
        "Root models that support both L2VPN type bridging
        and L3VPN type routing. This will typically
        not be an inline type mount point.";
    }
  }
}
}
}

// augment statements

augment "/if:interfaces/if:interface" {
  description
    "Add a node for the identification of the network
    instance associated with the information configured
    on a interface.

    Note that a standard error will be returned if the
    identified leafref isn't present. If an interfaces cannot
    be assigned for any other reason, the operation SHALL fail
    with an error-tag of 'operation-failed' and an

```

```
    error-app-tag of 'ni-assignment-failed'. A meaningful
    error-info that indicates the source of the assignment
    failure SHOULD also be provided.";
  leaf bind-ni-name {
    type leafref {
      path "/network-instances/network-instance/name";
    }
    description
      "Network Instance to which an interface is bound.";
  }
}
augment "/if:interfaces/if:interface/ip:ipv4" {
  description
    "Add a node for the identification of the network
    instance associated with the information configured
    on an IPv4 interface.

    Note that a standard error will be returned if the
    identified leafref isn't present. If an interfaces cannot
    be assigned for any other reason, the operation SHALL fail
    with an error-tag of 'operation-failed' and an
    error-app-tag of 'ni-assignment-failed'. A meaningful
    error-info that indicates the source of the assignment
    failure SHOULD also be provided.";
  leaf bind-ni-name {
    type leafref {
      path "/network-instances/network-instance/name";
    }
    description
      "Network Instance to which IPv4 interface is bound.";
  }
}
augment "/if:interfaces/if:interface/ip:ipv6" {
  description
    "Add a node for the identification of the network
    instance associated with the information configured
    on an IPv6 interface.

    Note that a standard error will be returned if the
    identified leafref isn't present. If an interfaces cannot
    be assigned for any other reason, the operation SHALL fail
    with an error-tag of 'operation-failed' and an
    error-app-tag of 'ni-assignment-failed'. A meaningful
    error-info that indicates the source of the assignment
    failure SHOULD also be provided.";
  leaf bind-ni-name {
    type leafref {
      path "/network-instances/network-instance/name";
```

```
    }
    description
      "Network Instance to which IPv6 interface is bound.";
  }
}

// notification statements

notification bind-ni-name-failed {
  description
    "Indicates an error in the association of an interface to an
    NI. Only generated after success is initially returned when
    bind-ni-name is set.

    Note: some errors may need to be reported for multiple
    associations, e.g., a single error may need to be reported
    for an IPv4 and an IPv6 bind-ni-name.

    At least one container with a bind-ni-name leaf MUST be
    included in this notification.";
  leaf name {
    type leafref {
      path "/if:interfaces/if:interface/if:name";
    }
    mandatory true;
    description
      "Contains the interface name associated with the
      failure.";
  }
  container interface {
    description
      "Generic interface type.";
    leaf bind-ni-name {
      type leafref {
        path "/if:interfaces/if:interface/ni:bind-ni-name";
      }
      description
        "Contains the bind-ni-name associated with the
        failure.";
    }
  }
}
container ipv4 {
  description
    "IPv4 interface type.";
  leaf bind-ni-name {
    type leafref {
      path "/if:interfaces/if:interface"
        + "/ip:ipv4/ni:bind-ni-name";
    }
  }
}
```

```
    }
    description
      "Contains the bind-ni-name associated with the
       failure.";
  }
}
container ipv6 {
  description
    "IPv6 interface type.";
  leaf bind-ni-name {
    type leafref {
      path "/if:interfaces/if:interface"
        + "/ip:ipv6/ni:bind-ni-name";
    }
    description
      "Contains the bind-ni-name associated with the
       failure.";
  }
}
leaf error-info {
  type string;
  description
    "Optionally, indicates the source of the assignment
     failure.";
}
}
}
<CODE ENDS>
```

7. References

7.1. Normative References

- [I-D.ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K.,
and R. Wilton, "Network Management Datastore
Architecture", draft-ietf-netmod-revised-datastores-10
(work in progress), January 2018.
- [I-D.ietf-netmod-rfc7223bis]
Bjorklund, M., "A YANG Data Model for Interface
Management", draft-ietf-netmod-rfc7223bis-03 (work in
progress), January 2018.
- [I-D.ietf-netmod-rfc7277bis]
Bjorklund, M., "A YANG Data Model for IP Management",
draft-ietf-netmod-rfc7277bis-03 (work in progress),
January 2018.

- [I-D.ietf-netmod-schema-mount]
Bjorklund, M. and L. Lhotka, "YANG Schema Mount", draft-ietf-netmod-schema-mount-08 (work in progress), October 2017.
- [I-D.ietf-netmod-yang-tree-diagrams]
Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-06 (work in progress), February 2018.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<https://www.rfc-editor.org/info/rfc6536>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.

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

7.2. Informative References

- [I-D.ietf-bess-l2vpn-yang]
Shah, H., Brissette, P., Chen, I., Hussain, I., Wen, B., and K. Tiruveedhula, "YANG Data Model for MPLS-based L2VPN", draft-ietf-bess-l2vpn-yang-08 (work in progress), February 2018.
- [I-D.ietf-bess-l3vpn-yang]
Jain, D., Patel, K., Brissette, P., Li, Z., Zhuang, S., Liu, X., Haas, J., Esale, S., and B. Wen, "Yang Data Model for BGP/MPLS L3 VPNs", draft-ietf-bess-l3vpn-yang-02 (work in progress), October 2017.
- [I-D.ietf-netmod-rfc8022bis]
Lhotka, L., Lindem, A., and Y. Qu, "A YANG Data Model for Routing Management (NMDA Version)", draft-ietf-netmod-rfc8022bis-11 (work in progress), January 2018.
- [I-D.ietf-ospf-yang]
Yeung, D., Qu, Y., Zhang, Z., Chen, I., and A. Lindem, "Yang Data Model for OSPF Protocol", draft-ietf-ospf-yang-10 (work in progress), March 2018.
- [I-D.ietf-rtgwg-device-model]
Lindem, A., Berger, L., Bogdanovic, D., and C. Hopps, "Network Device YANG Logical Organization", draft-ietf-rtgwg-device-model-02 (work in progress), March 2017.
- [I-D.ietf-rtgwg-lne-model]
Berger, L., Hopps, C., Lindem, A., Bogdanovic, D., and X. Liu, "YANG Model for Logical Network Elements", draft-ietf-rtgwg-lne-model-09 (work in progress), March 2018.
- [RFC4026] Andersson, L. and T. Madsen, "Provider Provisioned Virtual Private Network (VPN) Terminology", RFC 4026, DOI 10.17487/RFC4026, March 2005, <<https://www.rfc-editor.org/info/rfc4026>>.
- [RFC4364] Rosen, E. and Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4364, DOI 10.17487/RFC4364, February 2006, <<https://www.rfc-editor.org/info/rfc4364>>.

- [RFC4664] Andersson, L., Ed. and E. Rosen, Ed., "Framework for Layer 2 Virtual Private Networks (L2VPNs)", RFC 4664, DOI 10.17487/RFC4664, September 2006, <<https://www.rfc-editor.org/info/rfc4664>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<https://www.rfc-editor.org/info/rfc7223>>.
- [RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <<https://www.rfc-editor.org/info/rfc7277>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8022] Lhotka, L. and A. Lindem, "A YANG Data Model for Routing Management", RFC 8022, DOI 10.17487/RFC8022, November 2016, <<https://www.rfc-editor.org/info/rfc8022>>.

Appendix A. Acknowledgments

The Routing Area Yang Architecture design team members included Acee Lindem, Anees Shaikh, Christian Hopps, Dean Bogdanovic, Lou Berger, Qin Wu, Rob Shakir, Stephane Litkowski, and Yan Gang. Useful review comments were also received by Martin Bjorklund and John Scudder.

This document was motivated by, and derived from, [I-D.ietf-rtgwg-device-model].

Thanks for AD and IETF last call comments from Alia Atlas, Liang Xia, Benoit Claise, and Adam Roach.

The RFC text was produced using Marshall Rose's xml2rfc tool.

Appendix B. Example NI usage

The following subsections provide example uses of NIs.

B.1. Configuration Data

The following shows an example where two customer specific network instances are configured:

```
{
  "ietf-network-instance:network-instances": {
    "network-instance": [
```



```

{
  "name": "vrf-red",
  "vrf-root": {
    "ietf-routing:routing": {
      "router-id": "192.0.2.1",
      "control-plane-protocols": {
        "control-plane-protocol": [
          {
            "type": "ietf-routing:ospf",
            "name": "1",
            "ietf-ospf:ospf": {
              "af": "ipv4",
              "areas": {
                "area": [
                  {
                    "area-id": "203.0.113.1",
                    "interfaces": {
                      "interface": [
                        {
                          "name": "eth1",
                          "cost": 10
                        }
                      ]
                    }
                  }
                ]
              }
            }
          ]
        }
      }
    }
  },
  {
    "name": "vrf-blue",
    "vrf-root": {
      "ietf-routing:routing": {
        "router-id": "192.0.2.2",
        "control-plane-protocols": {
          "control-plane-protocol": [
            {
              "type": "ietf-routing:ospf",
              "name": "1",
              "ietf-ospf:ospf": {
                "af": "ipv4",
                "areas": {
                  "area": [

```

```

        {
          "area-id": "203.0.113.1",
          "interfaces": {
            "interface": [
              {
                "name": "eth2",
                "cost": 10
              }
            ]
          }
        }
      ]
    }
  ]
},
"ietf-interfaces:interfaces": {
  "interfaces": {
    "interface": [
      {
        "name": "eth0",
        "ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.10",
              "prefix-length": 24,
            }
          ]
        },
        "ip:ipv6": {
          "address": [
            {
              "ip": "2001:db8:0:2::10",
              "prefix-length": 64,
            }
          ]
        }
      },
      {
        "name": "eth1",
        "ip:ipv4": {

```

```

        "address": [
            {
                "ip": "192.0.2.11",
                "prefix-length": 24,
            }
        ],
    },
    "ip:ipv6": {
        "address": [
            {
                "ip": "2001:db8:0:2::11",
                "prefix-length": 64,
            }
        ]
    },
    "ni:bind-network-instance-name": "vrf-red"
},
{
    "name": "eth2",
    "ip:ipv4": {
        "address": [
            {
                "ip": "192.0.2.11",
                "prefix-length": 24,
            }
        ]
    },
    "ip:ipv6": {
        "address": [
            {
                "ip": "2001:db8:0:2::11",
                "prefix-length": 64,
            }
        ]
    },
    "ni:bind-network-instance-name": "vrf-blue"
}
]
}
},
"ietf-system:system": {
    "authentication": {
        "user": [
            {
                "name": "john",
                "password": "$0$password"
            }
        ]
    }
}

```

```

    }
  }
}

```

B.2. State Data - Non-NMDA Version

The following shows state data for the configuration example above based on [RFC7223], [RFC7277], and [RFC8022].

```

{
  "ietf-network-instance:network-instances": {
    "network-instance": [
      {
        "name": "vrf-red",
        "vrf-root": {
          "ietf-yang-library:modules-state": {
            "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
            "module": [
              {
                "name": "ietf-yang-library",
                "revision": "2016-06-21",
                "namespace":
                  "urn:ietf:params:xml:ns:yang:ietf-yang-library",
                "conformance-type": "implement"
              },
              {
                "name": "ietf-ospf",
                "revision": "2018-03-03",
                "namespace":
                  "urn:ietf:params:xml:ns:yang:ietf-ospf",
                "conformance-type": "implement"
              },
              {
                "name": "ietf-routing",
                "revision": "2018-03-13",
                "namespace":
                  "urn:ietf:params:xml:ns:yang:ietf-routing",
                "conformance-type": "implement"
              }
            ]
          }
        },
        "ietf-routing:routing-state": {
          "router-id": "192.0.2.1",
          "control-plane-protocols": {
            "control-plane-protocol": [
              {
                "type": "ietf-routing:ospf",

```

```

        "name": "1",
        "ietf-ospf:ospf": {
            "af": "ipv4",
            "areas": {
                "area": [
                    {
                        "area-id": "203.0.113.1",
                        "interfaces": {
                            "interface": [
                                {
                                    "name": "eth1",
                                    "cost": 10
                                }
                            ]
                        }
                    }
                ]
            }
        }
    },
    {
        "name": "vrf-blue",
        "vrf-root": {
            "ietf-yang-library:modules-state": {
                "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
                "module": [
                    {
                        "name": "ietf-yang-library",
                        "revision": "2016-06-21",
                        "namespace":
                            "urn:ietf:params:xml:ns:yang:ietf-yang-library",
                        "conformance-type": "implement"
                    },
                    {
                        "name": "ietf-ospf",
                        "revision": "2018-03-03",
                        "namespace":
                            "urn:ietf:params:xml:ns:yang:ietf-ospf",
                        "conformance-type": "implement"
                    },
                    {
                        "name": "ietf-routing",
                        "revision": "2018-03-13",

```

```

        "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-routing",
        "conformance-type": "implement"
    }
]
},
"ietf-routing:routing-state": {
    "router-id": "192.0.2.2",
    "control-plane-protocols": {
        "control-plane-protocol": [
            {
                "type": "ietf-routing:ospf",
                "name": "1",
                "ietf-ospf:ospf": {
                    "af": "ipv4",
                    "areas": {
                        "area": [
                            {
                                "area-id": "203.0.113.1",
                                "interfaces": {
                                    "interface": [
                                        {
                                            "name": "eth2",
                                            "cost": 10
                                        }
                                    ]
                                }
                            }
                        ]
                    }
                }
            }
        ]
    }
}
},
"ietf-interfaces:interfaces-state": {
    "interfaces": {
        "interface": [
            {
                "name": "eth0",
                "type": "iana-if-type:ethernetCsmacd",
                "oper-status": "up",
                "phys-address": "00:01:02:A1:B1:C0",

```

```
    "statistics": {
      "discontinuity-time": "2017-06-26T12:34:56-05:00"
    },
    "ip:ipv4": {
      "address": [
        {
          "ip": "192.0.2.10",
          "prefix-length": 24,
        }
      ]
    }
    "ip:ipv6": {
      "address": [
        {
          "ip": "2001:db8:0:2::10",
          "prefix-length": 64,
        }
      ]
    }
  },
  {
    "name": "eth1",
    "type": "iana-if-type:ethernetCsmacd",
    "oper-status": "up",
    "phys-address": "00:01:02:A1:B1:C1",
    "statistics": {
      "discontinuity-time": "2017-06-26T12:34:56-05:00"
    },
    "ip:ipv4": {
      "address": [
        {
          "ip": "192.0.2.11",
          "prefix-length": 24,
        }
      ]
    }
    "ip:ipv6": {
      "address": [
        {
          "ip": "2001:db8:0:2::11",
          "prefix-length": 64,
        }
      ]
    }
  }
},
{
  "name": "eth2",
  "type": "iana-if-type:ethernetCsmacd",
```

```
    "oper-status": "up",
    "phys-address": "00:01:02:A1:B1:C2",
    "statistics": {
      "discontinuity-time": "2017-06-26T12:34:56-05:00"
    },
    "ip:ipv4": {
      "address": [
        {
          "ip": "192.0.2.11",
          "prefix-length": 24,
        }
      ]
    }
    "ip:ipv6": {
      "address": [
        {
          "ip": "2001:db8:0:2::11",
          "prefix-length": 64,
        }
      ]
    }
  }
}
},
"ietf-system:system-state": {
  "platform": {
    "os-name": "NetworkOS"
  }
}

"ietf-yang-library:modules-state": {
  "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
  "module": [
    {
      "name": "iana-if-type",
      "revision": "2014-05-08",
      "namespace":
        "urn:ietf:params:xml:ns:yang:iana-if-type",
      "conformance-type": "import"
    },
    {
      "name": "ietf-inet-types",
      "revision": "2013-07-15",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-inet-types",
      "conformance-type": "import"
    }
  ]
}
```



```
    },
    {
      "name": "ietf-interfaces",
      "revision": "2014-05-08",
      "feature": [
        "arbitrary-names",
        "pre-provisioning"
      ],
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-interfaces",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-ip",
      "revision": "2014-06-16",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-ip",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-network-instance",
      "revision": "2018-02-03",
      "feature": [
        "bind-network-instance-name"
      ],
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-network-instance",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-ospf",
      "revision": "2018-03-03",
      "namespace": "urn:ietf:params:xml:ns:yang:ietf-ospf",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-routing",
      "revision": "2018-03-13",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-routing",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-system",
      "revision": "2014-08-06",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-system",
      "conformance-type": "implement"
    }
  ],
  "namespace":
    "urn:ietf:params:xml:ns:yang:ietf-system",
  "conformance-type": "implement"
}
```

```

    },
    {
      "name": "ietf-yang-library",
      "revision": "2016-06-21",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-yang-library",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-yang-schema-mount",
      "revision": "2017-05-16",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-yang-types",
      "revision": "2013-07-15",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-yang-types",
      "conformance-type": "import"
    }
  ]
},
"ietf-yang-schema-mount:schema-mounts": {
  "mount-point": [
    {
      "module": "ietf-network-instance",
      "label": "vrf-root",
      "shared-schema": {
        "parent-reference": [
          "/*[namespace-uri() = 'urn:ietf:...:ietf-interfaces']"
        ]
      }
    }
  ]
}
}

```

B.3. State Data - NMDA Version

The following shows state data for the configuration example above based on [I-D.ietf-netmod-rfc7223bis], [I-D.ietf-netmod-rfc7277bis], and [I-D.ietf-netmod-rfc8022bis].

```

{
  "ietf-network-instance:network-instances": {

```

```
"network-instance": [
  {
    "name": "vrf-red",
    "vrf-root": {
      "ietf-yang-library:yang-library": {
        "checksum": "41e2ab5dc325f6d86f743e8da3de323f1a61a801",
        "module-set": [
          {
            "name": "ni-modules",
            "module": [
              {
                "name": "ietf-yang-library",
                "revision": "2016-06-21",
                "namespace":
                  "urn:ietf:params:xml:ns:yang:ietf-yang-library",
                "conformance-type": "implement"
              },
              {
                "name": "ietf-ospf",
                "revision": "2018-03-03",
                "namespace":
                  "urn:ietf:params:xml:ns:yang:ietf-ospf",
                "conformance-type": "implement"
              },
              {
                "name": "ietf-routing",
                "revision": "2018-03-13",
                "namespace":
                  "urn:ietf:params:xml:ns:yang:ietf-routing",
                "conformance-type": "implement"
              }
            ]
          },
          {
            "import-only-module": [
              {
                "name": "ietf-inet-types",
                "revision": "2013-07-15",
                "namespace": "urn:ietf:params:xml:ns:yang:ietf-inet-types"
              },
              {
                "name": "ietf-yang-types",
                "revision": "2013-07-15",
                "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-types"
              },
              {
                "name": "ietf-datastores",
                "revision": "2018-02-14",
                "namespace": "urn:ietf:params:xml:ns:yang:ietf-datastores"
              }
            ]
          }
        ]
      }
    }
  ]
}
```

```

    ]
  }
],
"schema": [
  {
    "name": "ni-schema",
    "module-set": [ "ni-modules" ]
  }
],
"datastore": [
  {
    "name": "ietf-datastores:running",
    "schema": "ni-schema"
  },
  {
    "name": "ietf-datastores:operational",
    "schema": "ni-schema"
  }
]
},
"ietf-routing:routing": {
  "router-id": "192.0.2.1",
  "control-plane-protocols": {
    "control-plane-protocol": [
      {
        "type": "ietf-routing:ospf",
        "name": "1",
        "ietf-ospf:ospf": {
          "af": "ipv4",
          "areas": {
            "area": [
              {
                "area-id": "203.0.113.1",
                "interfaces": {
                  "interface": [
                    {
                      "name": "eth1",
                      "cost": 10
                    }
                  ]
                }
              }
            ]
          }
        }
      }
    ]
  }
}

```

```
    }  
  },  
{  
  "name": "vrf-blue",  
  "vrf-root": {  
    "ietf-yang-library:yang-library": {  
      "checksum": "41e2ab5dc325f6d86f743e8da3de323f1a61a801",  
      "module-set": [  
        {  
          "name": "ni-modules",  
          "module": [  
            {  
              "name": "ietf-yang-library",  
              "revision": "2016-06-21",  
              "namespace":  
                "urn:ietf:params:xml:ns:yang:ietf-yang-library",  
              "conformance-type": "implement"  
            },  
            {  
              "name": "ietf-ospf",  
              "revision": "2018-03-03",  
              "namespace":  
                "urn:ietf:params:xml:ns:yang:ietf-ospf",  
              "conformance-type": "implement"  
            },  
            {  
              "name": "ietf-routing",  
              "revision": "2018-03-13",  
              "namespace":  
                "urn:ietf:params:xml:ns:yang:ietf-routing",  
              "conformance-type": "implement"  
            }  
          ],  
          "import-only-module": [  
            {  
              "name": "ietf-inet-types",  
              "revision": "2013-07-15",  
              "namespace": "urn:ietf:params:xml:ns:yang:ietf-inet-types"  
            },  
            {  
              "name": "ietf-yang-types",  
              "revision": "2013-07-15",  
              "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-types"  
            },  
            {  
              "name": "ietf-datastores",  
              "revision": "2018-02-14",  
            }  
          ]  
        }  
      ]  
    }  
  }  
}
```

```

        "namespace": "urn:ietf:params:xml:ns:yang:ietf-datastores"
      }
    ]
  },
  "schema": [
    {
      "name": "ni-schema",
      "module-set": [ "ni-modules" ]
    }
  ],
  "datastore": [
    {
      "name": "ietf-datastores:running",
      "schema": "ni-schema"
    },
    {
      "name": "ietf-datastores:operational",
      "schema": "ni-schema"
    }
  ]
},
"ietf-routing:routing": {
  "router-id": "192.0.2.2",
  "control-plane-protocols": {
    "control-plane-protocol": [
      {
        "type": "ietf-routing:ospf",
        "name": "1",
        "ietf-ospf:ospf": {
          "af": "ipv4",
          "areas": {
            "area": [
              {
                "area-id": "203.0.113.1",
                "interfaces": {
                  "interface": [
                    {
                      "name": "eth2",
                      "cost": 10
                    }
                  ]
                }
              }
            ]
          }
        }
      }
    ]
  }
}

```

```

    ]
  }
}
}
],
},
"ietf-interfaces:interfaces": {
  "interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "oper-status": "up",
        "phys-address": "00:01:02:A1:B1:C0",
        "statistics": {
          "discontinuity-time": "2017-06-26T12:34:56-05:00"
        },
        "ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.10",
              "prefix-length": 24,
            }
          ]
        },
        "ip:ipv6": {
          "address": [
            {
              "ip": "2001:db8:0:2::10",
              "prefix-length": 64,
            }
          ]
        }
      },
      {
        "name": "eth1",
        "type": "iana-if-type:ethernetCsmacd",
        "oper-status": "up",
        "phys-address": "00:01:02:A1:B1:C1",
        "statistics": {
          "discontinuity-time": "2017-06-26T12:34:56-05:00"
        },
        "ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.11",

```

```

        "prefix-length": 24,
      }
    ]
  }
  "ip:ipv6": {
    "address": [
      {
        "ip": "2001:db8:0:2::11",
        "prefix-length": 64,
      }
    ]
  }
},
{
  "name": "eth2",
  "type": "iana-if-type:ethernetCsmacd",
  "oper-status": "up",
  "phys-address": "00:01:02:A1:B1:C2",
  "statistics": {
    "discontinuity-time": "2017-06-26T12:34:56-05:00"
  },
  "ip:ipv4": {
    "address": [
      {
        "ip": "192.0.2.11",
        "prefix-length": 24,
      }
    ]
  }
  "ip:ipv6": {
    "address": [
      {
        "ip": "2001:db8:0:2::11",
        "prefix-length": 64,
      }
    ]
  }
}
]
},
{
  "ietf-system:system-state": {
    "platform": {
      "os-name": "NetworkOS"
    }
  }
}

```



```
"ietf-yang-library:modules-state": {
  "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
  "module": [
    {
      "name": "iana-if-type",
      "revision": "2014-05-08",
      "namespace":
        "urn:ietf:params:xml:ns:yang:iana-if-type",
      "conformance-type": "import"
    },
    {
      "name": "ietf-inet-types",
      "revision": "2013-07-15",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-inet-types",
      "conformance-type": "import"
    },
    {
      "name": "ietf-interfaces",
      "revision": "2018-01-09",
      "feature": [
        "arbitrary-names",
        "pre-provisioning"
      ],
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-interfaces",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-ip",
      "revision": "2018-01-09",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-ip",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-network-instance",
      "revision": "2018-02-03",
      "feature": [
        "bind-network-instance-name"
      ],
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-network-instance",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-ospf",
      "revision": "2017-10-30",
```

```
    "namespace": "urn:ietf:params:xml:ns:yang:ietf-ospf",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-routing",
    "revision": "2018-01-25",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-routing",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-system",
    "revision": "2014-08-06",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-system",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-yang-library",
    "revision": "2016-06-21",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-yang-library",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-yang-schema-mount",
    "revision": "2017-05-16",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-yang-types",
    "revision": "2013-07-15",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-yang-types",
    "conformance-type": "import"
  }
]
},
"ietf-yang-schema-mount:schema-mounts": {
  "mount-point": [
    {
      "module": "ietf-network-instance",
      "label": "vrf-root",
      "shared-schema": {
        "parent-reference": [
```

```
        "/*[namespace-uri() = 'urn:ietf:...:ietf-interfaces']"  
      ]  
    }  
  ]  
}
```

Authors' Addresses

Lou Berger
LabN Consulting, L.L.C.

Email: lberger@labn.net

Christan Hopps
Deutsche Telekom

Email: chopps@chopps.org

Acee Lindem
Cisco Systems
301 Midenhall Way
Cary, NC 27513
USA

Email: acee@cisco.com

Dean Bogdanovic

Email: ivandean@gmail.com

Xufeng Liu
Jabil

Email: Xufeng_Liu@jabil.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 16, 2018

X. Liu
Jabil
Y. Qu
Futurewei Technologies, Inc.
A. Lindem
Cisco Systems
C. Hopps
Deutsche Telekom
L. Berger
LabN Consulting, L.L.C.
October 13, 2017

Routing Area Common YANG Data Types
draft-ietf-rtgwg-routing-types-17

Abstract

This document defines a collection of common data types using the YANG data modeling language. These derived common types are designed to be imported by other modules defined in the routing area.

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 April 16, 2018.

Copyright Notice

Copyright (c) 2017 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. Terminology	2
2. Overview	2
3. IETF Routing Types YANG Module	6
4. IANA Routing Types YANG Module	22
5. IANA Considerations	31
5.1. IANA-Maintained iana-routing-types Module	32
6. Security Considerations	33
7. Acknowledgements	33
8. References	34
8.1. Normative References	34
8.2. Informative References	34
Authors' Addresses	36

1. Introduction

The YANG [RFC6020] [RFC7950] is a data modeling language used to model configuration data, state data, Remote Procedure Calls, and notifications for network management protocols. The YANG language supports a small set of built-in data types and provides mechanisms to derive other types from the built-in types.

This document introduces a collection of common data types derived from the built-in YANG data types. The derived types are designed to be the common types applicable for modeling in the routing area.

1.1. Terminology

The terminology for describing YANG data models is found in [RFC7950].

2. Overview

This document defines the two models for common routing types, ietf-routing-types and iana-routing-types. The only module imports are from [RFC6991]. The ietf-routing-types model contains common routing types other than those corresponding directly to IANA mappings. These include:

router-id

Router Identifiers are commonly used to identify nodes in routing and other control plane protocols. An example usage of router-id can be found in [I-D.ietf-ospf-yang].

route-target

Route Targets (RTs) are commonly used to control the distribution of virtual routing and forwarding (VRF) information, see [RFC4364], in support of BGP/MPLS IP virtual private networks (VPNs) and BGP/MPLS Ethernet VPNs [RFC7432]. An example usage can be found in [I-D.ietf-bess-l2vpn-yang].

ipv6-route-target

IPv6 Route Targets (RTs) are similar to standard Route Targets only they are IPv6 Address Specific BGP Extended Communities as described in [RFC5701]. An IPv6 Route Target is 20 octets and includes an IPv6 address as the global administrator.

route-target-type

This type defines the import and export rules of Route Targets, as described in Section 4.3.1 of [RFC4364]. An example usage can be found in [I-D.ietf-idr-bgp-model].

route-distinguisher

Route Distinguishers (RDs) are commonly used to identify separate routes in support of virtual private networks (VPNs). For example, in [RFC4364], RDs are commonly used to identify independent VPNs and VRFs, and more generally, to identify multiple routes to the same prefix. An example usage can be found in [I-D.ietf-idr-bgp-model].

route-origin

Route Origin is commonly used to indicate the Site of Origin for Routing and forwarding (VRF) information, see [RFC4364], in support of BGP/MPLS IP virtual private networks (VPNs) and BGP/MPLS Ethernet VPNs [RFC7432]. An example usage can be found in [I-D.ietf-bess-l3vpn-yang].

ipv6-route-origin

An IPv6 Route Origin would also be used to indicate the Site of Origin for Routing and forwarding (VRF) information, see [RFC4364], in support of virtual private networks (VPNs). IPv6 Route Origins are IPv6 Address Specific BGP Extended Communities as described in [RFC5701]. An IPv6 Route Origin is 20 octets and includes an IPv6 address as the global administrator.

ipv4-multicast-group-address

This type defines the representation of an IPv4 multicast group address, which is in the range from 224.0.0.0 to 239.255.255.255. An example usage can be found in [I-D.ietf-pim-yang].

ipv6-multicast-group-address

This type defines the representation of an IPv6 multicast group address, which is in the range of FF00::/8. An example usage can be found in [I-D.ietf-pim-yang].

ip-multicast-group-address

This type represents an IP multicast group address and is IP version neutral. The format of the textual representation implies the IP version. An example usage can be found in [I-D.ietf-pim-yang].

ipv4-multicast-source-address

IPv4 source address type for use in multicast control protocols. This type also allows the indication of wildcard sources, i.e., "*". An example of where this type may/will be used is [I-D.ietf-pim-yang].

ipv6-multicast-source-address

IPv6 source address type for use in multicast control protocols. This type also allows the indication of wildcard sources, i.e., "*". An example of where this type may/will be used is [I-D.ietf-pim-yang].

bandwidth-ieee-float32

Bandwidth in IEEE 754 floating point 32-bit binary format [IEEE754]. Commonly used in Traffic Engineering control plane protocols. An example of where this type may/will be used is [I-D.ietf-ospf-yang].

link-access-type

This type identifies the IGP link type. An example of where this type may/will be used is [I-D.ietf-ospf-yang].

timer-multiplier

This type is used in conjunction with a timer-value type. It is generally used to indicate define the number of timer-value intervals that may expire before a specific event must occur. Examples of this include the arrival of any BFD packets, see [RFC5880] Section 6.8.4, or hello_interval in [RFC3209]. Example of where this type may/will be used is [I-D.ietf-idr-bgp-model] and [I-D.ietf-teas-yang-rsvp].

timer-value-seconds16

This type covers timers which can be set in seconds, not set, or set to infinity. This type supports a range of values that can be represented in a uint16 (2 octets). An example of where this type may/will be used is [I-D.ietf-ospf-yang].

timer-value-seconds32

This type covers timers which can be set in seconds, not set, or set to infinity. This type supports a range of values that can be represented in a uint32 (4 octets). An example of where this type may/will be used is [I-D.ietf-teas-yang-rsvp].

timer-value-milliseconds

This type covers timers which can be set in milliseconds, not set, or set to infinity. This type supports a range of values that can be represented in a uint32 (4 octets). Examples of where this type may/will be used include [I-D.ietf-teas-yang-rsvp] and [I-D.ietf-bfd-yang].

percentage

This type defines a percentage with a range of 0-100%. An example usage can be found in [I-D.ietf-idr-bgp-model].

timeticks64

This type is based on the timeticks type defined in [RFC6991] but with 64-bit precision. It represents the time in hundredths of a second between two epochs. An example usage can be found in [I-D.ietf-idr-bgp-model].

uint24

This type defines a 24-bit unsigned integer. It is used by [I-D.ietf-ospf-yang].

generalized-label

This type represents a generalized label for Generalized Multi-Protocol Label Switching (GMPLS) [RFC3471]. The Generalized Label does not identify its type, which is known from the context. An example usage can be found in [I-D.ietf-teas-yang-tel].

mpls-label-special-purpose

This type represents the special-purpose Multiprotocol Label Switching (MPLS) label values [RFC7274]. An example usage can be found in [I-D.ietf-mpls-base-yang].

mpls-label-general-use

The 20 bits label values in an MPLS label stack entry, specified in [RFC3032]. This label value does not include the encodings of Traffic Class and TTL (time to live). The label range specified by this type is for general use, with special-purpose MPLS label

values excluded. An example usage can be found in [I-D.ietf-mpls-base-yang].

mpls-label

The 20 bits label values in an MPLS label stack entry, specified in [RFC3032]. This label value does not include the encodings of Traffic Class and TTL (time to live). The label range specified by this type covers the general use values and the special-purpose label values. An example usage can be found in [I-D.ietf-mpls-base-yang].

This document defines the following YANG groupings:

mpls-label-stack

This grouping defines a reusable collection of schema nodes representing an MPLS label stack [RFC3032]. An example usage can be found in [I-D.ietf-mpls-base-yang].

vpn-route-targets

This grouping defines a reusable collection of schema nodes representing Route Target import-export rules used in the BGP enabled Virtual Private Networks (VPNs). [RFC4364][RFC4664]. An example usage can be found in [I-D.ietf-bess-l2vpn-yang].

The iana-routing-types model contains common routing types corresponding directly to IANA mappings. These include:

address-family

This type defines values for use in address family identifiers. The values are based on the IANA Address Family Numbers Registry [IANA-ADDRESS-FAMILY-REGISTRY]. An example usage can be found in [I-D.ietf-idr-bgp-model].

subsequent-address-family

This type defines values for use in subsequent address family (SAFI) identifiers. The values are based on the IANA Subsequent Address Family Identifiers (SAFI) Parameters Registry [IANA-SAFI-REGISTRY].

3. IETF Routing Types YANG Module

```
<CODE BEGINS> file "ietf-routing-types@2017-10-13.yang"
module ietf-routing-types {
  namespace "urn:ietf:params:xml:ns:yang:ietf-routing-types";
  prefix rt-types;

  import ietf-yang-types {
    prefix yang;
```

```
}
import ietf-inet-types {
  prefix inet;
}

organization
  "IETF RTGWG - Routing Area Working Group";
contact
  "WG Web:    <http://tools.ietf.org/wg/rtgwg/>
  WG List:    <mailto:rtgwg@ietf.org>

  Editor:     Xufeng Liu
               <mailto:Xufeng_Liu@jabail.com>
               Yingzhen Qu
               <mailto:yingzhen.qu@huawei.com>
               Acee Lindem
               <mailto:acee@cisco.com>
               Christian Hopps
               <mailto:chopps@chopps.org>
               Lou Berger
               <mailto:lberger@labn.com>";
description
  "This module contains a collection of YANG data types
  considered generally useful for routing protocols.

  Copyright (c) 2017 IETF Trust and the persons
  identified as authors of the code.  All rights reserved.

  Redistribution and use in source and binary forms, with or
  without modification, is permitted pursuant to, and subject
  to the license terms contained in, the Simplified BSD License
  set forth in Section 4.c of the IETF Trust's Legal Provisions
  Relating to IETF Documents
  (http://trustee.ietf.org/license-info).

  This version of this YANG module is part of RFC XXXX; see
  the RFC itself for full legal notices.";
reference "RFC XXXX";

revision 2017-10-13 {
  description "Initial revision.";
  reference "RFC TBD: Routing YANG Data Types";
}

/*** Identities related to MPLS/GMPLS ***/

identity mpls-label-special-purpose-value {
  description
```

```
    "Base identity for deriving identities describing
    special-purpose Multiprotocol Label Switching (MPLS) label
    values.";
  reference
    "RFC7274: Allocating and Retiring Special-Purpose MPLS
    Labels.";
}

identity ipv4-explicit-null-label {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the IPv4 Explicit NULL Label.";
  reference "RFC3032: MPLS Label Stack Encoding. Section 2.1.";
}

identity router-alert-label {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the Router Alert Label.";
  reference "RFC3032: MPLS Label Stack Encoding. Section 2.1.";
}

identity ipv6-explicit-null-label {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the IPv6 Explicit NULL Label.";
  reference "RFC3032: MPLS Label Stack Encoding. Section 2.1.";
}

identity implicit-null-label {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the Implicit NULL Label.";
  reference "RFC3032: MPLS Label Stack Encoding. Section 2.1.";
}

identity entropy-label-indicator {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the Entropy Label Indicator.";
  reference
    "RFC6790: The Use of Entropy Labels in MPLS Forwarding.
    Sections 3 and 10.1.";
}

identity gal-label {
  base mpls-label-special-purpose-value;
  description
```

```
    "This identity represents the Generic Associated Channel
    Label (GAL).";
  reference
    "RFC5586: MPLS Generic Associated Channel.
    Sections 4 and 10.";
}

identity oam-alert-label {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the OAM Alert Label.";
  reference
    "RFC3429: Assignment of the 'OAM Alert Label' for
    Multiprotocol Label Switching Architecture (MPLS)
    Operation and Maintenance (OAM) Functions.
    Sections 3 and 6.";
}

identity extension-label {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the Extension Label.";
  reference
    "RFC7274: Allocating and Retiring Special-Purpose MPLS
    Labels. Sections 3.1 and 5.";
}

/*** Collection of types related to routing ***/

typedef router-id {
  type yang:dotted-quad;
  description
    "A 32-bit number in the dotted quad format assigned to each
    router. This number uniquely identifies the router within
    an Autonomous System.";
}

/*** Collection of types related to VPN ***/

typedef route-target {
  type string {
    pattern
      '(0:(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2})|'
      + '6[0-4][0-9]{3})|'
      + '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0):(429496729[0-5]|'
      + '42949672[0-8][0-9]|'
      + '4294967[01][0-9]{2}|429496[0-6][0-9]{3})|'
      + '42949[0-5][0-9]{4})|'
```

```

+      '4294[0-8][0-9]{5}|429[0-3][0-9]{6}|'
+      '42[0-8][0-9]{7}|4[01][0-9]{8}|'
+      '[1-3][0-9]{9}|[1-9][0-9]{0,8}|0))|'
+      '(1:(((0-9)|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|'
+      '25[0-5])\.){3}([0-9]|[1-9][0-9]|'
+      '1[0-9]{2}|2[0-4][0-9]|25[0-5])):(6553[0-5]|'
+      '655[0-2][0-9]|'
+      '65[0-4][0-9]{2}|6[0-4][0-9]{3}|'
+      '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0))|'
+      '(2:(429496729[0-5]|42949672[0-8][0-9]|'
+      '4294967[01][0-9]{2}|'
+      '429496[0-6][0-9]{3}|42949[0-5][0-9]{4}|'
+      '4294[0-8][0-9]{5}|'
+      '429[0-3][0-9]{6}|42[0-8][0-9]{7}|4[01][0-9]{8}|'
+      '[1-3][0-9]{9}|[1-9][0-9]{0,8}|0):'
+      '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|'
+      '6[0-4][0-9]{3}|'
+      '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0))|'
+      '(6:[a-fA-F0-9]{2}){6})|'
+      '(((3-57-9a-fA-F)|[1-9a-fA-F][0-9a-fA-F]{1,3}):'
+      '[0-9a-fA-F]{1,12})' ;
}

```

description

"A route target is an 8-octet BGP extended community initially identifying a set of sites in a BGP VPN (RFC 4364). However, it has since taken on a more general role in BGP route filtering.

A route target consists of two or three fields: a 2-octet type field, an administrator field, and, optionally, an assigned number field.

According to the data formats for type 0, 1, 2, and 6 defined in RFC4360, RFC5668, and RFC7432, the encoding pattern is defined as:

```

0:2-octet-asn:4-octet-number
1:4-octet-ipv4addr:2-octet-number
2:4-octet-asn:2-octet-number.
6:6-octet-mac-address.

```

Additionally, a generic pattern is defined for future route target types:

```
2-octet-other-hex-number:6-octet-hex-number
```

Some valid examples are: 0:100:100, 1:1.1.1.1:100, 2:1234567890:203 and 6:26:00:08:92:78:00";

reference

```

    "RFC4360: BGP Extended Communities Attribute.
    RFC4364: BGP/MPLS IP Virtual Private Networks (VPNs)
    RFC5668: 4-Octet AS Specific BGP Extended Community.
    RFC7432: BGP MPLS-Based Ethernet VPN";
}

typedef ipv6-route-target {
  type string {
    pattern
      '(:|0-9a-fA-F){0,4}:([0-9a-fA-F]{0,4}:){0,5}'
      + '(((0-9a-fA-F){0,4}:)?(:|0-9a-fA-F){0,4})|'
      + '(((25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])\.){3}'
      + '(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9]))'
      + ':'
      + '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|'
      + '6[0-4][0-9]{3}|'
      + '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0)';
    pattern '(((^[^:]+:){6}([^[^:]+:[^:]+)|(.*\..*)))|'
      + '(((^[^:]+:)*[^[^:]+)?::([^[^:]+)*[^[^:]+]?))'
      + ':'
      + '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|'
      + '6[0-4][0-9]{3}|'
      + '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0)';
  }
  description
    "An IPv6 route target is a 20-octet BGP IPv6 address
    specific extended community serving the same function
    as a standard 8-octet route target only allowing for
    an IPv6 address as the global administrator. The format
    is <ipv6-address:2-octet-number>.

    Some valid examples are: 2001:DB8::1:6544 and
    2001:DB8::5eb1:791:6b37:17958";
  reference
    "RFC5701: IPv6 Address Specific BGP Extended Community
    Attribute";
}

typedef route-target-type {
  type enumeration {
    enum "import" {
      value 0;
      description
        "The route target applies to route import.";
    }
    enum "export" {
      value 1;
      description

```

```

        "The route target applies to route export.";
    }
    enum "both" {
        value 2;
        description
            "The route target applies to both route import and
            route export.";
    }
}
description
    "Indicates the role a route target takes
    in route filtering.";
reference "RFC4364: BGP/MPLS IP Virtual Private Networks
(VPNs).";
}

typedef route-distinguisher {
    type string {
        pattern
            '(0:(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2})|'
        +   '6[0-4][0-9]{3})|'
        +   '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0):(429496729[0-5]|'
        +   '42949672[0-8][0-9]|'
        +   '4294967[01][0-9]{2}|429496[0-6][0-9]{3})|'
        +   '42949[0-5][0-9]{4})|'
        +   '4294[0-8][0-9]{5}|429[0-3][0-9]{6})|'
        +   '42[0-8][0-9]{7}|4[01][0-9]{8})|'
        +   '[1-3][0-9]{9}|[1-9][0-9]{0,8}|0))|'
        +   '(1:(((0-9)|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9])|'
        +   '25[0-5])\.){3}([0-9]|[1-9][0-9]|'
        +   '1[0-9]{2}|2[0-4][0-9]|25[0-5])):(6553[0-5]|'
        +   '655[0-2][0-9]|'
        +   '65[0-4][0-9]{2}|6[0-4][0-9]{3})|'
        +   '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0))|'
        +   '(2:(429496729[0-5]|42949672[0-8][0-9]|'
        +   '4294967[01][0-9]{2}|'
        +   '429496[0-6][0-9]{3}|42949[0-5][0-9]{4})|'
        +   '4294[0-8][0-9]{5}|'
        +   '429[0-3][0-9]{6}|42[0-8][0-9]{7}|4[01][0-9]{8})|'
        +   '[1-3][0-9]{9}|[1-9][0-9]{0,8}|0):'
        +   '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2})|'
        +   '6[0-4][0-9]{3})|'
        +   '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0))|'
        +   '(6(:[a-fA-F0-9]{2}){6})|'
        +   '([3-57-9a-fA-F]|1-9a-fA-F)[0-9a-fA-F]{1,3}):'
        +   '[0-9a-fA-F]{1,12})';
    }
}
description

```

"A route distinguisher is an 8-octet value used to distinguish routes from different BGP VPNs (RFC 4364). As per RFC 4360, a route distinguisher will have the same format as a route target and will consist of two or three fields including a 2-octet type field, an administrator field, and, optionally, an assigned number field.

According to the data formats for type 0, 1, 2, and 6 defined in RFC4360, RFC5668, and RFC7432, the encoding pattern is defined as:

```
0:2-octet-asn:4-octet-number
1:4-octet-ipv4addr:2-octet-number
2:4-octet-asn:2-octet-number.
6:6-octet-mac-address.
```

Additionally, a generic pattern is defined for future route discriminator types:

```
2-octet-other-hex-number:6-octet-hex-number
```

Some valid examples are: 0:100:100, 1:1.1.1.1:100, 2:1234567890:203 and 6:26:00:08:92:78:00";

reference

```
"RFC4360: BGP Extended Communities Attribute.
RFC4364: BGP/MPLS IP Virtual Private Networks (VPNs)
RFC5668: 4-Octet AS Specific BGP Extended Community.
RFC7432: BGP MPLS-Based Ethernet VPN";
```

}

```
typedef route-origin {
  type string {
    pattern
      '(0:(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]){2}|'
    +   '6[0-4][0-9]{3}|'
    +   '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0):(429496729[0-5]|'
    +   '42949672[0-8][0-9]|'
    +   '4294967[01][0-9]{2}|429496[0-6][0-9]{3}|'
    +   '42949[0-5][0-9]{4}|'
    +   '4294[0-8][0-9]{5}|429[0-3][0-9]{6}|'
    +   '42[0-8][0-9]{7}|4[01][0-9]{8}|'
    +   '[1-3][0-9]{9}|[1-9][0-9]{0,8}|0))|'
    +   '(1:(((0-9)|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|'
    +   '25[0-5])\.)}{3}([0-9]|[1-9][0-9]|'
    +   '1[0-9]{2}|2[0-4][0-9]|25[0-5])):(6553[0-5]|'
    +   '655[0-2][0-9]|'
    +   '65[0-4][0-9]{2}|6[0-4][0-9]{3}|'
    +   '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0))|'
```



```

+ '(2:(429496729[0-5]|42949672[0-8][0-9]|'
+   '4294967[01][0-9]{2}|'
+   '429496[0-6][0-9]{3}|42949[0-5][0-9]{4}|'
+   '4294[0-8][0-9]{5}|'
+   '429[0-3][0-9]{6}|42[0-8][0-9]{7}|4[01][0-9]{8}|'
+   '[1-3][0-9]{9}|[1-9][0-9]{0,8}|0):'
+   '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|'
+   '6[0-4][0-9]{3}|'
+   '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0))|'
+ '(6(:[a-fA-F0-9]{2}){6})|'
+ '(((3-57-9a-fA-F)|[1-9a-fA-F][0-9a-fA-F]{1,3}):'
+   '[0-9a-fA-F]{1,12}))';

```

}
description

"A route origin is an 8-octet BGP extended community identifying the set of sites where the BGP route originated (RFC 4364). A route target consists of two or three fields: a 2-octet type field, an administrator field, and, optionally, an assigned number field.

According to the data formats for type 0, 1, 2, and 6 defined in RFC4360, RFC5668, and RFC7432, the encoding pattern is defined as:

```

0:2-octet-asn:4-octet-number
1:4-octet-ipv4addr:2-octet-number
2:4-octet-asn:2-octet-number.
6:6-octet-mac-address.

```

Additionally, a generic pattern is defined for future route origin types:

```
2-octet-other-hex-number:6-octet-hex-number
```

Some valid examples are: 0:100:100, 1:1.1.1.1:100, 2:1234567890:203 and 6:26:00:08:92:78:00";

reference

```

"RFC4360: BGP Extended Communities Attribute.
RFC4364: BGP/MPLS IP Virtual Private Networks (VPNs)
RFC5668: 4-Octet AS Specific BGP Extended Community.
RFC7432: BGP MPLS-Based Ethernet VPN";

```

}

```

typedef ipv6-route-origin {
  type string {
    pattern
      '(((:[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}):){0,5}'
      + '(((([0-9a-fA-F]{0,4}):)?(:[0-9a-fA-F]{0,4}))|'

```

```

+ '((25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])\.){3}'
+ '(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9]))'
+ ':'
+ '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|'
+ '6[0-4][0-9]{3}|'
+ '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0)';
pattern '(((^[^:]+:){6}([^[^:]+:[^:]+)|(.*\..*)))|'
+ '(((^[^:]+:)*[^[^:]+)?::([^[^:]+:[^:]+)?))'
+ ':'
+ '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|'
+ '6[0-4][0-9]{3}|'
+ '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0)';
}
description
  "An IPv6 route origin is a 20-octet BGP IPv6 address
  specific extended community serving the same function
  as a standard 8-octet route only allowing for
  an IPv6 address as the global administrator. The format
  is <ipv6-address:2-octet-number>.

  Some valid examples are: 2001:DB8::1:6544 and
  2001:DB8::5eb1:791:6b37:17958";
reference
  "RFC5701: IPv6 Address Specific BGP Extended Community
  Attribute";
}

/*** Collection of types common to multicast ***/

typedef ipv4-multicast-group-address {
  type inet:ipv4-address {
    pattern '(2((2[4-9])|(3[0-9]))\.)\.*';
  }
  description
    "This type represents an IPv4 multicast group address,
    which is in the range from 224.0.0.0 to 239.255.255.255.";
  reference "RFC1112: Host Extensions for IP Multicasting.";
}

typedef ipv6-multicast-group-address {
  type inet:ipv6-address {
    pattern '([fF]{2}[0-9a-fA-F]{2})\.*';
  }
  description
    "This type represents an IPv6 multicast group address,
    which is in the range of FF00::/8.";
  reference

```

```
    "RFC4291: IP Version 6 Addressing Architecture. Sec 2.7.
    RFC7346: IPv6 Multicast Address Scopes.";
}

typedef ip-multicast-group-address {
    type union {
        type ipv4-multicast-group-address;
        type ipv6-multicast-group-address;
    }
    description
        "This type represents a version-neutral IP multicast group
        address. The format of the textual representation implies
        the IP version.";
}

typedef ipv4-multicast-source-address {
    type union {
        type enumeration {
            enum "*" {
                description
                    "Any source address.";
            }
        }
        type inet:ipv4-address;
    }
    description
        "Multicast source IPv4 address type.";
}

typedef ipv6-multicast-source-address {
    type union {
        type enumeration {
            enum "*" {
                description
                    "Any source address.";
            }
        }
        type inet:ipv6-address;
    }
    description
        "Multicast source IPv6 address type.";
}

/**/ Collection of types common to protocols ***/

typedef bandwidth-ieee-float32 {
    type string {
        pattern
```

```

    '0[xX](0((\.0?)?[pP](\+)?0?|(\.0?))|'
+ '1(\.([0-9a-fA-F]{0,5}[02468aAcCeE]?)?)?[pP](\+)?(12[0-7]|'
+ '1[01][0-9]|0?[0-9]?[0-9])?)';
}
description
  "Bandwidth in IEEE 754 floating point 32-bit binary format:
   $(-1)^{(S)} * 2^{(Exponent-127)} * (1 + Fraction)$ ,
  where Exponent uses 8 bits, and Fraction uses 23 bits.
  The units are octets per second.
  The encoding format is the external hexadecimal-significant
  character sequences specified in IEEE 754 and C99. The
  format is restricted to be normalized, non-negative, and
  non-fraction: 0x1.hhhhhhp{+}d, 0X1.HHHHHHP{+}D, or 0x0p0,
  where 'h' and 'H' are hexadecimal digits and 'd' and 'D' are
  integers in the range of [0..127].
  When six hexadecimal digits are used for 'hhhhh' or
  'HHHHH', the least significant digit must be an even
  number. 'x' and 'X' indicate hexadecimal; 'p' and 'P'
  indicate power of two. Some examples are: 0x0p0, 0x1p10, and
  0x1.abcde2p+20";
reference
  "IEEE Std 754-2008: IEEE Standard for Floating-Point
  Arithmetic.";
}

typedef link-access-type {
  type enumeration {
    enum "broadcast" {
      description
        "Specify broadcast multi-access network.";
    }
    enum "non-broadcast-multiaccess" {
      description
        "Specify Non-Broadcast Multi-Access (NBMA) network.";
    }
    enum "point-to-multipoint" {
      description
        "Specify point-to-multipoint network.";
    }
    enum "point-to-point" {
      description
        "Specify point-to-point network.";
    }
  }
  description
    "Link access type.";
}

```

```
typedef timer-multiplier {
  type uint8;
  description
    "The number of timer value intervals that should be
    interpreted as a failure.";
}

typedef timer-value-seconds16 {
  type union {
    type uint16 {
      range "1..65535";
    }
    type enumeration {
      enum "infinity" {
        description
          "The timer is set to infinity.";
      }
      enum "not-set" {
        description
          "The timer is not set.";
      }
    }
  }
  units "seconds";
  description
    "Timer value type, in seconds (16-bit range).";
}

typedef timer-value-seconds32 {
  type union {
    type uint32 {
      range "1..4294967295";
    }
    type enumeration {
      enum "infinity" {
        description
          "The timer is set to infinity.";
      }
      enum "not-set" {
        description
          "The timer is not set.";
      }
    }
  }
  units "seconds";
  description
    "Timer value type, in seconds (32-bit range).";
}
```

```
typedef timer-value-milliseconds {
  type union {
    type uint32 {
      range "1..4294967295";
    }
    type enumeration {
      enum "infinity" {
        description
          "The timer is set to infinity.";
      }
      enum "not-set" {
        description
          "The timer is not set.";
      }
    }
  }
  units "milliseconds";
  description
    "Timer value type, in milliseconds.";
}

typedef percentage {
  type uint8 {
    range "0..100";
  }
  description
    "Integer indicating a percentage value";
}

typedef timeticks64 {
  type uint64;
  description
    "This type is based on the timeticks type defined in
    RFC 6991, but with 64-bit width. It represents the time,
    modulo 2^64, in hundredths of a second between two epochs.";
  reference "RFC 6991 - Common YANG Data Types";
}

typedef uint24 {
  type uint32 {
    range "0 .. 16777215";
  }
  description
    "24-bit unsigned integer";
}

/*** Collection of types related to MPLS/GMPLS ***/
```

```
typedef generalized-label {
  type binary;
  description
    "Generalized label. Nodes sending and receiving the
    Generalized Label are aware of the link-specific
    label context and type.";
  reference "RFC3471: Section 3.2";
}

typedef mpls-label-special-purpose {
  type identityref {
    base mpls-label-special-purpose-value;
  }
  description
    "This type represents the special-purpose Multiprotocol Label
    Switching (MPLS) label values.";
  reference
    "RFC3032: MPLS Label Stack Encoding.
    RFC7274: Allocating and Retiring Special-Purpose MPLS
    Labels.";
}

typedef mpls-label-general-use {
  type uint32 {
    range "16..1048575";
  }
  description
    "The 20-bit label values in an MPLS label stack entry,
    specified in RFC3032. This label value does not include
    the encodings of Traffic Class and TTL (time to live).
    The label range specified by this type is for general use,
    with special-purpose MPLS label values excluded.";
  reference "RFC3032: MPLS Label Stack Encoding.";
}

typedef mpls-label {
  type union {
    type mpls-label-special-purpose;
    type mpls-label-general-use;
  }
  description
    "The 20-bit label values in an MPLS label stack entry,
    specified in RFC3032. This label value does not include
    the encodings of Traffic Class and TTL (time to live).";
  reference "RFC3032: MPLS Label Stack Encoding.";
}

/*** Groupings **/
```

```
grouping mpls-label-stack {
  description
    "This grouping specifies an MPLS label stack. The label
    stack is encoded as a list of label stack entries. The
    list key is an identifier which indicates relative
    ordering of each entry, with the lowest value identifier
    corresponding to the top of the label stack.";
  container mpls-label-stack {
    description
      "Container for a list of MPLS label stack entries.";
    list entry {
      key "id";
      description
        "List of MPLS label stack entries.";
      leaf id {
        type uint8;
        description
          "Identifies the entry in a sequence of MPLS label
          stack entries. An entry with a smaller identifier
          value precedes an entry with a larger identifier
          value in the label stack. The value of this ID has
          no semantic meaning other than relative ordering
          and referencing the entry.";
      }
      leaf label {
        type rt-types:mpls-label;
        description
          "Label value.";
      }
      leaf ttl {
        type uint8;
        description
          "Time to Live (TTL).";
        reference "RFC3032: MPLS Label Stack Encoding.";
      }
      leaf traffic-class {
        type uint8 {
          range "0..7";
        }
        description
          "Traffic Class (TC).";
        reference
          "RFC5462: Multiprotocol Label Switching (MPLS) Label
          Stack Entry: 'EXP' Field Renamed to 'Traffic Class'
          Field.";
      }
    }
  }
}
```



```

    }

    grouping vpn-route-targets {
      description
        "A grouping that specifies Route Target import-export rules
        used in the BGP enabled Virtual Private Networks (VPNs).";
      reference
        "RFC4364: BGP/MPLS IP Virtual Private Networks (VPNs).
        RFC4664: Framework for Layer 2 Virtual Private Networks
        (L2VPNs)";
      list vpn-target {
        key "route-target";
        description
          "List of Route Targets.";
        leaf route-target {
          type rt-types:route-target;
          description
            "Route Target value";
        }
        leaf route-target-type {
          type rt-types:route-target-type;
          mandatory true;
          description
            "Import/export type of the Route Target.";
        }
      }
    }
  }
}

```

<CODE ENDS>

4. IANA Routing Types YANG Module

```

<CODE BEGINS> file "iana-routing-types@2017-09-19.yang"
module iana-routing-types {
  namespace "urn:ietf:params:xml:ns:yang:iana-routing-types";
  prefix iana-rt-types;

  organization
    "IANA";
  contact
    "
      Internet Assigned Numbers Authority

      Postal: ICANN
      4676 Admiralty Way, Suite 330
      Marina del Rey, CA 90292

      Tel: +1 310 823 9358
    "

```

```
<mailto:iana@iana.org>";
description
  "This module contains a collection of YANG data types
  considered defined by IANA and used for routing
  protocols.

  Copyright (c) 2017 IETF Trust and the persons
  identified as authors of the code.  All rights reserved.

  Redistribution and use in source and binary forms, with or
  without modification, is permitted pursuant to, and subject
  to the license terms contained in, the Simplified BSD License
  set forth in Section 4.c of the IETF Trust's Legal Provisions
  Relating to IETF Documents
  (http://trustee.ietf.org/license-info).

  This version of this YANG module is part of RFC XXXX; see
  the RFC itself for full legal notices.";
reference "RFC XXXX";

revision 2017-09-19 {
  description "Initial revision.";
  reference "RFC TBD: IANA Routing YANG Data Types";
}

/** Collection of IANA types related to routing */
/** IANA address family enumeration */

typedef address-family {
  type enumeration {
    enum ipv4 {
      value 1;
      description "IPv4 Address Family";
    }

    enum ipv6 {
      value 2;
      description "IPv6 Address Family";
    }

    enum nsap {
      value 3;
      description "OSI Network Service Access Point (NSAP)
                  Address Family";
    }

    enum hdlc {
      value 4;
    }
  }
}
```

```
        description "High-Level Data Link Control (HDLC)
                    Address Family";
    }

    enum bbn1822 {
        value 5;
        description "Bolt, Beranek, and Newman Report
                    1822 (BBN 1822) Address Family";
    }

    enum ieee802 {
        value 6;
        description "IEEE 802 Committee Address Family (aka,
                    MAC address)";
    }

    enum e163 {
        value 7;
        description "ITU-T E.163 Address Family";
    }

    enum e164 {
        value 8;
        description "ITU-T E.164 (SMDS, Frame Relay, ATM)
                    Address Family";
    }

    enum f69 {
        value 9;
        description "ITU-T F.69 (Telex) Address Family";
    }

    enum x121 {
        value 10;
        description "ITU-T X.121 (X.25, Frame Relay)
                    Address Family";
    }

    enum ipx {
        value 11;
        description "Novell Internetwork Packet Exchange (IPX)
                    Address Family";
    }

    enum appletalk {
        value 12;
        description "Apple AppleTalk Address Family";
    }
```

```
    }

    enum decnet-iv {
        value 13;
        description "Digital Equipment DECnet Phase IV
                    Address Family";
    }

    enum vines {
        value 14;
        description "Banyan Vines Address Family";
    }

    enum el64-nsap {
        value 15;
        description "ITU-T E.164 with NSAP sub-address
                    Address Family";
    }

    enum dns {
        value 16;
        description "Domain Name System (DNS) Address
                    Family";
    }

    enum distinguished-name {
        value 17;
        description "Distinguished Name Address Family";
    }

    enum as-num {
        value 18;
        description "AS Number Address Family";
    }

    enum xtp-v4 {
        value 19;
        description "Xpress Transport Protocol (XTP) over IPv4
                    Address Family";
    }

    enum xtp-v6 {
        value 20;
        description "Xpress Transport Protocol (XTP) over IPv6
                    Address Family";
    }

    enum xtp-native {
```

```
    value 21;
    description "Xpress Transport Protocol (XTP) native mode
                Address Family";
}

enum fc-port {
    value 22;
    description "Fibre Channel (FC) World-Wide Port Name
                Address Family";
}

enum fc-node {
    value 23;
    description "Fibre Channel (FC) World-Wide Node Name
                Address Family";
}

enum gwid {
    value 24;
    description "ATM Gateway Identifier (GWID) Number Address Family";
}

enum l2vpn {
    value 25;
    description "Layer-2 VPN (L2VPN) Address Family";
}

enum mpls-tp-section-eid {
    value 26;
    description "MPLS-TP Section Endpoint Identifier
                Address Family";
}

enum mpls-tp-lsp-eid {
    value 27;
    description "MPLS-TP LSP Endpoint Identifier
                Address Family";
}

enum mpls-tp-pwe-eid {
    value 28;
    description "MPLS-TP Pseudowire Endpoint Identifier
                Address Family";
}

enum mt-v4 {
    value 29;
```

```
    description "Multi-Topology IPv4 Address Family";
  }

  enum mt-v6 {
    value 30;
    description "Multi-Topology IPv6 Address Family";
  }

  enum eigrp-common-sf {
    value 16384;
    description "Enhanced Interior Gateway Routing Protocol
                 (EIGRP) Common Service Family Address
                 Family";
  }

  enum eigrp-v4-sf {
    value 16385;
    description "Enhanced Interior Gateway Routing Protocol
                 (EIGRP) IPv4 Service Family Address Family";
  }

  enum eigrp-v6-sf {
    value 16386;
    description "Enhanced Interior Gateway Routing Protocol
                 (EIGRP) IPv6 Service Family Address Family";
  }

  enum lcaf {
    value 16387;
    description "LISP Canonical Address Format (LCAF)
                 Address Family";
  }

  enum bgp-ls {
    value 16388;
    description "Border Gateway Protocol - Link State (BGP-LS)
                 Address Family";
  }

  enum mac-48 {
    value 16389;
    description "IEEE 48-bit Media Access Control (MAC)
                 Address Family";
  }

  enum mac-64 {
    value 16390;
    description "IEEE 64-bit Media Access Control (MAC)";
  }
```

```
        Address Family";
    }

    enum trill-oui {
        value 16391;
        description "TRILL IEEE Organizationally Unique
                    Identifier (OUI) Address Family";
    }

    enum trill-mac-24 {
        value 16392;
        description "TRILL Final 3 octets of 48-bit MAC
                    address Address Family";
    }

    enum trill-mac-40 {
        value 16393;
        description "TRILL Final 5 octets of 64-bit MAC
                    address Address Family";
    }

    enum ipv6-64 {
        value 16394;
        description "First 8 octets (64-bits) of an IPv6
                    address Address Family";
    }

    enum trill-rbridge-port-id {
        value 16395;
        description "TRILL Remote Bridge (RBridge) Port ID
                    Address Family";
    }

    enum trill-nickname {
        value 16396;
        description "TRILL Nickname Address Family";
    }
}
description "Enumeration containing all the IANA
            defined address families.";

}

/**** SAFIs for Multi-Protocol BGP enumeration ****/

typedef bgp-safi {
    type enumeration {
        enum unicast-safi {
```

```
    value 1;
    description "Unicast SAFI";
}

enum multicast-safi {
    value 2;
    description "Multicast SAFI";
}

enum labeled-unicast-safi {
    value 4;
    description "Labeled Unicast SAFI";
}

enum multicast-vpn-safi {
    value 5;
    description "Multicast VPN SAFI";
}

enum pseudowire-safi {
    value 6;
    description "Multi-segment Pseudowire VPN SAFI";
}

enum tunnel-encap-safi {
    value 7;
    description "Tunnel Encap SAFI";
}

enum mcast-vpls-safi {
    value 8;
    description "Multicast Virtual Private LAN Service
                  (VPLS) SAFI";
}

enum tunnel-safi {
    value 64;
    description "Tunnel SAFI";
}

enum vpls-safi {
    value 65;
    description "Virtual Private LAN Service (VPLS) SAFI";
}

enum mdt-safi {
    value 66;
    description "Multicast Distribution Tree (MDT) SAFI";
}
```



```
    }

    enum v4-over-v6-safi {
        value 67;
        description "IPv4 over IPv6 SAFI";
    }

    enum v6-over-v4-safi {
        value 68;
        description "IPv6 over IPv4 SAFI";
    }

    enum l1-vpn-auto-discovery-safi {
        value 69;
        description "Layer-1 VPN Auto Discovery SAFI";
    }

    enum evpn-safi {
        value 70;
        description "Ethernet VPN (EVPN) SAFI";
    }

    enum bgp-ls-safi {
        value 71;
        description "BGP Link-State (BGP-LS) SAFI";
    }

    enum bgp-ls-vpn-safi {
        value 72;
        description "BGP Link-State (BGP-LS) VPN SAFI";
    }

    enum sr-te-safi {
        value 73;
        description "Segment Routing - Traffic Engineering
                    (SR-TE) SAFI";
    }

    enum labeled-vpn-safi {
        value 128;
        description "MPLS Labeled VPN SAFI";
    }

    enum multicast-mpls-vpn-safi {
        value 129;
        description "Multicast for BGP/MPLS IP VPN SAFI";
    }
```

```
enum route-target-safi {
  value 132;
  description "Route Target SAFI";
}

enum ipv4-flow-spec-safi {
  value 133;
  description "IPv4 Flow Specification SAFI";
}

enum vpnv4-flow-spec-safi {
  value 134;
  description "IPv4 VPN Flow Specification SAFI";
}

enum vpn-auto-discovery-safi {
  value 140;
  description "VPN Auto-Discovery SAFI";
}
}
description "Enumeration for BGP Subsequent Address
             Family Identifier (SAFI) - RFC 4760."
}
}

<CODE ENDS>
```

5. IANA Considerations

RFC Ed.: In this section, replace all occurrences of 'XXXX' with the actual RFC number (and remove this note).

This document registers the following namespace URIs in the IETF XML registry [RFC3688]:

```
-----
URI: urn:ietf:params:xml:ns:yang:ietf-routing-types
Registrant Contact: The IESG.
XML: N/A, the requested URI is an XML namespace.
-----
```

```
-----
URI: urn:ietf:params:xml:ns:yang:iana-routing-types
Registrant Contact: IANA
XML: N/A, the requested URI is an XML namespace.
-----
```

This document registers the following YANG modules in the YANG Module Names registry [RFC6020]:

```
-----
name:      ietf-routing-types
namespace: urn:ietf:params:xml:ns:yang:ietf-routing-types
prefix:    rt-types
reference:  RFC XXXX
-----
```

```
-----
name:      iana-routing-types
namespace: urn:ietf:params:xml:ns:yang:iana-routing-types
prefix:    iana-rt-types
reference:  RFC XXXX
-----
```

5.1. IANA-Maintained iana-routing-types Module

This document defines the initial version of the IANA-maintained iana-routing-types YANG module Section 4.

The iana-routing-types YANG module is intended to reflect the "Address Family Numbers" registry [IANA-ADDRESS-FAMILY-REGISTRY] and "Subsequent Address Family Identifiers (SAFI) Parameters" registry [IANA-SAFI-REGISTRY].

IANA has added this notes to the "iana-routing-types YANG Module" registry:

Address Families and Subsequent Address Families must not be directly added to the iana-routing-types YANG module. They must instead be respectively added to the "Address Family Numbers" and "Subsequent Address Family Identifiers (SAFI) Parameters" registries.

When an Address Family or Subsequent Address Family is respectively added to the "Address Family Numbers" registry or the "Subsequent Address Family Identifiers (SAFI) Parameters" registry, a new "enum" statement must be added to the iana-routing-types YANG module. The name of the "enum" is the same as the corresponding address family or SAFI only it will be a valid YANG identifier in all lowercase and with hyphens separating individual words in compound identifiers. The following substatements to the "enum" statement should be defined:

"enum": Contains the YANG enum identifier for the address-family or "bgp-safi" for subsequent address families. This may be the same as the address-family or "bgp-safi" or it may be a shorter version to facilitate YANG identifier usage.

"value": Contains the IANA assigned value corresponding to the address-family or "bgp-safi" for subsequent address families.

"status": Include only if a registration has been deprecated (use the value "deprecated") or obsoleted (use the value "obsolete").

"description": Replicate the description from the registry, if any. Insert line breaks as needed so that the line does not exceed 72 characters.

"reference": Replicate the reference from the registry, if any, and add the title of the document.

Unassigned or reserved values are not present in these modules.

When the iana-routing-types YANG module is updated, a new "revision" statement must be added in front of the existing revision statements.

IANA has added this new note to the "Address Family Numbers" and "Subsequent Address Family Identifiers (SAFI) Parameters" registries:

When this registry is modified, the YANG module iana-routing-types must be updated as defined in RFC XXXX.

6. Security Considerations

This document defines common routing type definitions (i.e., typedef statements) using the YANG data modeling language. The definitions themselves have no security or privacy impact on the Internet, but the usage of these definitions in concrete YANG modules might have. The security considerations spelled out in the YANG specification [RFC7950] apply for this document as well.

7. Acknowledgements

The Routing Area Yang Architecture design team members included Acee Lindem, Anees Shaikh, Christian Hopps, Dean Bogdanovic, Ebben Aries, Lou Berger, Qin Wu, Rob Shakir, Xufeng Liu, and Yingzhen Qu.

Thanks to Martin Bjorkland, Tom Petch, Stewart Bryant, and Radek Krejci for comments on the model and document text. Thanks to Jeff

Haas and Robert Raszuk for suggestions for additional common routing types.

8. References

8.1. Normative References

- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [IANA-ADDRESS-FAMILY-REGISTRY]
"IANA Address Family Registry",
<<https://www.iana.org/assignments/address-family-numbers/address-family-numbers.xhtml#address-family-numbers-2>>.
- [IANA-SAFI-REGISTRY]
"IANA Subsequent Address Family Identities (SAFI) Parameters Registry", <<https://www.iana.org/assignments/safi-namespace/safi-namespace.xhtml#safi-namespace-2>>.

8.2. Informative References

- [IEEE754] IEEE, "IEEE Standard for Floating-Point Arithmetic", IEEE Std 754-2008, August 2008.
- [I-D.ietf-bfd-yang]
Rahman, R., Zheng, L., Jethanandani, M., Networks, J., and G. Mirsky, "YANG Data Model for Bidirectional Forwarding Detection (BFD)", draft-ietf-bfd-yang-06 (work in progress), June 2017.

[I-D.ietf-idr-bgp-model]

Shaikh, A., Shakir, R., Patel, K., Hares, S., D'Souza, K., Bansal, D., Clemm, A., Zhdankin, A., Jethanandani, M., and X. Liu, "BGP Model for Service Provider Networks", draft-ietf-idr-bgp-model-02 (work in progress), July 2016.

[I-D.ietf-ospf-yang]

Yeung, D., Qu, Y., Zhang, Z., Chen, I., and A. Lindem, "Yang Data Model for OSPF Protocol", draft-ietf-ospf-yang-08 (work in progress), July 2017.

[I-D.ietf-pim-yang]

Liu, X., McAllister, P., Peter, A., Sivakumar, M., Liu, Y., and f. hu, "A YANG data model for Protocol-Independent Multicast (PIM)", draft-ietf-pim-yang-10 (work in progress), September 2017.

[I-D.ietf-teas-yang-rsvp]

Beeram, V., Saad, T., Gandhi, R., Liu, X., Bryskin, I., and H. Shah, "A YANG Data Model for Resource Reservation Protocol (RSVP)", draft-ietf-teas-yang-rsvp-07 (work in progress), March 2017.

[I-D.ietf-teas-yang-te]

Saad, T., Gandhi, R., Liu, X., Beeram, V., Shah, H., and I. Bryskin, "A YANG Data Model for Traffic Engineering Tunnels and Interfaces", draft-ietf-teas-yang-te-08 (work in progress), July 2017.

[I-D.ietf-bess-l2vpn-yang]

Shah, H., Brissette, P., Chen, I., Hussain, I., Wen, B., and K. Tiruveedhula, "YANG Data Model for MPLS-based L2VPN", draft-ietf-bess-l2vpn-yang-07 (work in progress), October 2017.

[I-D.ietf-bess-l3vpn-yang]

Jain, D., Patel, K., Brissette, P., Li, Z., Zhuang, S., Liu, X., Haas, J., Esale, S., and B. Wen, "Yang Data Model for BGP/MPLS L3 VPNs", draft-ietf-bess-l3vpn-yang-01 (work in progress), April 2017.

[I-D.ietf-mpls-base-yang]

Raza, K., Gandhi, R., Liu, X., Beeram, V., Saad, T., Bryskin, I., Chen, X., Jones, R., and B. Wen, "A YANG Data Model for MPLS Base", draft-ietf-mpls-base-yang-05 (work in progress), July 2017.

- [RFC3032] Rosen, E., Tappan, D., Fedorkow, G., Rekhter, Y., Farinacci, D., Li, T., and A. Conta, "MPLS Label Stack Encoding", RFC 3032, DOI 10.17487/RFC3032, January 2001, <<https://www.rfc-editor.org/info/rfc3032>>.
- [RFC3209] Awduche, D., Berger, L., Gan, D., Li, T., Srinivasan, V., and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels", RFC 3209, DOI 10.17487/RFC3209, December 2001, <<https://www.rfc-editor.org/info/rfc3209>>.
- [RFC3471] Berger, L., Ed., "Generalized Multi-Protocol Label Switching (GMPLS) Signaling Functional Description", RFC 3471, DOI 10.17487/RFC3471, January 2003, <<https://www.rfc-editor.org/info/rfc3471>>.
- [RFC4364] Rosen, E. and Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4364, DOI 10.17487/RFC4364, February 2006, <<https://www.rfc-editor.org/info/rfc4364>>.
- [RFC4664] Andersson, L., Ed. and E. Rosen, Ed., "Framework for Layer 2 Virtual Private Networks (L2VPNs)", RFC 4664, DOI 10.17487/RFC4664, September 2006, <<https://www.rfc-editor.org/info/rfc4664>>.
- [RFC5701] Rekhter, Y., "IPv6 Address Specific BGP Extended Community Attribute", RFC 5701, DOI 10.17487/RFC5701, November 2009, <<https://www.rfc-editor.org/info/rfc5701>>.
- [RFC5880] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD)", RFC 5880, DOI 10.17487/RFC5880, June 2010, <<https://www.rfc-editor.org/info/rfc5880>>.
- [RFC7274] Kompella, K., Andersson, L., and A. Farrel, "Allocating and Retiring Special-Purpose MPLS Labels", RFC 7274, DOI 10.17487/RFC7274, June 2014, <<https://www.rfc-editor.org/info/rfc7274>>.
- [RFC7432] Sajassi, A., Ed., Aggarwal, R., Bitar, N., Isaac, A., Uttaro, J., Drake, J., and W. Henderickx, "BGP MPLS-Based Ethernet VPN", RFC 7432, DOI 10.17487/RFC7432, February 2015, <<https://www.rfc-editor.org/info/rfc7432>>.

Authors' Addresses

Xufeng Liu
Jabil
8281 Greensboro Drive, Suite 200
McLean VA 22102
USA

EMail: Xufeng_Liu@jabil.com

Yingzhen Qu
Futurewei Technologies, Inc.
2330 Central Expressway
Santa Clara CA 95050
USA

EMail: yingzhen.qu@huawei.com

Acee Lindem
Cisco Systems
301 Midenhall Way
Cary, NC 27513
USA

EMail: acee@cisco.com

Christian Hopps
Deutsche Telekom

EMail: chopps@chopps.org

Lou Berger
LabN Consulting, L.L.C.

EMail: lberger@labn.net

RIFT Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 2, 2018

T. Przygienda, Ed.
Juniper Networks
A. Sharma
Comcast
A. Atlas
J. Drake
Juniper Networks
Mar 01, 2018

RIFT: Routing in Fat Trees
draft-przygienda-rift-05

Abstract

This document outlines a specialized, dynamic routing protocol for Clos and fat-tree network topologies. The protocol (1) deals with automatic construction of fat-tree topologies based on detection of links, (2) minimizes the amount of routing state held at each level, (3) automatically prunes the topology distribution exchanges to a sufficient subset of links, (4) supports automatic disaggregation of prefixes on link and node failures to prevent black-holing and suboptimal routing, (5) allows traffic steering and re-routing policies, (6) allows non-ECMP forwarding, (7) automatically re-balances traffic towards the spines based on bandwidth available and ultimately (8) provides mechanisms to synchronize a limited key-value data-store that can be used after protocol convergence to e.g. bootstrap higher levels of functionality on nodes.

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 September 2, 2018.

Copyright Notice

Copyright (c) 2018 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	4
1.1. Requirements Language	5
2. Reference Frame	5
2.1. Terminology	5
2.2. Topology	8
3. Requirement Considerations	10
4. RIFT: Routing in Fat Trees	12
4.1. Overview	12
4.2. Specification	13
4.2.1. Transport	13
4.2.2. Link (Neighbor) Discovery (LIE Exchange)	13
4.2.3. Topology Exchange (TIE Exchange)	14
4.2.3.1. Topology Information Elements	14
4.2.3.2. South- and Northbound Representation	15
4.2.3.3. Flooding	18
4.2.3.4. TIE Flooding Scopes	18
4.2.3.5. Initial and Periodic Database Synchronization	20
4.2.3.6. Purging	20
4.2.3.7. Southbound Default Route Origination	21
4.2.3.8. Optional Automatic Flooding Reduction and Partitioning	21
4.2.4. Policy-Guided Prefixes	23
4.2.4.1. Ingress Filtering	24
4.2.4.2. Applying Policy	24
4.2.4.3. Store Policy-Guided Prefix for Route Computation and Regeneration	25
4.2.4.4. Re-origination	26
4.2.4.5. Overlap with Disaggregated Prefixes	26
4.2.5. Reachability Computation	26
4.2.5.1. Northbound SPF	27
4.2.5.2. Southbound SPF	27

4.2.5.3.	East-West Forwarding Within a Level	28
4.2.6.	Attaching Prefixes	28
4.2.7.	Attaching Policy-Guided Prefixes	29
4.2.8.	Automatic Disaggregation on Link & Node Failures . .	30
4.2.9.	Optional Autoconfiguration	33
4.2.9.1.	Terminology	34
4.2.9.2.	Automatic SystemID Selection	35
4.2.9.3.	Generic Fabric Example	35
4.2.9.4.	Level Determination Procedure	36
4.2.9.5.	Resulting Topologies	37
4.2.10.	Stability Considerations	39
4.3.	Further Mechanisms	40
4.3.1.	Overload Bit	40
4.3.2.	Optimized Route Computation on Leafs	40
4.3.3.	Key/Value Store	40
4.3.3.1.	Southbound	40
4.3.3.2.	Northbound	41
4.3.4.	Interactions with BFD	41
4.3.5.	Fabric Bandwidth Balancing	41
4.3.5.1.	Northbound Direction	42
4.3.5.2.	Southbound Direction	43
4.3.6.	Segment Routing Support with RIFT	43
4.3.6.1.	Global Segment Identifiers Assignment	43
4.3.6.2.	Distribution of Topology Information	44
4.3.7.	Leaf to Leaf Procedures	44
4.3.8.	Other End-to-End Services	45
4.3.9.	Address Family and Multi Topology Considerations .	45
4.3.10.	Reachability of Internal Nodes in the Fabric	45
4.3.11.	One-Hop Healing of Levels with East-West Links . . .	45
5.	Examples	46
5.1.	Normal Operation	46
5.2.	Leaf Link Failure	47
5.3.	Partitioned Fabric	48
5.4.	Northbound Partitioned Router and Optional East-West Links	50
6.	Implementation and Operation: Further Details	51
6.1.	Considerations for Leaf-Only Implementation	51
6.2.	Adaptations to Other Proposed Data Center Topologies .	51
6.3.	Originating Non-Default Route Southbound	52
7.	Security Considerations	52
8.	Information Elements Schema	53
8.1.	common.thrift	53
8.2.	encoding.thrift	57
9.	IANA Considerations	63
10.	Acknowledgments	63
11.	References	63
11.1.	Normative References	63
11.2.	Informative References	65

Authors' Addresses	66
------------------------------	----

1. Introduction

Clos [CLOS] and Fat-Tree [FATTREE] have gained prominence in today's networking, primarily as result of the paradigm shift towards a centralized data-center based architecture that is poised to deliver a majority of computation and storage services in the future. Today's routing protocols were geared towards a network with an irregular topology and low degree of connectivity originally but given they were the only available mechanisms, consequently several attempts to apply those to Clos have been made. Most successfully BGP [RFC4271] [RFC7938] has been extended to this purpose, not as much due to its inherent suitability to solve the problem but rather because the perceived capability to modify it "quicker" and the immanent difficulties with link-state [DIJKSTRA] based protocols to perform in large scale densely meshed topologies.

In looking at the problem through the lens of its requirements an optimal approach does not seem however to be a simple modification of either a link-state (distributed computation) or distance-vector (diffused computation) approach but rather a mixture of both, colloquially best described as "link-state towards the spine" and "distance vector towards the leafs". In other words, "bottom" levels are flooding their link-state information in the "northern" direction while each switch generates under normal conditions a default route and floods it in the "southern" direction. Obviously, such aggregation can blackhole in cases of misconfiguration or failures and this has to be addressed somehow.

For the visually oriented reader, Figure 1 presents a first simplified view of the resulting information and routes on a RIFT fabric. The top of the fabric is holding in its link-state database the nodes below it and routes to them. In the second row of the database we indicate that a partial information of other nodes in the same level is available as well; the details of how this is achieved should be postponed for the moment. Whereas when we look at the "bottom" of the fabric we see that the topology of the leafs is basically empty and they only hold a load balanced default route to the next level.

The balance of this document details the resulting protocol and fills in the missing details.

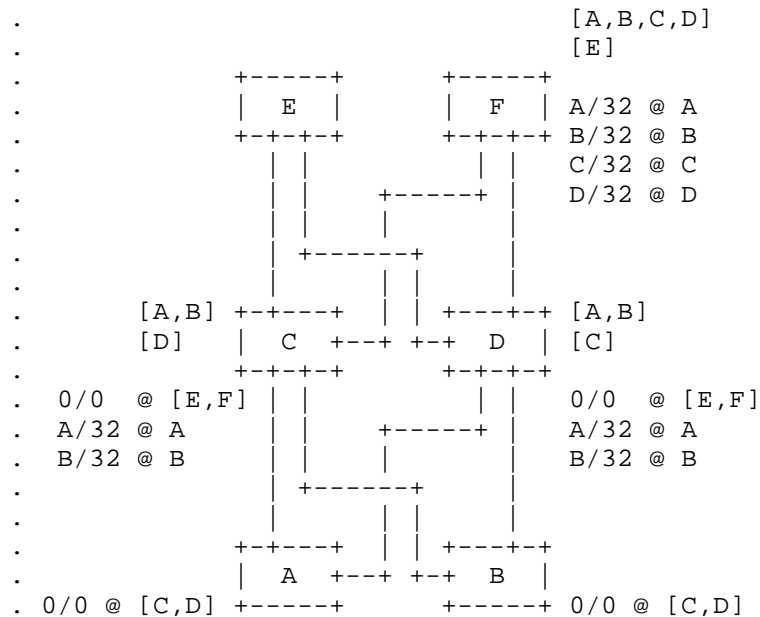


Figure 1: RIFT information distribution

1.1. 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 RFC 2119 [RFC2119].

2. Reference Frame

2.1. Terminology

This section presents the terminology used in this document. It is assumed that the reader is thoroughly familiar with the terms and concepts used in OSPF [RFC2328] and IS-IS [RFC1142], [ISO10589] as well as the according graph theoretical concepts of shortest path first (SPF) [DIJKSTRA] computation and directed acyclic graphs (DAG).

Level: Clos and Fat Tree networks are trees and 'level' denotes the set of nodes at the same height in such a network, where the bottom level is level 0. A node has links to nodes one level down and/or one level up. Under some circumstances, a node may have links to nodes at the same level. As footnote: Clos terminology uses often the concept of "stage" but due to the folded nature of the Fat Tree we do not use it to prevent misunderstandings.

Spine/Aggregation/Edge Levels: Traditional names for Level 2, 1 and 0 respectively. Level 0 is often called leaf as well.

Point of Delivery (PoD): A self-contained vertical slice of a Clos or Fat Tree network containing normally only level 0 and level 1 nodes. It communicates with nodes in other PoDs via the spine. We number PoDs to distinguish them and use PoD #0 to denote "undefined" PoD.

Spine: The set of nodes that provide inter-PoD communication. These nodes are also organized into levels (typically one, three, or five levels). Spine nodes do not belong to any PoD and are assigned the PoD value 0 to indicate this.

Leaf: A node without southbound adjacencies. Its level is 0 (except cases where it is deriving its level via ZTP and is running without LEAF_ONLY which will be explained in Section 4.2.9).

Connected Spine: In case a spine level represents a connected graph (discounting links terminating at different levels), we call it a "connected spine", in case a spine level consists of multiple partitions, we call it a "disconnected" or "partitioned spine". In other terms, a spine without east-west links is disconnected and is the typical configuration for Clos and Fat Tree networks.

South/Southbound and North/Northbound (Direction): When describing protocol elements and procedures, we will be using in different situations the directionality of the compass. I.e., 'south' or 'southbound' mean moving towards the bottom of the Clos or Fat Tree network and 'north' and 'northbound' mean moving towards the top of the Clos or Fat Tree network.

Northbound Link: A link to a node one level up or in other words, one level further north.

Southbound Link: A link to a node one level down or in other words, one level further south.

East-West Link: A link between two nodes at the same level. East-west links are normally not part of Clos or "fat-tree" topologies.

Leaf shortcuts (L2L): East-west links at leaf level will need to be differentiated from East-west links at other levels.

Southbound representation: Information sent towards a lower level representing only limited amount of information.

TIE: This is an acronym for a "Topology Information Element". TIEs are exchanged between RIFT nodes to describe parts of a network such as links and address prefixes. It can be thought of as largely equivalent to ISIS LSPs or OSPF LSA. We will talk about N-TIEs when talking about TIEs in the northbound representation and S-TIEs for the southbound equivalent.

Node TIE: This is an acronym for a "Node Topology Information Element", largely equivalent to OSPF Node LSA, i.e. it contains all neighbors the node discovered and information about node itself.

Prefix TIE: This is an acronym for a "Prefix Topology Information Element" and it contains all prefixes directly attached to this node in case of a N-TIE and in case of S-TIE the necessary default and de-aggregated prefixes the node passes southbound.

Policy-Guided Information: Information that is passed in either southbound direction or north-bound direction by the means of diffusion and can be filtered via policies. Policy-Guided Prefixes and KV Ties are examples of Policy-Guided Information.

Key Value TIE: A S-TIE that is carrying a set of key value pairs [DYNAMO]. It can be used to distribute information in the southbound direction within the protocol.

TIDE: Topology Information Description Element, equivalent to CSNP in ISIS.

TIRE: Topology Information Request Element, equivalent to PSNP in ISIS. It can both confirm received and request missing TIEs.

PGP: Policy-Guided Prefixes allow to support traffic engineering that cannot be achieved by the means of SPF computation or normal node and prefix S-TIE origination. S-PGPs are propagated in south direction only and N-PGPs follow northern direction strictly.

De-aggregation/Disaggregation: Process in which a node decides to advertise certain prefixes it received in N-TIEs to prevent black-holing and suboptimal routing upon link failures.

LIE: This is an acronym for a "Link Information Element", largely equivalent to HELLOs in IGPs and exchanged over all the links between systems running RIFT to form adjacencies.

FL: Flooding Leader for a specific system has a dedicated role to flood TIEs of that system.

BAD: This is an acronym for Bandwidth Adjusted Distance. RIFT calculates the amount of northbound bandwidth available for a node compared to other nodes at the same level and adjusts the default route distance accordingly to allow for the lower level to weight their forwarding load balancing.

Overloaded: Applies to a node advertising 'overload' attribute as set. The semantics closely follow the meaning of the same attribute in [RFC1142].

2.2. Topology

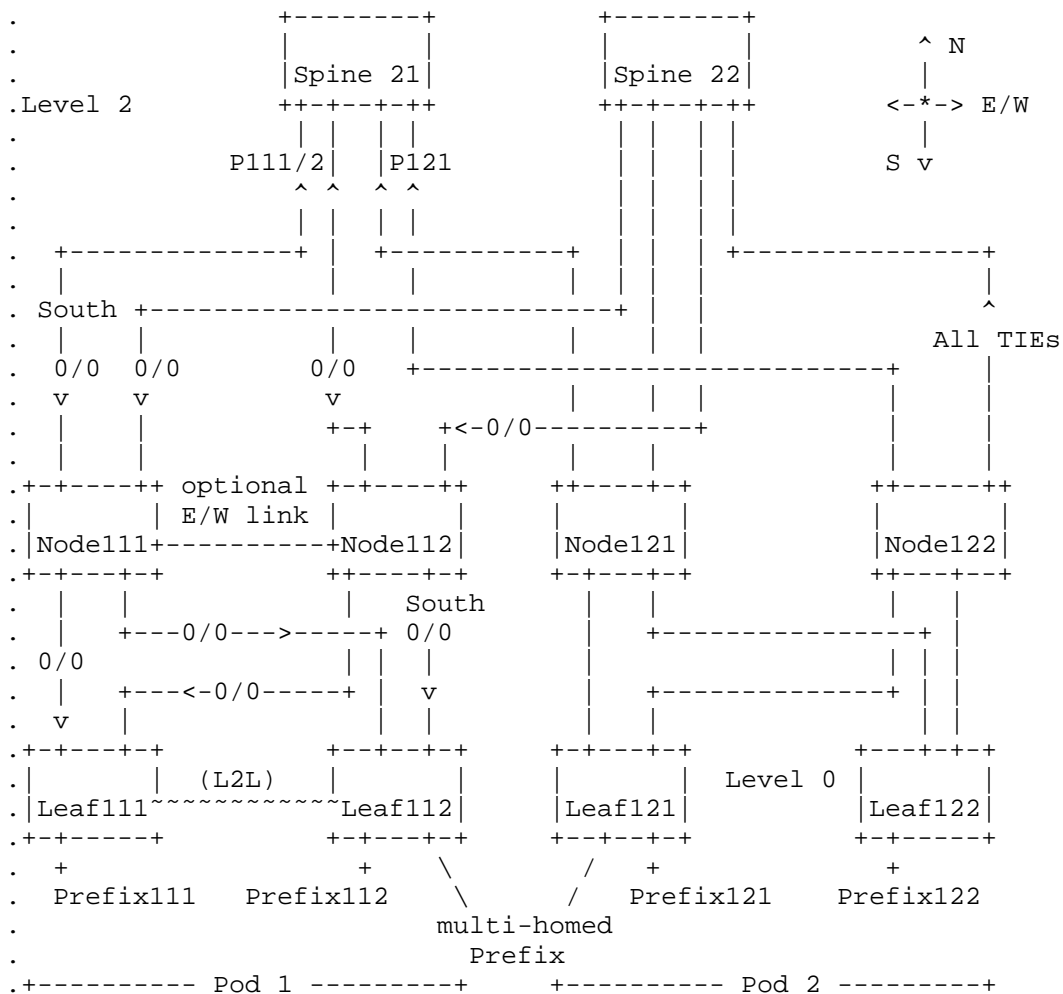


Figure 2: A two level spine-and-leaf topology

We will use this topology (called commonly a fat tree/network in modern DC considerations [VAHDAT08] as homonym to the original definition of the term [FATTREE]) in all further considerations. It depicts a generic "fat-tree" and the concepts explained in three levels here carry by induction for further levels and higher degrees of connectivity. However, this document will deal with designs that provide only sparser connectivity as well.

3. Requirement Considerations

[RFC7938] gives the original set of requirements augmented here based upon recent experience in the operation of fat-tree networks.

- REQ1: The control protocol should discover the physical links automatically and be able to detect cabling that violates fat-tree topology constraints. It must react accordingly to such mis-cabling attempts, at a minimum preventing adjacencies between nodes from being formed and traffic from being forwarded on those mis-cabled links. E.g. connecting a leaf to a spine at level 2 should be detected and ideally prevented.
- REQ2: A node without any configuration beside default values should come up at the correct level in any PoD it is introduced into. Optionally, it must be possible to configure nodes to restrict their participation to the PoD(s) targeted at any level.
- REQ3: Optionally, the protocol should allow to provision data centers where the individual switches carry no configuration information and are all deriving their level from a "seed". Observe that this requirement may collide with the desire to detect cabling misconfiguration and with that only one of the requirements can be fully met in a chosen configuration mode.
- REQ4: The solution should allow for minimum size routing information base and forwarding tables at leaf level for speed, cost and simplicity reasons. Holding excessive amount of information away from leaf nodes simplifies operation and lowers cost of the underlay.
- REQ5: Very high degree of ECMP must be supported. Maximum ECMP is currently understood as the most efficient routing approach to maximize the throughput of switching fabrics [MAKSIC2013].
- REQ6: Non equal cost anycast must be supported to allow for easy and robust multi-homing of services without regressing to careful balancing of link costs.
- REQ7: Traffic engineering should be allowed by modification of prefixes and/or their next-hops.
- REQ8: The solution should allow for access to link states of the whole topology to enable efficient support for modern

control architectures like SPRING [RFC7855] or PCE [RFC4655].

- REQ9: The solution should easily accommodate opaque data to be carried throughout the topology to subsets of nodes. This can be used for many purposes, one of them being a key-value store that allows bootstrapping of nodes based right at the time of topology discovery.
- REQ10: Nodes should be taken out and introduced into production with minimum wait-times and minimum of "shaking" of the network, i.e. radius of propagation (often called "blast radius") of changed information should be as small as feasible.
- REQ11: The protocol should allow for maximum aggregation of carried routing information while at the same time automatically de-aggregating the prefixes to prevent black-holing in case of failures. The de-aggregation should support maximum possible ECMP/N-ECMP remaining after failure.
- REQ12: Reducing the scope of communication needed throughout the network on link and state failure, as well as reducing advertisements of repeating, idiomatic or policy-guided information in stable state is highly desirable since it leads to better stability and faster convergence behavior.
- REQ13: Once a packet traverses a link in a "southbound" direction, it must not take any further "northbound" steps along its path to delivery to its destination under normal conditions. Taking a path through the spine in cases where a shorter path is available is highly undesirable.
- REQ14: Parallel links between same set of nodes must be distinguishable for SPF, failure and traffic engineering purposes.
- REQ15: The protocol must not rely on interfaces having discernible unique addresses, i.e. it must operate in presence of unnumbered links (even parallel ones) or links of a single node having same addresses.
- REQ16: It would be desirable to achieve fast re-balancing of flows when links, especially towards the spines are lost or provisioned without regressing to per flow traffic engineering which introduces significant amount of complexity while possibly not being reactive enough to account for short-lived flows.

Following list represents possible requirements and requirements under discussion:

PEND1: Supporting anything but point-to-point links is a non-requirement. Questions remain: for connecting to the leaves, is there a case where multipoint is desirable? One could still model it as point-to-point links; it seems there is no need for anything more than a NBMA-type construct.

PEND2: What is the maximum scale of number leaf prefixes we need to carry. Is 500'000 enough ?

Finally, following are the non-requirements:

NONREQ1: Broadcast media support is unnecessary.

NONREQ2: Purging is unnecessary given its fragility and complexity and today's large memory size on even modest switches and routers.

NONREQ3: Special support for layer 3 multi-hop adjacencies is not part of the protocol specification. Such support can be easily provided by using tunneling technologies the same way IGPs today are solving the problem.

4. RIFT: Routing in Fat Trees

Derived from the above requirements we present a detailed outline of a protocol optimized for Routing in Fat Trees (RIFT) that in most abstract terms has many properties of a modified link-state protocol [RFC2328][RFC1142] when "pointing north" and path-vector [RFC4271] protocol when "pointing south". Albeit an unusual combination, it does quite naturally exhibit the desirable properties we seek.

4.1. Overview

The singular property of RIFT is that it floods northbound "flat" link-state information so that each level understands the full topology of levels south of it. In contrast, in the southbound direction the protocol operates like a path vector protocol or rather a distance vector with implicit split horizon since the topology constraints make a diffused computation front propagating in all directions unnecessary.

To account for the "northern" and the "southern" information split the link state database is partitioned into "north representation" and "south representation" TIEs, whereas in simplest terms the N-TIEs contain a link state topology description of lower levels and and

S-TIEs carry simply default routes. This oversimplified view will be refined gradually in following sections while introducing protocol procedures aimed to fulfill the described requirements.

4.2. Specification

4.2.1. Transport

All protocol elements are carried over UDP. Once QUIC [QUIC] achieves the desired stability in deployments it may prove a valuable candidate for TIE transport.

All packet formats are defined in Thrift models in Section 8.

Future versions may include a [PROTOBUF] schema.

4.2.2. Link (Neighbor) Discovery (LIE Exchange)

LIE exchange happens over well-known administratively locally scoped IPv4 multicast address [RFC2365] or link-local multicast scope for IPv6 [RFC4291] and SHOULD be sent with a TTL of 1 to prevent RIFT information reaching beyond a single L3 next-hop in the topology. LIEs are exchanged over all links running RIFT.

Unless Section 4.2.9 is used, each node is provisioned with the level at which it is operating and its PoD (or otherwise a default level and "undefined" PoD are assumed; meaning that leafs do not need to be configured at all). Nodes in the spine are configured with an "undefined" PoD. This information is propagated in the LIEs exchanged.

A node tries to form a three way adjacency if and only if (definitions of LEAF_ONLY are found in Section 4.2.9)

1. the node is in the same PoD or either the node or the neighbor advertises "undefined" PoD membership (PoD# = 0) AND
2. the neighboring node is running the same MAJOR schema version AND
3. the neighbor is not member of some PoD while the node has a northbound adjacency already joining another PoD AND
4. the neighboring node uses a valid System ID AND
5. the neighboring node uses a different System ID than the node itself
6. the advertised MTUs match on both sides AND

7. both nodes advertise defined level values AND
8. [
 - i) the node is at level 0 and has no three way adjacencies already to nodes with level higher than the neighboring node OR
 - ii) the neighboring node is at level 0 OR
 - iii) both nodes are at level 0 AND both indicate support for Section 4.3.7 OR
 - iii) neither node is at level 0 and the neighboring node is at most one level away].

Rule in Paragraph 3 MAY be optionally disregarded by a node if PoD detection is undesirable or has to be disregarded.

A node configured with "undefined" PoD membership MUST, after building first northbound adjacency making it participant in a PoD, advertise that PoD as part of its LIEs.

LIEs arriving with a TTL larger than 1 MUST be ignored.

A node SHOULD NOT send out LIEs without defined level in the header but in certain scenarios it may be beneficial for trouble-shooting purposes.

LIE exchange uses three-way handshake mechanism [RFC5303]. Precise finite state machines will be provided in later versions of this specification. LIE packets contain nonces and may contain an SHA-1 [RFC6234] over nonces and some of the LIE data which prevents corruption and replay attacks. TIE flooding reuses those nonces to prevent mismatches and can use those for security purposes in case it is using QUIC [QUIC]. Section 7 will address the precise security mechanisms in the future.

4.2.3. Topology Exchange (TIE Exchange)

4.2.3.1. Topology Information Elements

Topology and reachability information in RIFT is conveyed by the means of TIEs which have good amount of commonalities with LSAs in OSPF.

TIE exchange mechanism uses port indicated by each node in the LIE exchange and the interface on which the adjacency has been formed as destination. It SHOULD use TTL of 1 as well.

TIEs contain sequence numbers, lifetimes and a type. Each type has a large identifying number space and information is spread across possibly many TIEs of a certain type by the means of a hash function that a node or deployment can individually determine. One extreme point of the design space is a prefix per TIE which leads to BGP-like behavior vs. dense packing into few TIEs leading to more traditional IGP trade-off with fewer TIEs. An implementation may even rehash at the cost of significant amount of re-advertisements of TIEs.

More information about the TIE structure can be found in the schema in Section 8.

4.2.3.2. South- and Northbound Representation

As a central concept to RIFT, each node represents itself differently depending on the direction in which it is advertising information. More precisely, a spine node represents two different databases to its neighbors depending whether it advertises TIEs to the north or to the south/sideways. We call those differing TIE databases either south- or northbound (S-TIEs and N-TIEs) depending on the direction of distribution.

The N-TIEs hold all of the node's adjacencies, local prefixes and northbound policy-guided prefixes while the S-TIEs hold only all of the node's adjacencies and the default prefix with necessary disaggregated prefixes and southbound policy-guided prefixes. We will explain this in detail further in Section 4.2.8 and Section 4.2.4.

The TIE types are symmetric in both directions and Table 1 provides a quick reference to the different TIE types including direction and their function.

TIE-Type	Content
node N-TIE	node properties, adjacencies and information helping in complex disaggregation scenarios
node S-TIE	same content as node N-TIE except the information to help disaggregation
Prefix N-TIE	contains nodes' directly reachable prefixes
Prefix S-TIE	contains originated defaults and de-aggregated prefixes
PGP N-TIE	contains nodes north PGPs
PGP S-TIE	contains nodes south PGPs
KV N-TIE	contains nodes northbound KVs
KV S-TIE	contains nodes southbound KVs

Table 1: TIE Types

As an example illustrating a databases holding both representations, consider the topology in Figure 2 with the optional link between node 111 and node 112 (so that the flooding on an east-west link can be shown). This example assumes unnumbered interfaces. First, here are the TIEs generated by some nodes. For simplicity, the key value elements and the PGP elements which may be included in their S-TIEs or N-TIEs are not shown.

Spine21 S-TIEs:

Node S-TIE:

```
NodeElement(layer=2, neighbors((Node111, layer 1, cost 1),
(Node112, layer 1, cost 1), (Node121, layer 1, cost 1),
(Node122, layer 1, cost 1)))
```

Prefix S-TIE:

```
SouthPrefixesElement(prefixes(0/0, cost 1), (::/0, cost 1))
```

Node111 S-TIEs:

Node S-TIE:

```
NodeElement(layer=1, neighbors((Spine21, layer 2, cost 1, links(...)),
    (Spine22, layer 2, cost 1, links(...)),
    (Node112, layer 1, cost 1, links(...)),
    (Leaf111, layer 0, cost 1, links(...)),
    (Leaf112, layer 0, cost 1, links(...))))
```

Prefix S-TIE:

```
SouthPrefixesElement(prefixes(0/0, cost 1), (::/0, cost 1))
```

Node111 N-TIEs:

Node N-TIE:

```
NodeElement(layer=1,
    neighbors((Spine21, layer 2, cost 1, links(...)),
    (Spine22, layer 2, cost 1, links(...)),
    (Node112, layer 1, cost 1, links(...)),
    (Leaf111, layer 0, cost 1, links(...)),
    (Leaf112, layer 0, cost 1, links(...))))
```

Prefix N-TIE:

```
NorthPrefixesElement(prefixes(Node111.loopback)
```

Node121 S-TIEs:

Node S-TIE:

```
NodeElement(layer=1, neighbors((Spine21, layer 2, cost 1),
    (Spine22, layer 2, cost 1), (Leaf121, layer 0, cost 1),
    (Leaf122, layer 0, cost 1)))
```

Prefix S-TIE:

```
SouthPrefixesElement(prefixes(0/0, cost 1), (::/0, cost 1))
```

Node121 N-TIEs:

Node N-TIE:

```
NodeLinkElement(layer=1,
    neighbors((Spine21, layer 2, cost 1, links(...)),
    (Spine22, layer 2, cost 1, links(...)),
    (Leaf121, layer 0, cost 1, links(...)),
    (Leaf122, layer 0, cost 1, links(...))))
```

Prefix N-TIE:

```
NorthPrefixesElement(prefixes(Node121.loopback)
```

Leaf112 N-TIEs:

Node N-TIE:

```
NodeLinkElement(layer=0,
    neighbors((Node111, layer 1, cost 1, links(...)),
    (Node112, layer 1, cost 1, links(...))))
```

Prefix N-TIE:

```
NorthPrefixesElement(prefixes(Leaf112.loopback, Prefix112,
    Prefix_MH))
```

Figure 3: example TIES generated in a 2 level spine-and-leaf topology

4.2.3.3. Flooding

The mechanism used to distribute TIES is the well-known (albeit modified in several respects to address fat tree requirements) flooding mechanism used by today's link-state protocols. Albeit initially more demanding to implement it avoids many problems with diffused computation update style used by path vector. As described before, TIES themselves are transported over UDP with the ports indicates in the LIE exchanges and using the destination address (for unnumbered IPv4 interfaces same considerations apply as in equivalent OSPF case) on which the LIE adjacency has been formed.

On reception of a TIE with an undefined level value in the packet header the node SHOULD issue a warning and indiscriminately discard the packet.

Precise finite state machines and procedures will be provided in later versions of this specification.

4.2.3.4. TIE Flooding Scopes

In a somewhat analogous fashion to link-local, area and domain flooding scopes, RIFT defines several complex "flooding scopes" depending on the direction and type of TIE propagated.

Every N-TIE is flooded northbound, providing a node at a given level with the complete topology of the Clos or Fat Tree network underneath it, including all specific prefixes. This means that a packet received from a node at the same or lower level whose destination is covered by one of those specific prefixes may be routed directly towards the node advertising that prefix rather than sending the packet to a node at a higher level.

A node's node S-TIES, consisting of all node's adjacencies and prefix S-TIES with default IP prefix and disaggregated prefixes, are flooded southbound in order to allow the nodes one level down to see connectivity of the higher level as well as reachability to the rest of the fabric. In order to allow a E-W disconnected node in a given level to receive the S-TIES of other nodes at its level, every *NODE* S-TIE is "reflected" northbound to level from which it was received. It should be noted that east-west links are included in South TIE flooding; those TIES need to be flooded to satisfy algorithms in Section 4.2.5. In that way nodes at same level can learn about each other without a lower level, e.g. in case of leaf level. The precise flooding scopes are given in Table 2. Those rules govern as well

what SHOULD be included in TIDEs towards neighbors. East-West flooding scopes are identical to South flooding scopes.

Node S-TIE "reflection" allows to support disaggregation on failures describes in Section 4.2.8 and flooding reduction in Section 4.2.3.8.

Packet Type vs. Peer Direction	South	North
node S-TIE	flood self-originated only	flood if TIE originator's level is higher than own level
non-node S-TIE	flood self-originated only	flood only if TIE originator is equal peer
all N-TIEs	never flood	flood always
TIDE	include TIEs in flooding scope	include TIEs in flooding scope
TIRE	include all N-TIEs and all peer's self-originated TIEs and all node S-TIEs	include only if TIE originator is equal peer

Table 2: Flooding Scopes

As an example to illustrate these rules, consider using the topology in Figure 2, with the optional link between node 111 and node 112, and the associated TIEs given in Figure 3. The flooding from particular nodes of the TIEs is given in Table 3.

Router floods to	Neighbor	TIEs
Leaf111	Node112	Leaf111 N-TIEs, Node111 node S-TIE
Leaf111	Node111	Leaf111 N-TIEs, Node112 node S-TIE
Node111	Leaf111	Node111 S-TIEs
Node111	Leaf112	Node111 S-TIEs
Node111	Node112	Node111 S-TIEs
Node111	Spine21	Node111 N-TIEs, Leaf111 N-TIEs, Leaf112 N-TIEs, Spine22 node S-TIE
Node111	Spine22	Node111 N-TIEs, Leaf111 N-TIEs, Leaf112 N-TIEs, Spine21 node S-TIE
...
Spine21	Node111	Spine21 S-TIEs
Spine21	Node112	Spine21 S-TIEs
Spine21	Node121	Spine21 S-TIEs
Spine21	Node122	Spine21 S-TIEs
...

Table 3: Flooding some TIEs from example topology

4.2.3.5. Initial and Periodic Database Synchronization

The initial exchange of RIFT is modeled after ISIS with TIDE being equivalent to CSNP and TIRE playing the role of PSNP. The content of TIDES and TIRES is governed by Table 2.

4.2.3.6. Purging

RIFT does not purge information that has been distributed by the protocol. Purging mechanisms in other routing protocols have proven to be complex and fragile over many years of experience. Abundant amounts of memory are available today even on low-end platforms. The information will age out and all computations will deliver correct results if a node leaves the network due to the new information distributed by its adjacent nodes.

Once a RIFT node issues a TIE with an ID, it MUST preserve the ID as long as feasible (also when the protocol restarts), even if the TIE loses all content. The re-advertisement of empty TIE fulfills the purpose of purging any information advertised in previous versions. The originator is free to not re-originate the according empty TIE again or originate an empty TIE with relatively short lifetime to prevent large number of long-lived empty stubs polluting the network.

Each node will timeout and clean up the according empty TIEs independently.

Upon restart a node MUST, as any link-state implementation, be prepared to receive TIEs with its own system ID and supercede them with equivalent, newly generated, empty TIEs with a higher sequence number. As above, the lifetime can be relatively short since it only needs to exceed the necessary propagation and processing delay by all the nodes that are within the TIE's flooding scope.

4.2.3.7. Southbound Default Route Origination

Under certain conditions nodes issue a default route in their South Prefix TIEs with metrics as computed in Section 4.3.5.1.

A node X that

1. is NOT overloaded AND
2. has southbound or east-west adjacencies

originates in its south prefix TIE such a default route IIF

1. all other nodes at X's' level are overloaded OR
2. all other nodes at X's' level have NO northbound adjacencies OR
3. X has computed reachability to a default route during N-SPF.

The term "all other nodes at X's' level" describes obviously just the nodes at the same level in the POD with a viable lower layer (otherwise the node S-TIEs cannot be reflected and the nodes in e.g. POD 1 nad POD 2 are "invisible" to each other).

A node originating a southbound default route MUST install a default discard route if it did not compute a default route during N-SPF.

4.2.3.8. Optional Automatic Flooding Reduction and Partitioning

Several nodes can, but strictly only under conditions defined below, run a hashing function based on TIE originator value and partition flooding between them.

Steps for flooding reduction and partitioning:

1. select all nodes in the same level for which
 - A. node S-TIEs have been received AND

- B. which have precisely the same non-empty sets of respectively north and south neighbor adjacencies AND
 - C. have at least one shared southern neighbor including backlink verification and
 - D. support flooding reduction (overload bits are ignored)
- and then
- 2. run on the chosen set a hash algorithm using nodes flood priorities and IDs to select flooding leader and backup per TIE originator ID, i.e. each node floods immediately through to all its necessary neighbors TIEs that it received with an originator ID that makes it the flooding leader or backup for this originator. The preference (higher is better) is computed as $\text{XOR}(\text{TIE-ORIGINATOR-ID} \ll 1, \sim \text{OWN-SYSTEM-ID})$, whereas \ll is a non-circular shift and \sim is bit-wise NOT.
 - 3. In the very unlikely case of hash collisions on either of the two nodes with highest values (i.e. either does NOT produce unique hashes as compared to all other hash values), the node running the election does not attempt to reduce flooding.

Additional rules for flooding reduction and partitioning:

- 1. A node always floods its own TIEs
- 2. A node generates TIDES as usual but when receiving TIREs with requests for TIEs for a node for which it is not a flooding leader or backup it ignores such TIDES on first request only. Normally, the flooding leader should satisfy the requestor and with that no further TIREs for such TIEs will be generated. Otherwise, the next set of TIDES and TIREs will lead to flooding independent of the flooding leader status.
- 3. A node receiving a TIE originated by a node for which it is not a flooding leader floods such TIEs only when receiving an out-of-date TIDE for them, except for the first one.

The mechanism can be implemented optionally in each node. The capability is carried in the node S-TIE (and for symmetry purposes in node N-TIE as well but it serves no purpose there currently).

Obviously flooding reduction does NOT apply to self originated TIEs. Observe further that all policy-guided information consists of self-originated TIEs.

4.2.4. Policy-Guided Prefixes

In a fat tree, it can be sometimes desirable to guide traffic to particular destinations or keep specific flows to certain paths. In RIFT, this is done by using policy-guided prefixes with their associated communities. Each community is an abstract value whose meaning is determined by configuration. It is assumed that the fabric is under a single administrative control so that the meaning and intent of the communities is understood by all the nodes in the fabric. Any node can originate a policy-guided prefix.

Since RIFT uses distance vector concepts in a southbound direction, it is straightforward to add a policy-guided prefix to an S-TIE. For easier troubleshooting, the approach taken in RIFT is that a node's southbound policy-guided prefixes are sent in its S-TIE and the receiver does inbound filtering based on the associated communities (an egress policy is imaginable but would lead to different S-TIEs per neighbor possibly which is not considered in RIFT protocol procedures). A southbound policy-guided prefix can only use links in the south direction. If an PGP S-TIE is received on an east-west or northbound link, it must be discarded by ingress filtering.

Conceptually, a southbound policy-guided prefix guides traffic from the leaves up to at most the north-most layer. It is also necessary to have northbound policy-guided prefixes to guide traffic from the north-most layer down to the appropriate leaves. Therefore, RIFT includes northbound policy-guided prefixes in its N PGP-TIE and the receiver does inbound filtering based on the associated communities. A northbound policy-guided prefix can only use links in the northern direction. If an N PGP TIE is received on an east-west or southbound link, it must be discarded by ingress filtering.

By separating southbound and northbound policy-guided prefixes and requiring that the cost associated with a PGP is strictly monotonically increasing at each hop, the path cannot loop. Because the costs are strictly increasing, it is not possible to have a loop between a northbound PGP and a southbound PGP. If east-west links were to be allowed, then looping could occur and issues such as counting to infinity would become an issue to be solved. If complete generality of path - such as including east-west links and using both north and south links in arbitrary sequence - then a Path Vector protocol or a similar solution must be considered.

If a node has received the same prefix, after ingress filtering, as a PGP in an S-TIE and in an N-TIE, then the node determines which policy-guided prefix to use based upon the advertised cost.

A policy-guided prefix is always preferred to a regular prefix, even if the policy-guided prefix has a larger cost. Section 8 provides normative indication of prefix preferences.

The set of policy-guided prefixes received in a TIE is subject to ingress filtering and then re-originated to be sent out in the receiver's appropriate TIE. Both the ingress filtering and the re-origination use the communities associated with the policy-guided prefixes to determine the correct behavior. The cost on re-advertisement MUST increase in a strictly monotonic fashion.

4.2.4.1. Ingress Filtering

When a node X receives a PGP S-TIE or a PGP N-TIE that is originated from a node Y which does not have an adjacency with X, all PGPs in such a TIE MUST be filtered. Similarly, if node Y is at the same layer as node X, then X MUST filter out PGPs in such S- and N-TIEs to prevent loops.

Next, policy can be applied to determine which policy-guided prefixes to accept. Since ingress filtering is chosen rather than egress filtering and per-neighbor PGPs, policy that applies to links is done at the receiver. Because the RIFT adjacency is between nodes and there may be parallel links between the two nodes, the policy-guided prefix is considered to start with the next-hop set that has all links to the originating node Y.

A policy-guided prefix has or is assigned the following attributes:

cost: This is initialized to the cost received

community_list: This is initialized to the list of the communities received.

next_hop_set: This is initialized to the set of links to the originating node Y.

4.2.4.2. Applying Policy

The specific action to apply based upon a community is deployment specific. Here are some examples of things that can be done with communities. The length of a community is a 64 bits number and it can be written as a single field M or as a multi-field (S = M[0-31], T = M[32-63]) in these examples. For simplicity, the policy-guided prefix is referred to as P, the processing node as X and the originator as Y.

Prune Next-Hops: Community Required: For each next-hop in P.next_hop_set, if the next-hop does not have the community, prune that next-hop from P.next_hop_set.

Prune Next-Hops: Avoid Community: For each next-hop in P.next_hop_set, if the next-hop has the community, prune that next-hop from P.next_hop_set.

Drop if Community: If node X has community M, discard P.

Drop if not Community: If node X does not have the community M, discard P.

Prune to ifIndex T: For each next-hop in P.next_hop_set, if the next-hop's ifIndex is not the value T specified in the community (S,T), then prune that next-hop from P.next_hop_set.

Add Cost T: For each appearance of community S in P.community_list, if the node X has community S, then add T to P.cost.

Accumulate Min-BW T: Let bw be the sum of the bandwidth for P.next_hop_set. If that sum is less than T, then replace (S,T) with (S, bw).

Add Community T if Node matches S: If the node X has community S, then add community T to P.community_list.

4.2.4.3. Store Policy-Guided Prefix for Route Computation and Regeneration

Once a policy-guided prefix has completed ingress filtering and policy, it is almost ready to store and use. It is still necessary to adjust the cost of the prefix to account for the link from the computing node X to the originating neighbor node Y.

There are three different policies that can be used:

Minimum Equal-Cost: Find the lowest cost C next-hops in P.next_hop_set and prune to those. Add C to P.cost.

Minimum Unequal-Cost: Find the lowest cost C next-hop in P.next_hop_set. Add C to P.cost.

Maximum Unequal-Cost: Find the highest cost C next-hop in P.next_hop_set. Add C to P.cost.

The default policy is Minimum Unequal-Cost but well-known communities can be defined to get the other behaviors.

Regardless of the policy used, a node MUST store a PGP cost that is at least 1 greater than the PGP cost received. This enforces the strictly monotonically increasing condition that avoids loops.

Two databases of PGPs - from N-TIEs and from S-TIEs are stored. When a PGP is inserted into the appropriate database, the usual tie-breaking on cost is performed. Observe that the node retains all PGP TIEs due to normal flooding behavior and hence loss of the best prefix will lead to re-evaluation of TIEs present and re-advertisement of a new best PGP.

4.2.4.4. Re-origination

A node must re-originate policy-guided prefixes and retransmit them. The node has its database of southbound policy-guided prefixes to send in its S-TIE and its database of northbound policy-guided prefixes to send in its N-TIE.

Of course, a leaf does not need to re-originate southbound policy-guided prefixes.

4.2.4.5. Overlap with Disaggregated Prefixes

PGPs may overlap with prefixes introduced by automatic de-aggregation. The topic is under further discussion. The break in connectivity that leads to infeasibility of a PGP is mirrored in adjacency tear-down and according removal of such PGPs. Nevertheless, the underlying link-state flooding will be likely reacting significantly faster than a hop-by-hop redistribution and with that the preference for PGPs may cause intermittent black-holes.

4.2.5. Reachability Computation

A node has three sources of relevant information. A node knows the full topology south from the received N-TIEs. A node has the set of prefixes with associated distances and bandwidths from received S-TIEs. A node can also have a set of PGPs.

To compute reachability, a node runs conceptually a northbound and a southbound SPF. We call that N-SPF and S-SPF.

Since neither computation can "loop" (with due considerations given to PGPs), it is possible to compute non-equal-cost or even k-shortest paths [EPPSTEIN] and "saturate" the fabric to the extent desired.

4.2.5.1. Northbound SPF

N-SPF uses northbound and east-west adjacencies in North Node TIEs when progressing Dijkstra. Observe that this is really just a one hop variety since South Node TIEs are not re-flooded southbound beyond a single level (or east-west) and with that the computation cannot progress beyond adjacent nodes.

Default route found when crossing an E-W link is used IIF

1. the node itself does NOT have any northbound adjacencies AND
2. the adjacent node has one or more northbound adjacencies

This rule forms a "one-hop default route split-horizon" and prevents looping over default routes while allowing for "one-hop protection" of nodes that lost all northbound adjacencies.

Other south prefixes found when crossing E-W link MAY be used IIF

1. no north neighbors are advertising same or superssuming non-default prefix AND
2. the node does not originate a non-default superssuming prefix itself.

i.e. the E-W link can be used as the gateway of last resort for a specific prefix only. Using south prefixes across E-W link can be beneficial e.g. on automatic de-aggregation in pathological fabric partitioning scenarios.

A detailed example can be found in Section 5.4.

For N-SPF we are using the South Node TIEs to find according adjacencies to verify backlink connectivity. Just as in case of IS-IS or OSPF, two unidirectional links are associated together to confirm bidirectional connectivity.

4.2.5.2. Southbound SPF

S-SPF uses only the southbound adjacencies in the south node TIEs, i.e. progresses towards nodes at lower levels. Observe that E-W adjacencies are NEVER used in the computation. This enforces the requirement that a packet traversing in a southbound direction must never change its direction.

S-SPF uses northbound adjacencies in north node TIEs to verify backlink connectivity.

4.2.5.3. East-West Forwarding Within a Level

Ultimately, it should be observed that in presence of a "ring" of E-W links in a level neither SPF will provide a "ring protection" scheme since such a computation would have to deal necessarily with breaking of "loops" in generic Dijkstra sense; an application for which RIFT is not intended. It is outside the scope of this document how an underlay can be used to provide a full-mesh connectivity between nodes in the same layer that would allow for N-SPF to provide protection for a single node loosing all its northbound adjacencies (as long as any of the other nodes in the level are northbound connected).

Using south prefixes over horizontal links is optional and can protect against pathological fabric partitioning cases that leave only paths to destinations that would necessitate multiple changes of forwarding direction between north and south.

4.2.6. Attaching Prefixes

After the SPF is run, it is necessary to attach according prefixes. For S-SPF, prefixes from an N-TIE are attached to the originating node with that node's next-hop set and a distance equal to the prefix's cost plus the node's minimized path distance. The RIFT route database, a set of (prefix, type=spf, path_distance, next-hop set), accumulates these results. Obviously, the prefix retains its type which is used to tie-break between the same prefix advertised with different types.

In case of N-SPF prefixes from each S-TIE need to also be added to the RIFT route database. The N-SPF is really just a stub so the computing node needs simply to determine, for each prefix in an S-TIE that originated from adjacent node, what next-hops to use to reach that node. Since there may be parallel links, the next-hops to use can be a set; presence of the computing node in the associated Node S-TIE is sufficient to verify that at least one link has bidirectional connectivity. The set of minimum cost next-hops from the computing node X to the originating adjacent node is determined.

Each prefix has its cost adjusted before being added into the RIFT route database. The cost of the prefix is set to the cost received plus the cost of the minimum cost next-hop to that neighbor. Then each prefix can be added into the RIFT route database with the next_hop_set; ties are broken based upon type first and then distance. RIFT route preferences are normalized by the according thrift model type.

An exemplary implementation for node X follows:

```
for each S-TIE
  if S-TIE.layer > X.layer
    next_hop_set = set of minimum cost links to the S-TIE.originator
    next_hop_cost = minimum cost link to S-TIE.originator
  end if
  for each prefix P in the S-TIE
    P.cost = P.cost + next_hop_cost
    if P not in route_database:
      add (P, type=DistVector, P.cost, next_hop_set) to route_database
    end if
    if (P in route_database) and
      (route_database[P].type is not PolicyGuided):
      if route_database[P].cost > P.cost:
        update route_database[P] with (P, DistVector, P.cost, next_hop_set)
      else if route_database[P].cost == P.cost
        update route_database[P] with (P, DistVector, P.cost,
          merge(next_hop_set, route_database[P].next_hop_set))
      else
        // Not preferred route so ignore
      end if
    end if
  end for
end for
```

Figure 4: Adding Routes from S-TIE Prefixes

4.2.7. Attaching Policy-Guided Prefixes

Each policy-guided prefix P has its cost and next_hop_set already stored in the associated database, as specified in Section 4.2.4.3; the cost stored for the PGP is already updated to considering the cost of the link to the advertising neighbor. By definition, a policy-guided prefix is preferred to a regular prefix.

```

for each policy-guided prefix P:
  if P not in route_database:
    add (P, type=PolicyGuided, P.cost, next_hop_set)
  end if
  if P in route_database :
    if (route_database[P].type is not PolicyGuided) or
      (route_database[P].cost > P.cost):
      update route_database[P] with (P, PolicyGuided, P.cost, next_hop_set
)
    else if route_database[P].cost == P.cost
      update route_database[P] with (P, PolicyGuided, P.cost,
        merge(next_hop_set, route_database[P].next_hop_set))
    else
      // Not preferred route so ignore
    end if
  end if
end for

```

Figure 5: Adding Routes from Policy-Guided Prefixes

4.2.8. Automatic Disaggregation on Link & Node Failures

Under normal circumstances, node's S-TIEs contain just the adjacencies, a default route and policy-guided prefixes. However, if a node detects that its default IP prefix covers one or more prefixes that are reachable through it but not through one or more other nodes at the same level, then it MUST explicitly advertise those prefixes in an S-TIE. Otherwise, some percentage of the northbound traffic for those prefixes would be sent to nodes without according reachability, causing it to be black-holed. Even when not black-holing, the resulting forwarding could 'backhaul' packets through the higher level spines, clearly an undesirable condition affecting the blocking probabilities of the fabric.

We refer to the process of advertising additional prefixes as 'de-aggregation' or 'dis-aggregation'.

A node determines the set of prefixes needing de-aggregation using the following steps:

1. A DAG computation in the southern direction is performed first, i.e. the N-TIEs are used to find all of prefixes it can reach and the set of next-hops in the lower level for each. Such a computation can be easily performed on a fat tree by e.g. setting all link costs in the southern direction to 1 and all northern directions to infinity. We term set of those prefixes $|R$, and for each prefix, r , in $|R$, we define its set of next-hops to

be $|H(r)$. Observe that policy-guided prefixes are NOT affected since their scope is controlled by configuration.

2. The node uses reflected S-TIEs to find all nodes at the same level in the same PoD and the set of southbound adjacencies for each. The set of nodes at the same level is termed $|N$ and for each node, n , in $|N$, we define its set of southbound adjacencies to be $|A(n)$.
3. For a given r , if the intersection of $|H(r)$ and $|A(n)$, for any n , is null then that prefix r must be explicitly advertised by the node in an S-TIE.
4. Identical set of de-aggregated prefixes is flooded on each of the node's southbound adjacencies. In accordance with the normal flooding rules for an S-TIE, a node at the lower level that receives this S-TIE will not propagate it south-bound. Neither is it necessary for the receiving node to reflect the disaggregated prefixes back over its adjacencies to nodes at the level from which it was received.

To summarize the above in simplest terms: if a node detects that its default route encompasses prefixes for which one of the other nodes in its level has no possible next-hops in the level below, it has to disaggregate it to prevent black-holing or suboptimal routing. Hence a node X needs to determine if it can reach a different set of south neighbors than other nodes at the same level, which are connected to it via at least one common south or east-west neighbor. If it can, then prefix disaggregation may be required. If it can't, then no prefix disaggregation is needed. An example of disaggregation is provided in Section 5.3.

A possible algorithm is described last:

1. Create `partial_neighbors = (empty)`, a set of neighbors with partial connectivity to the node X 's layer from X 's perspective. Each entry is a list of south neighbor of X and a list of nodes of X .layer that can't reach that neighbor.
2. A node X determines its set of southbound neighbors `X.south_neighbors`.
3. For each S-TIE originated from a node Y that X has which is at X .layer, if `Y.south_neighbors` is not the same as `X.south_neighbors` but the nodes share at least one southern neighbor, for each neighbor N in `X.south_neighbors` but not in `Y.south_neighbors`, add $(N, (Y))$ to `partial_neighbors` if N isn't there or add Y to the list for N .

4. If `partial_neighbors` is empty, then node X does not to disaggregate any prefixes. If node X is advertising disaggregated prefixes in its S-TIE, X SHOULD remove them and re-advertise its according S-TIEs.

A node X computes its SPF based upon the received N-TIEs. This results in a set of routes, each categorized by (`prefix`, `path_distance`, `next-hop-set`). Alternately, for clarity in the following procedure, these can be organized by `next-hop-set` as (`next-hops`), `{(prefix, path_distance)}`). If `partial_neighbors` isn't empty, then the following procedure describes how to identify prefixes to disaggregate.

```

disaggregated_prefixes = {empty }
nodes_same_layer = { empty }
for each S-TIE
  if (S-TIE.layer == X.layer and
      X shares at least one S-neighbor with X)
    add S-TIE.originator to nodes_same_layer
  end if
end for

for each next-hop-set NHS
  isolated_nodes = nodes_same_layer
  for each NH in NHS
    if NH in partial_neighbors
      isolated_nodes = intersection(isolated_nodes,
                                   partial_neighbors[NH].nodes)
    end if
  end for

  if isolated_nodes is not empty
    for each prefix using NHS
      add (prefix, distance) to disaggregated_prefixes
    end for
  end if
end for

copy disaggregated_prefixes to X's S-TIE
if X's S-TIE is different
  schedule S-TIE for flooding
end if

```

Figure 6: Computation to Disaggregate Prefixes

Each disaggregated prefix is sent with the accurate path_distance. This allows a node to send the same S-TIE to each south neighbor. The south neighbor which is connected to that prefix will thus have a shorter path.

Finally, to summarize the less obvious points partially omitted in the algorithms to keep them more tractable:

1. all neighbor relationships MUST perform backlink checks.
2. overload bits as introduced in Section 4.3.1 have to be respected during the computation.
3. all the lower level nodes are flooded the same disaggregated prefixes since we don't want to build an S-TIE per node and complicate things unnecessarily. The PoD containing the prefix will prefer southbound anyway.
4. disaggregated prefixes do NOT have to propagate to lower levels. With that the disturbance in terms of new flooding is contained to a single level experiencing failures only.
5. disaggregated prefix S-TIEs are not "reflected" by the lower layer, i.e. nodes within same level do NOT need to be aware which node computed the need for disaggregation.
6. The fabric is still supporting maximum load balancing properties while not trying to send traffic northbound unless necessary.

Ultimately, complex partitions of superspine on sparsely connected fabrics can lead to necessity of transitive disaggregation through multiple layers. The topic will be described and standardized in later versions of this document.

4.2.9. Optional Autoconfiguration

Each RIFT node can optionally operate in zero touch provisioning (ZTP) mode, i.e. it has no configuration (unless it is a superspine at the top of the topology or it MUST operate as leaf and/or support leaf-2-leaf procedures) and it will fully configure itself after being attached to the topology. Configured nodes and nodes operating in ZTP can be mixed and will form a valid topology if achievable. This section describes the necessary concepts and procedures.

4.2.9.1. Terminology

Automatic Level Derivation: Procedures which allow nodes without level configured to derive it automatically. Only applied if CONFIGURED_LEVEL is undefined.

UNDEFINED_LEVEL: An imaginary value that indicates that the level has not been determined and has not been configured. Schemas normally indicate that by a missing optional value without an available defined default.

LEAF_ONLY: An optional configuration flag that can be configured on a node to make sure it never leaves the "bottom of the hierarchy". SUPERSPINE_FLAG and CONFIGURED_LEVEL cannot be defined at the same time as this flag. It implies CONFIGURED_LEVEL value of 0.

CONFIGURED_LEVEL: A level value provided manually. When this is defined (i.e. it is not an UNDEFINED_LEVEL) the node is not participating in ZTP. SUPERSPINE_FLAG is ignored when this value is defined. LEAF_ONLY can be set only if this value is undefined or set to 0.

DERIVED_LEVEL: Level value computed via automatic level derivation when CONFIGURED_LEVEL is equal to UNDEFINED_LEVEL.

LEAF_2_LEAF: An optional flag that can be configured on a node to make sure it supports procedures defined in Section 4.3.7. SUPERSPINE_FLAG is ignored when set at the same time as this flag. LEAF_2_LEAF implies LEAF_ONLY and the according restrictions.

LEVEL_VALUE: In ZTP case the original definition of "level" in Section 2.1 is both extended and relaxed. First, level is defined now as LEVEL_VALUE and is the first defined value of CONFIGURED_LEVEL followed by DERIVED_LEVEL. Second, it is possible for nodes to be more than one level apart to form adjacencies if any of the nodes is at least LEAF_ONLY.

Valid Offered Level (VOL): A neighbor's level received on a valid LIE (i.e. passing all checks for adjacency formation while disregarding all clauses involving level values) persisting for the duration of the holdtime interval on the LIE. Observe that offers from nodes offering level value of 0 do not constitute VOLs (since no valid DERIVED_LEVEL can be obtained from those). Offers from LIEs with 'not_a_ztp_offer' being true are not VOLs either.

Highest Available Level (HAL): Highest defined level value seen from all VOLs received.

Highest Adjacency Three Way (HAT): Highest neighbor level of all the formed three way adjacencies for the node.

SUPERSPINE_FLAG: Configuration flag provided to all superspines. LEAF_FLAG and CONFIGURED_LEVEL cannot be defined at the same time as this flag. It implies CONFIGURED_LEVEL value of 16. In fact, it is basically a shortcut for configuring same level at all superspine nodes which is unavoidable since an initial 'seed' is needed for other ZTP nodes to derive their level in the topology.

4.2.9.2. Automatic SystemID Selection

RIFT identifies each node via a SystemID which is a 64 bits wide integer. It is relatively simple to derive a, for all practical purposes collision free, value for each node on startup. As simple examples either system MAC and two random bytes can be used or an IPv4/IPv6 router ID interface address recycled as System ID. The router MUST ensure that such identifier is not changing very frequently (at least not without sending all its TIEs with fairly short lifetimes) since otherwise the network may be left with large amounts of stale TIEs in other nodes (though this is not necessarily a serious problem if the procedures suggested in Section 7 are implemented).

4.2.9.3. Generic Fabric Example

ZTP forces us to think about miscabled or unusually cabled fabric and how such a topology can be forced into a "lattice" structure which a fabric represents (with further restrictions). Let us consider a necessary and sufficient physical cabling in Figure 7. We assume all nodes being in the same PoD.

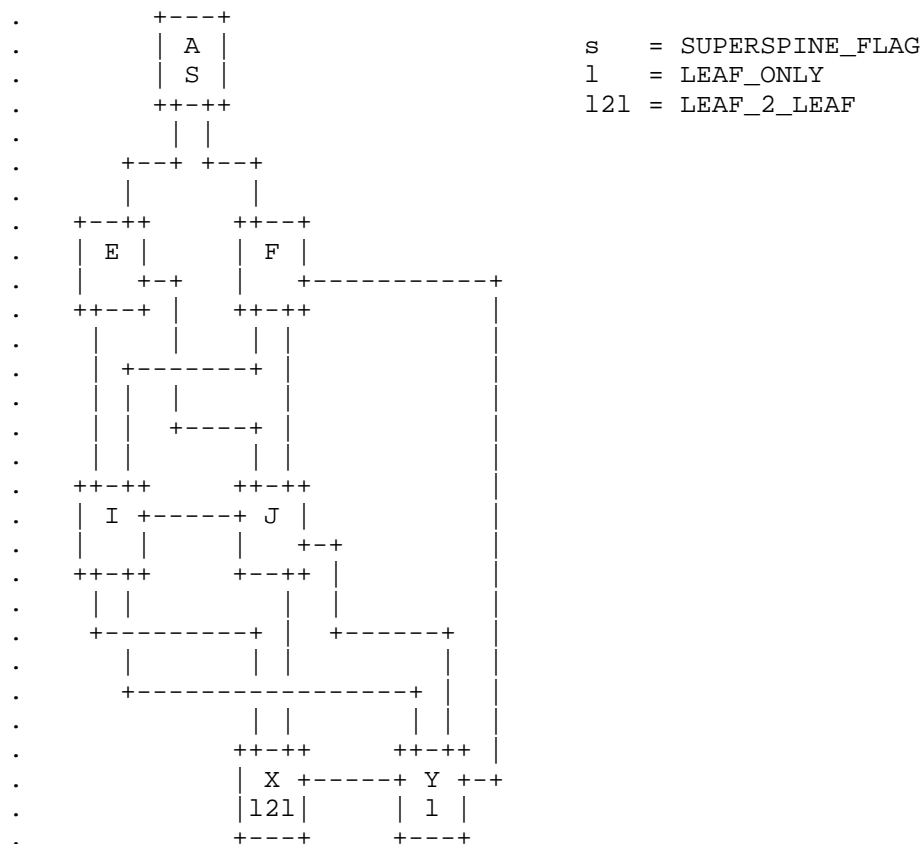


Figure 7: Generic ZTP Cabling Considerations

First, we need to anchor the "top" of the cabling and that's what the SUPERSPINE_FLAG at node A is for. Then things look smooth until we have to decide whether node Y is at the same level as I, J or at the same level as Y and consequently, X is south of it. This is unresolvable here until we "nail down the bottom" of the topology. To achieve that we use the leaf flags. We will see further then whether Y chooses to form adjacencies to F or I, J successively.

4.2.9.4. Level Determination Procedure

A node starting up with UNDEFINED_VALUE (i.e. without a CONFIGURED_LEVEL or any leaf or superspine flag) MUST follow those additional procedures:

1. It advertises its `LEVEL_VALUE` on all LIEs (observe that this can be `UNDEFINED_LEVEL` which in terms of the schema is simply an omitted optional value).
2. It chooses on an ongoing basis from all VOLs the value of $\text{MAX}(\text{HAL}-1, 0)$ as its `DERIVED_LEVEL`. The node then starts to advertise this derived level.
3. A node that lost all adjacencies with HAL value MUST hold down computation of new `DERIVED_LEVEL` for a short period of time unless it has no VOLs from southbound adjacencies. After the holddown expired, it MUST discard all received offers, recompute `DERIVED_LEVEL` and announce it to all neighbors.
4. A node MUST reset any adjacency that has changed the level it is offering and is in three way state.
5. A node that changed its defined level value MUST readvertise its own TIEs (since the new 'PacketHeader' will contain a different level than before). Sequence number of each TIE MUST be increased.
6. After a level has been derived the node MUST set the 'not_a_ztp_offer' on LIEs towards all systems extending a VOL for HAL.

A node starting with `LEVEL_VALUE` being 0 (i.e. it assumes a leaf function or has a `CONFIGURED_LEVEL` of 0) MUST follow those additional procedures:

1. It computes HAT per procedures above but does NOT use it to compute `DERIVED_LEVEL`. HAT is used to limit adjacency formation per Section 4.2.2.

Precise finite state machines will be provided in later versions of this specification.

4.2.9.5. Resulting Topologies

The procedures defined in Section 4.2.9.4 will lead to the RIFT topology and levels depicted in Figure 8.

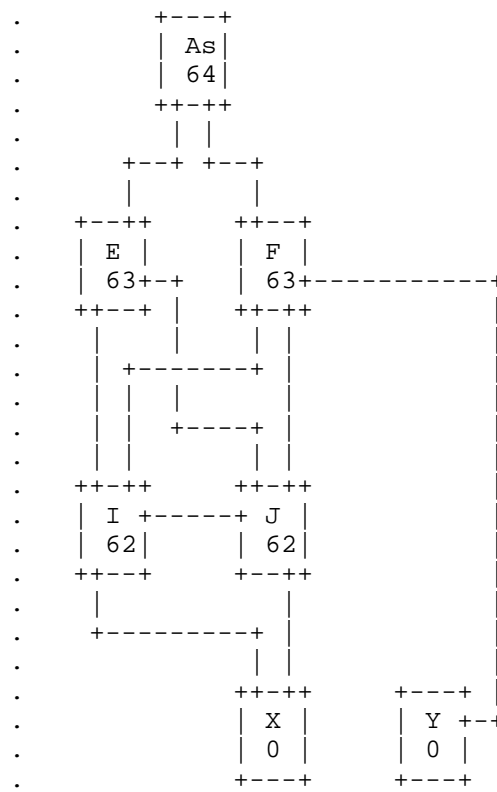


Figure 8: Generic ZTP Topology Autoconfigured

In case we imagine the LEAF_ONLY restriction on Y is removed the outcome would be very different however and result in Figure 9. This demonstrates basically that auto configuration prevents miscabling detection and with that can lead to undesirable effects when leafs are not "nailed" and arbitrarily cabled.

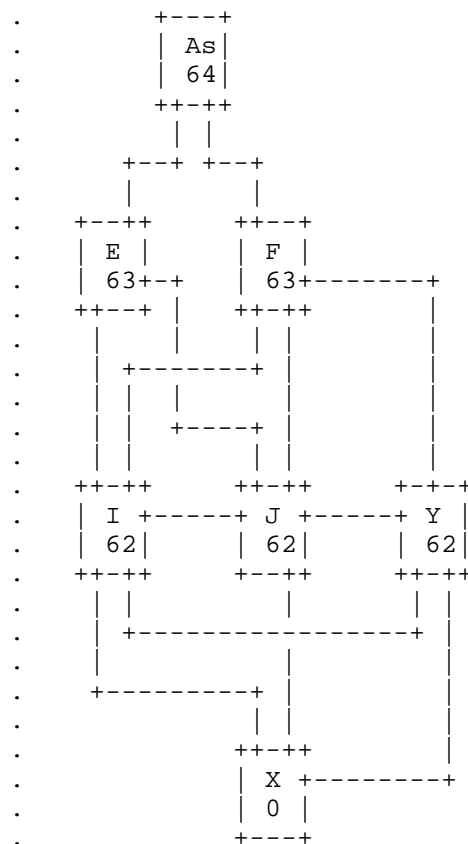


Figure 9: Generic ZTP Topology Autoconfigured

4.2.10. Stability Considerations

The autoconfiguration mechanism computes a global maximum of levels by diffusion. The achieved equilibrium can be disturbed massively by all nodes with highest level either leaving or entering the domain (with some finer distinctions not explained further). It is therefore recommended that each node is multi-homed towards nodes with respective HAL offerings. Fortunately, this is the natural state of things for the topology variants considered in RIFT.

4.3. Further Mechanisms

4.3.1. Overload Bit

Overload Bit MUST be respected in all according reachability computations. A node with overload bit set SHOULD NOT advertise any reachability prefixes southbound except locally hosted ones.

The leaf node SHOULD set the 'overload' bit on its node TIEs, since if the spine nodes were to forward traffic not meant for the local node, the leaf node does not have the topology information to prevent a routing/forwarding loop.

4.3.2. Optimized Route Computation on Leafs

Since the leafs do see only "one hop away" they do not need to run a full SPF but can simply gather prefix candidates from their neighbors and build the according routing table.

A leaf will have no N-TIEs except its own and optionally from its east-west neighbors. A leaf will have S-TIEs from its neighbors.

Instead of creating a network graph from its N-TIEs and neighbor's S-TIEs and then running an SPF, a leaf node can simply compute the minimum cost and next_hop_set to each leaf neighbor by examining its local interfaces, determining bi-directionality from the associated N-TIE, and specifying the neighbor's next_hop_set set and cost from the minimum cost local interfaces to that neighbor.

Then a leaf attaches prefixes as in Section 4.2.6 as well as the policy-guided prefixes as in Section 4.2.7.

4.3.3. Key/Value Store

4.3.3.1. Southbound

The protocol supports a southbound distribution of key-value pairs that can be used to e.g. distribute configuration information during topology bring-up. The KV S-TIEs can arrive from multiple nodes and hence need tie-breaking per key. We use the following rules

1. Only KV TIEs originated by a node to which the receiver has an adjacency are considered.
2. Within all valid KV S-TIEs containing the key, the value of the KV S-TIE for which the according node S-TIE is present, has the highest level and within the same level has highest originator ID

is preferred. If keys in the most preferred TIEs are overlapping, the behavior is undefined.

Observe that if a node goes down, the node south of it loses adjacencies to it and with that the KVs will be disregarded and on tie-break changes new KV re-advertised to prevent stale information being used by nodes further south. KV information in southbound direction is not result of independent computation of every node but a diffused computation.

4.3.3.2. Northbound

Certain use cases seem to necessitate distribution of essentially KV information that is generated in the leafs in the northbound direction. Such information is flooded in KV N-TIEs. Since the originator of northbound KV is preserved during northbound flooding, overlapping keys could be used. However, to omit further protocol complexity, only the value of the key in TIE tie-broken in same fashion as southbound KV TIEs is used.

4.3.4. Interactions with BFD

RIFT MAY incorporate BFD [RFC5881] to react quickly to link failures. In such case following procedures are introduced:

After RIFT 3-way hello adjacency convergence a BFD session MAY be formed automatically between the RIFT endpoints without further configuration.

In case RIFT loses 3-way hello adjacency, the BFD session should be brought down until 3-way adjacency is formed again.

In case established BFD session goes Down after it was Up, RIFT adjacency should be re-initialized from scratch.

In case of parallel links between nodes each link may run its own independent BFD session.

In case RIFT changes link identifiers both the hello as well as the BFD sessions will be brought down and back up again.

4.3.5. Fabric Bandwidth Balancing

A well understood problem in fabrics is that in case of link losses it would be ideal to rebalance how much traffic is offered to switches in the next layer based on the ingress and egress bandwidth they have. Current attempts rely mostly on specialized traffic

engineering via controller or leafs being aware of complete topology with according cost and complexity.

RIFT presents a very light weight mechanism that can deal with the problem in an approximative way based on the fact that RIFT is loop-free.

4.3.5.1. Northbound Direction

In a first step, a node can compare the amount of northbound bandwidth available to neighbors at the same level and modify metric on its advertised default route (or even other routes) to present a different distance leading to e.g. e.g. weighted ECMP forwarding on leafs. We call such a distance Bandwidth Adjusted Distance or BAD. This is best illustrated by a simple example.

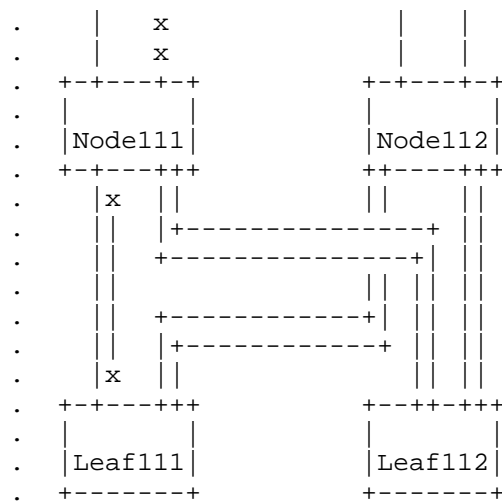


Figure 10: Balancing Bandwidth

All links in Figure 10 are assumed to have the same bandwidth for simplicity. Node 111 sees in the node S-TIE of 112 that Node 112 has twice the amount of bandwidth going northbound and therefore Node 111 will advertise its default route cost (BAD) as twice the default which without further failures would lead to Leaf 111 and Leaf 112 distributing 2/3 of the traffic to Node 111 and 1/3 to Node 112.

Further, in Figure 10 we assume that Leaf111 lost one of the parallel links to Node 111 and with that wants to push more traffic onto Node 112. This leads to local modification of the received BADs and each

node can choose the ratio here independently based on understanding of e.g. traffic distribution between E-W and N-S or queue occupancy. If we assume that 50% of the leaf's traffic is for Leaf112 and 50% exits northbound we would modify the BADs accordingly to the bandwidth available towards each of them and end in Leaf 111 with a weight of 4 to Node 111 and weight of 1 to Node 112 which gives us roughly 4/5 of the traffic going to Node 112.

Future version of this document will provide the precise algorithm to compute BADs from all other nodes at the same level using the same algorithm as Section 4.2.3.8 while ignoring overloaded nodes.

Observe that since BAD is only computed for default routes any disaggregated prefixes or PGP are not affected.

Observe further that a change in available bandwidth will only affect one level down in the fabric, i.e. blast radius of bandwidth changes is contained.

4.3.5.2. Southbound Direction

Due to its loop free properties a node could take during S-SPF into account the available bandwidth on the nodes in lower layers and modify the amount of traffic offered to next level's "southbound" nodes based as what it sees is the total achievable maximum flow through those nodes. It is worth observing that such computations will work better if standardized but does not have to be necessarily. As long the packet keeps on heading south it will take one of the available paths and arrive at the intended destination.

Future versions of this document will fill in more details.

4.3.6. Segment Routing Support with RIFT

Recently, alternative architecture to reuse labels as segment identifiers [I-D.ietf-spring-segment-routing] has gained traction and may present use cases in DC fabric that would justify its deployment. Such use cases will either precondition an assignment of a label per node (or other entities where the mechanisms are equivalent) or a global assignment and a knowledge of topology everywhere to compute segment stacks of interest. We deal with the two issues separately.

4.3.6.1. Global Segment Identifiers Assignment

Global segment identifiers are normally assumed to be provided by some kind of a centralized "controller" instance and distributed to other entities. This can be performed in RIFT by attaching a controller to the superspine nodes at the top of the fabric where the

whole topology is always visible, assign such identifiers and then distribute those via the KV mechanism towards all nodes so they can perform things like probing the fabric for failures using a stack of segments.

4.3.6.2. Distribution of Topology Information

Some segment routing use cases seem to precondition full knowledge of fabric topology in all nodes which can be performed albeit at the loss of one of highly desirable properties of RIFT, namely minimal blast radius. Basically, RIFT can function as a flat IGP by switching off its flooding scopes. All nodes will end up with full topology view and albeit the N-SPF and S-SPF are still performed based on RIFT rules, any computation with segment identifiers that needs full topology can use it.

Beside blast radius problem, excessive flooding may present significant load on implementations. RIFT can be extended beside the mechanism in Section 4.2.3.8 to provide an algorithm for globally optimized flooding minimalization should demand for such a use case solidify.

4.3.7. Leaf to Leaf Procedures

RIFT can optionally allow special leaf East-West adjacencies under additional set of rules. The leaf supporting those procedures MUST:

- advertise the LEAF_2_LEAF flag in node capabilities AND
- set the overload bit on all leaf's node TIEs AND
- flood only node's own north and south TIEs over E-W leaf adjacencies AND
- always use E-W leaf adjacency in both north as well as south computation AND
- install a discard route for any advertised aggregate in leaf's TIEs AND
- never form southbound adjacencies.

This will allow the E-W leaf nodes to exchange traffic strictly for the prefixes advertised in each other's north prefix TIEs (since the southbound computation will find the reverse direction in the other node's TIE and install its north prefixes).

4.3.8. Other End-to-End Services

Losing full, flat topology information at every node will have an impact on some of the end-to-end network services. This is the price paid for minimal disturbance in case of failures and reduced flooding and memory requirements on nodes lower south in the level hierarchy.

4.3.9. Address Family and Multi Topology Considerations

Multi-Topology (MT)[RFC5120] and Multi-Instance (MI)[RFC6822] is used today in link-state routing protocols to support several domains on the same physical topology. RIFT supports this capability by carrying transport ports in the LIE protocol exchanges. Multiplexing of LIEs can be achieved by either choosing varying multicast addresses or ports on the same address.

BFD interactions in Section 4.3.4 are implementation dependent when multiple RIFT instances run on the same link.

4.3.10. Reachability of Internal Nodes in the Fabric

RIFT does not precondition that its nodes have reachable addresses albeit for operational purposes this is clearly desirable. Under normal operating conditions this can be easily achieved by e.g. injecting the node's loopback address into North Prefix TIEs.

Things get more interesting in case a node loses all its northbound adjacencies but is not at the top of the fabric. In such a case a node that detects that some other members at its level are advertising northbound adjacencies MAY inject its loopback address into southbound PGP TIE and become reachable "from the south" that way. Further, a solution may be implemented where based on e.g. a "well known" community such a southbound PGP is reflected at level 0 and advertised as northbound PGP again to allow for "reachability from the north" at the cost of additional flooding.

4.3.11. One-Hop Healing of Levels with East-West Links

Based on the rules defined in Section 4.2.5, Section 4.2.3.7 and given presence of E-W links, RIFT can provide a one-hop protection of nodes that lost all their northbound links or in other complex link set failure scenarios. Section 5.4 explains the resulting behavior based on one such example.

5. Examples

5.1. Normal Operation

This section describes RIFT deployment in the example topology without any node or link failures. We disregard flooding reduction for simplicity's sake.

As first step, the following bi-directional adjacencies will be created (and any other links that do not fulfill LIE rules in Section 4.2.2 disregarded):

1. Spine 21 (PoD 0) to Node 111, Node 112, Node 121, and Node 122
2. Spine 22 (PoD 0) to Node 111, Node 112, Node 121, and Node 122
3. Node 111 to Leaf 111, Leaf 112
4. Node 112 to Leaf 111, Leaf 112
5. Node 121 to Leaf 121, Leaf 122
6. Node 122 to Leaf 121, Leaf 122

Consequently, N-TIEs would be originated by Node 111 and Node 112 and each set would be sent to both Spine 21 and Spine 22. N-TIEs also would be originated by Leaf 111 (w/ Prefix 111) and Leaf 112 (w/ Prefix 112 and the multi-homed prefix) and each set would be sent to Node 111 and Node 112. Node 111 and Node 112 would then flood these N-TIEs to Spine 21 and Spine 22.

Similarly, N-TIEs would be originated by Node 121 and Node 122 and each set would be sent to both Spine 21 and Spine 22. N-TIEs also would be originated by Leaf 121 (w/ Prefix 121 and the multi-homed prefix) and Leaf 122 (w/ Prefix 122) and each set would be sent to Node 121 and Node 122. Node 121 and Node 122 would then flood these N-TIEs to Spine 21 and Spine 22.

At this point both Spine 21 and Spine 22, as well as any controller to which they are connected, would have the complete network topology. At the same time, Node 111/112/121/122 hold only the N-ties of level 0 of their respective PoD. Leafs hold only their own N-TIEs.

S-TIEs with adjacencies and a default IP prefix would then be originated by Spine 21 and Spine 22 and each would be flooded to Node 111, Node 112, Node 121, and Node 122. Node 111, Node 112, Node 121, and Node 122 would each send the S-TIE from Spine 21 to Spine 22 and

the S-TIE from Spine 22 to Spine 21. (S-TIEs are reflected up to level from which they are received but they are NOT propagated southbound.)

An S Tie with a default IP prefix would be originated by Node 111 and Node 112 and each would be sent to Leaf 111 and Leaf 112. Leaf 111 and Leaf 112 would each send the S-TIE from Node 111 to Node 112 and the S-TIE from Node 112 to Node 111.

Similarly, an S Tie with a default IP prefix would be originated by Node 121 and Node 122 and each would be sent to Leaf 121 and Leaf 122. Leaf 121 and Leaf 122 would each send the S-TIE from Node 121 to Node 122 and the S-TIE from Node 122 to Node 121. At this point IP connectivity with maximum possible ECMP has been established between the leafs while constraining the amount of information held by each node to the minimum necessary for normal operation and dealing with failures.

5.2. Leaf Link Failure

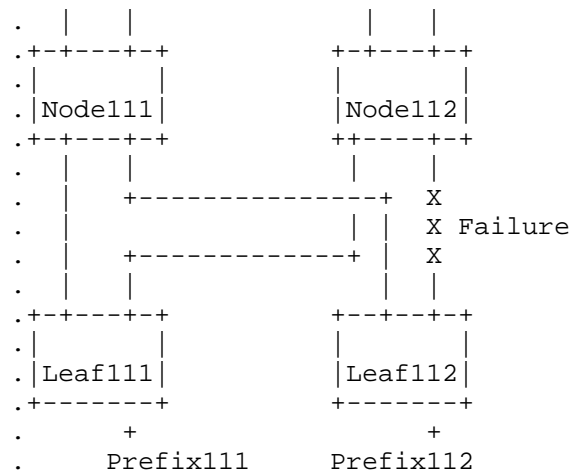


Figure 11: Single Leaf link failure

In case of a failing leaf link between node 112 and leaf 112 the link-state information will cause re-computation of the necessary SPF and the higher levels will stop forwarding towards prefix 112 through node 112. Only nodes 111 and 112, as well as both spines will see control traffic. Leaf 111 will receive a new S-TIE from node 112 and reflect back to node 111. Node 111 will de-aggregate prefix 111 and prefix 112 but we will not describe it further here since de-aggregation is emphasized in the next example. It is worth observing

however in this example that if leaf 111 would keep on forwarding traffic towards prefix 112 using the advertised south-bound default of node 112 the traffic would end up on spine 21 and spine 22 and cross back into pod 1 using node 111. This is arguably not as bad as black-holing present in the next example but clearly undesirable. Fortunately, de-aggregation prevents this type of behavior except for a transitory period of time.

5.3. Partitioned Fabric

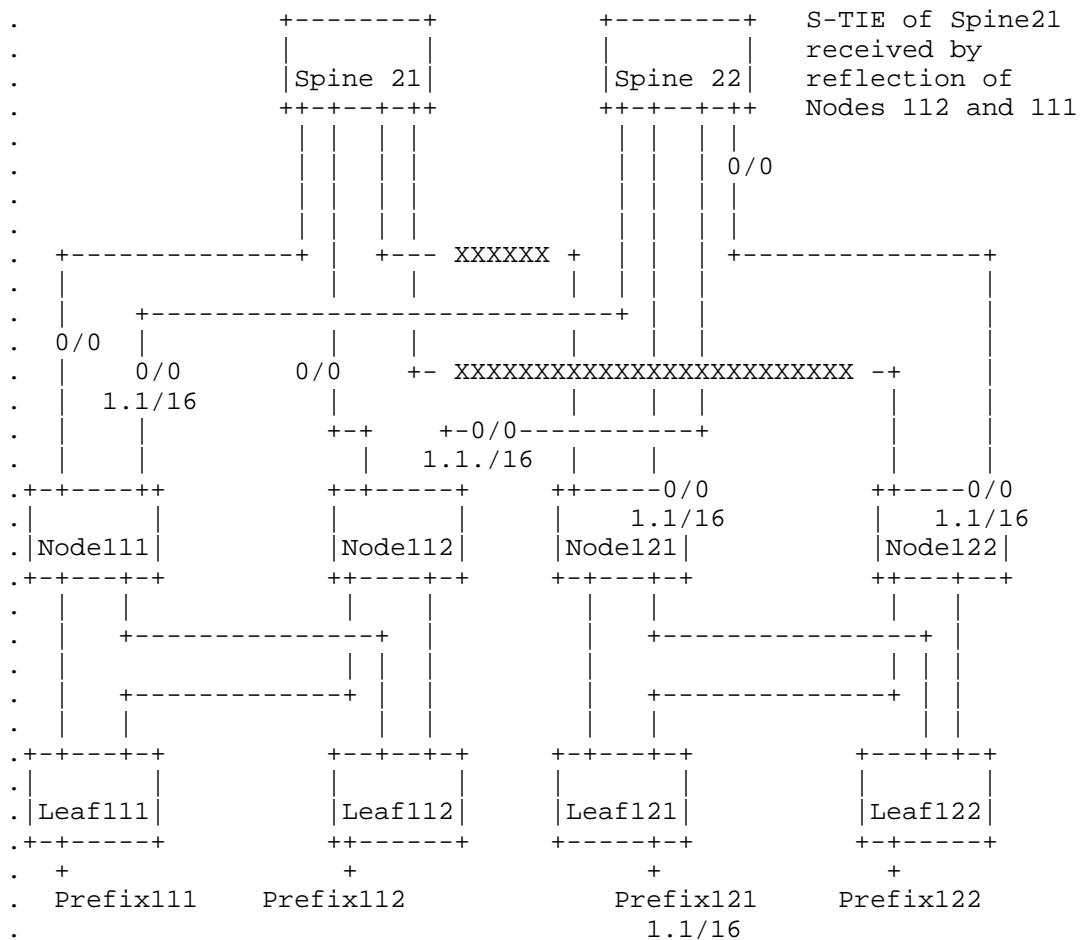


Figure 12: Fabric partition

Figure 12 shows the arguably most catastrophic but also the most interesting case. Spine 21 is completely severed from access to

Prefix 121 (we use in the figure 1.1/16 as example) by double link failure. However unlikely, if left unresolved, forwarding from leaf 111 and leaf 112 to prefix 121 would suffer 50% black-holing based on pure default route advertisements by spine 21 and spine 22.

The mechanism used to resolve this scenario is hinging on the distribution of southbound representation by spine 21 that is reflected by node 111 and node 112 to spine 22. Spine 22, having computed reachability to all prefixes in the network, advertises with the default route the ones that are reachable only via lower level neighbors that spine 21 does not show an adjacency to. That results in node 111 and node 112 obtaining a longest-prefix match to prefix 121 which leads through spine 22 and prevents black-holing through spine 21 still advertising the 0/0 aggregate only.

The prefix 121 advertised by spine 22 does not have to be propagated further towards leafs since they do no benefit from this information. Hence the amount of flooding is restricted to spine 21 reissuing its S-TIEs and reflection of those by node 111 and node 112. The resulting SPF in spine 22 issues a new prefix S-TIEs containing 1.1/16. None of the leafs become aware of the changes and the failure is constrained strictly to the level that became partitioned.

To finish with an example of the resulting sets computed using notation introduced in Section 4.2.8, spine 22 constructs the following sets:

|R = Prefix 111, Prefix 112, Prefix 121, Prefix 122

|H (for r=Prefix 111) = Node 111, Node 112

|H (for r=Prefix 112) = Node 111, Node 112

|H (for r=Prefix 121) = Node 121, Node 122

|H (for r=Prefix 122) = Node 121, Node 122

|A (for Spine 21) = Node 111, Node 112

With that and |H (for r=prefix 121) and |H (for r=prefix 122) being disjoint from |A (for spine 21), spine 22 will originate an S-TIE with prefix 121 and prefix 122, that is flooded to nodes 112, 112, 121 and 122.

5.4. Northbound Partitioned Router and Optional East-West Links

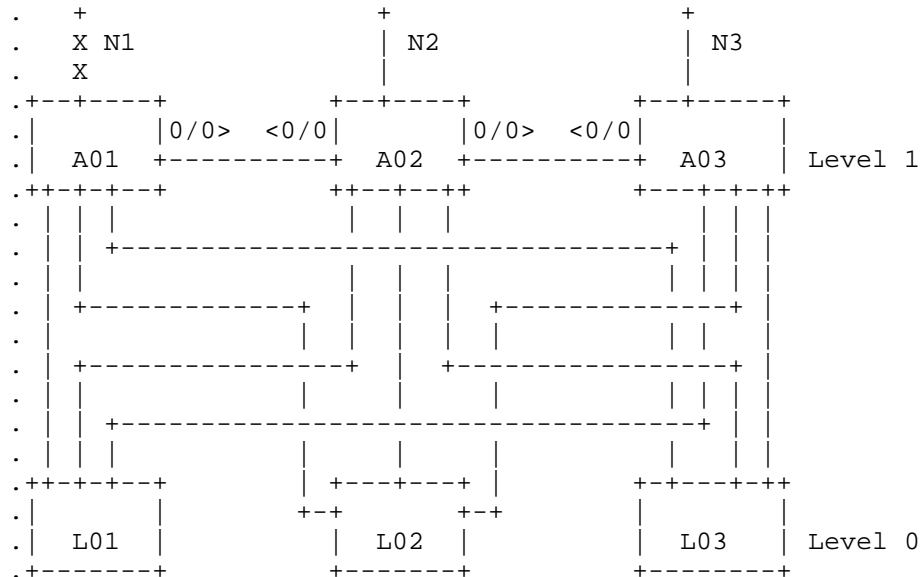


Figure 13: North Partitioned Router

Figure 13 shows a part of a fabric where level 1 is horizontally connected and A01 lost its only northbound adjacency. Based on N-SPF rules in Section 4.2.5.1 A01 will compute northbound reachability by using the link A01 to A02 (whereas A02 will NOT use this link during N-SPF). Hence A01 will still advertise the default towards level 0 and route unidirectionally using the horizontal link. Moreover, based on Section 4.3.10 it may advertise its loopback address as south PGP to remain reachable "from the south" for operational purposes. This is necessary since A02 will NOT route towards A01 using the E-W link (doing otherwise may form routing loops).

As further consideration, the moment A02 loses link N2 the situation evolves again. A01 will have no more northbound reachability while still seeing A03 advertising northbound adjacencies in its south node tie. With that it will stop advertising a default route due to Section 4.2.3.7. Moreover, A02 may now inject its loopback address as south PGP.

6. Implementation and Operation: Further Details

6.1. Considerations for Leaf-Only Implementation

Ideally RIFT can be stretched out to the lowest level in the IP fabric to integrate ToRs or even servers. Since those entities would run as leafs only, it is worth to observe that a leaf only version is significantly simpler to implement and requires much less resources:

1. Under normal conditions, the leaf needs to support a multipath default route only. In worst partitioning case it has to be capable of accommodating all the leaf routes in its own POD to prevent black-holing.
2. Leaf nodes hold only their own N-TIEs and S-TIEs of Level 1 nodes they are connected to; so overall few in numbers.
3. Leaf node does not have to support flooding reduction and de-aggregation.
4. Unless optional leaf-2-leaf procedures are desired default route origination, S-TIE origination is unnecessary.

6.2. Adaptations to Other Proposed Data Center Topologies

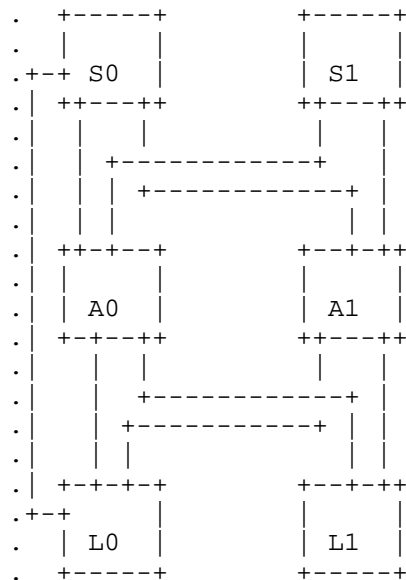


Figure 14: Level Shortcut

Strictly speaking, RIFT is not limited to Clos variations only. The protocol preconditions only a sense of 'compass rose direction' achieved by configuration (or derivation) of levels and other topologies are possible within this framework. So, conceptually, one could include leaf to leaf links and even shortcut between layers but certain requirements in Section 3 will not be met anymore. As an example, shortcutting levels illustrated in Figure 14 will lead either to suboptimal routing when L0 sends traffic to L1 (since using S0's default route will lead to the traffic being sent back to A0 or A1) or the leafs need each other's routes installed to understand that only A0 and A1 should be used to talk to each other.

Whether such modifications of topology constraints make sense is dependent on many technology variables and the exhausting treatment of the topic is definitely outside the scope of this document.

6.3. Originating Non-Default Route Southbound

Obviously, an implementation may choose to originate southbound instead of a strict default route (as described in Section 4.2.3.7) a shorter prefix P' but in such a scenario all addresses carried within the RIFT domain must be contained within P'.

7. Security Considerations

The protocol has provisions for nonces and can include authentication mechanisms in the future comparable to [RFC5709] and [RFC7987].

One can consider additionally attack vectors where a router may reboot many times while changing its system ID and pollute the network with many stale TIEs or TIEs are sent with very long lifetimes and not cleaned up when the routes vanishes. Those attack vectors are not unique to RIFT. Given large memory footprints available today those attacks should be relatively benign. Otherwise a node can implement a strategy of e.g. discarding contents of all TIEs of nodes that were not present in the SPF tree over a certain period of time. Since the protocol, like all modern link-state protocols, is self-stabilizing and will advertise the presence of such TIEs to its neighbors, they can be re-requested again if a computation finds that it sees an adjacency formed towards the system ID of the discarded TIEs.

Section 4.2.9 presents many attack vectors in untrusted environments, starting with nodes that oscillate their level offers to the possibility of a node offering a three way adjacency with the highest possible level value with a very long holdtime trying to put itself "on top of the lattice" and with that gaining access to the whole

southbound topology. Session authentication mechanisms are necessary in environments where this is possible.

8. Information Elements Schema

This section introduces the schema for information elements.

On schema changes that

1. change field numbers or
2. add new required fields or
3. remove fields or
4. change lists into sets, unions into structures or
5. change multiplicity of fields or
6. changes name of any field
7. change datatypes of any field or
8. changes default value of any field

major version of the schema MUST increase. All other changes MUST increase minor version within the same major.

Thrift serializer/deserializer MUST not discard optional, unknown fields but preserve and serialize them again when re-flooding whereas missing optional fields MAY be replaced with according default values if present.

All signed integer as forced by Thrift support must be cast for internal purposes to equivalent unsigned values without discarding the signedness bit. An implementation SHOULD try to avoid using the signedness bit when generating values.

The schema is normative.

8.1. common.thrift

```
/**  
    Thrift file with common definitions for RIFT  
*/  
  
namespace * common
```

```
/** @note MUST be interpreted in implementation as unsigned 64 bits.
 *      The implementation SHOULD NOT use the MSB.
 */
typedef i64      SystemIDType
typedef i32      IPv4Address
/** this has to be of length long enough to accomodate prefix */
typedef binary   IPv6Address
/** @note MUST be interpreted in implementation as unsigned 16 bits */
typedef i16      UDPPortType
/** @note MUST be interpreted in implementation as unsigned 32 bits */
typedef i32      TIENrType
/** @note MUST be interpreted in implementation as unsigned 32 bits */
typedef i32      MTUSizeType
/** @note MUST be interpreted in implementation as unsigned 32 bits */
typedef i32      SeqNrType
/** @note MUST be interpreted in implementation as unsigned 32 bits */
typedef i32      LifeTimeInSecType
/** @note MUST be interpreted in implementation as unsigned 16 bits */
typedef i16      LevelType
/** @note MUST be interpreted in implementation as unsigned 32 bits */
typedef i32      PodType
/** @note MUST be interpreted in implementation as unsigned 16 bits */
typedef i16      VersionType
/** @note MUST be interpreted in implementation as unsigned 32 bits */
typedef i32      MetricType
/** @note MUST be interpreted in implementation as unsigned 32 bits */
typedef i32      BandwithInMegaBitsType
typedef string   KeyIDType
/** node local, unique identification for a link (interface/tunnel
 * etc. Basically anything RIFT runs on). This is kept
 * at 32 bits so it aligns with BFD [RFC5880] discriminator size.
 */
typedef i32      LinkIDType
typedef string   KeyNameType
typedef i8       PrefixLenType
/** timestamp in seconds since the epoch */
typedef i64      TimestampInSecsType
/** security nonce */
typedef i64      NonceType
/** adjacency holdtime */
typedef i16      HoldTimeInSecType

/** Flags indicating nodes behavior in case of ZTP and support
 * for special optimization procedures. It will force level to 'leaf_level'
 */
enum LeafIndications {
    leaf_only = 0,
    leaf_only_and_leaf_2_leaf_procedures = 1,
```

```

}

/** default bandwidth on a link */
const BandwidthInMegaBitsType default_bandwidth = 10
/** fixed leaf level when ZTP is not used */
const LevelType leaf_level = 0
const LevelType default_level = leaf_level
/** This MUST be used when node is configured as superspine in ZTP.
    This is kept reasonably low to allow for fast ZTP convergence on
    failures. */
const LevelType default_superspine_level = 24
const PodType default_pod = 0
const LinkIDType undefined_linkid = 0
/** default distance used */
const MetricType default_distance = 1
/** any distance larger than this will be considered infinity */
const MetricType infinite_distance = 0x70000000
/** any element with 0 distance will be ignored,
    * missing metrics will be replaced with default_distance
    */
const MetricType invalid_distance = 0
const bool overload_default = false
const bool flood_reduction_default = true
const HoldTimeInSecType default_holdtime = 3
/** by default LIE levels are ZTP offers */
const bool default_not_a_ztp_offer = false
/** 0 is illegal for SystemID */
const SystemIDType IllegalSystemID = 0
/** empty set of nodes */
const set<SystemIDType> empty_set_of_nodeids = {}

/** normalized bandwidth metric maximum, i.e. node with lowest northbound bandwidth
    * at its level uses this metric to advertise its default route */
const MetricType normalized_bw_metric_max = 0x1fff
/** normalized bandwidth metric minimum, i.e. node with highest northbound bandwidth
    * at its level uses this metric to advertise its default route */
const MetricType normalized_bw_metric_min = 0x00ff

/** default UDP port to run LIEs on */
const UDPPortType default_lie_udp_port = 6949
const UDPPortType default_tie_udp_flood_port = 6950

/** default MTU size to use */
const MTUSizeType default_mtu_size = 1400
/** default mcast is v4 224.0.1.150, we make it i64 to
    * help languages struggling with highest bit */
const i64 default_lie_v4_mcast_group = 3758096790

```

```
/** indicates whether the direction is northbound/east-west
 * or southbound */
enum TieDirectionType {
    Illegal          = 0,
    South            = 1,
    North            = 2,
    DirectionMaxValue = 3,
}

enum AddressFamilyType {
    Illegal          = 0,
    AddressFamilyMinValue = 1,
    IPv4             = 2,
    IPv6             = 3,
    AddressFamilyMaxValue = 4,
}

struct IPv4PrefixType {
    1: required IPv4Address    address;
    2: required PrefixLenType  prefixlen;
}

struct IPv6PrefixType {
    1: required IPv6Address    address;
    2: required PrefixLenType  prefixlen;
}

union IPAddressType {
    1: optional IPv4Address    ipv4address;
    2: optional IPv6Address    ipv6address;
}

union IPPrefixType {
    1: optional IPv4PrefixType  ipv4prefix;
    2: optional IPv6PrefixType  ipv6prefix;
}

enum TIETypeType {
    Illegal          = 0,
    TIETypeMinValue  = 1,
    /** first legal value */
    NodeTIEType      = 2,
    PrefixTIEType     = 3,
    TransitivePrefixTIEType = 4,
    PGPPrefixTIEType  = 5,
    KeyValueTIEType   = 6,
    TIETypeMaxValue   = 7,
}
```



```
/** @note: route types which MUST be ordered on their preference
 * PGP prefixes are most preferred attracting
 * traffic north (towards spine) and then south
 * normal prefixes are attracting traffic south (towards leafs),
 * i.e. prefix in NORTH PREFIX TIE is preferred over SOUTH PREFIX TIE
 */
enum RouteType {
    Illegal                = 0,
    RouteTypeMinValue      = 1,
    /** First legal value. */
    /** Discard routes are most preferred */
    Discard                = 2,

    /** Local prefixes are directly attached prefixes on the
     * system such as e.g. interface routes.
     */
    LocalPrefix            = 3,
    /** advertised in S-TIEs */
    SouthPGPPrefix         = 4,
    /** advertised in N-TIEs */
    NorthPGPPrefix         = 5,
    /** advertised in N-TIEs */
    NorthPrefix            = 6,
    /** advertised in S-TIEs */
    SouthPrefix            = 7,
    /** transitive southbound are least preferred */
    TransitiveSouthPrefix  = 8,
    RouteTypeMaxValue      = 9
}
```

8.2. encoding.thrift

```
/**
 * Thrift file for packet encodings for RIFT
 */

include "common.thrift"

/** represents protocol encoding schema major version */
const i32 protocol_major_version = 8
/** represents protocol encoding schema minor version */
const i32 protocol_minor_version = 0

/** common RIFT packet header */
struct PacketHeader {
```

```
1: required common.VersionType major_version = protocol_major_version;
2: required common.VersionType minor_version = protocol_minor_version;
/** this is the node sending the packet, in case of LIE/TIRE/TIDE
    also the originator of it */
3: required common.SystemIDType sender;
/** level of the node sending the packet, required on everything except
    * LIEs. Lack of presence on LIEs indicates UNDEFINED_LEVEL and is used
    * in ZTP procedures.
    */
4: optional common.LevelType level;
}

/** Community serves as community for PGP purposes */
struct Community {
    1: required i32 top;
    2: required i32 bottom;
}

/** Neighbor structure */
struct Neighbor {
    1: required common.SystemIDType originator;
    2: required common.LinkIDType remote_id;
}

/** Capabilities the node supports */
struct NodeCapabilities {
    /** can this node participate in flood reduction,
        only relevant at level > 0 */
    1: optional bool flood_reduction =
        common.flood_reduction_default;
    /** does this node restrict itself to be leaf only (in ZTP) and
        does it support leaf-2-leaf procedures */
    2: optional common.LeafIndications leaf_indications;
}

/** RIFT LIE packet

    @note this node's level is already included on the packet header */
struct LIEPacket {
    /** optional node or adjacency name */
    1: optional string name;
    /** local link ID */
    2: required common.LinkIDType local_id;
    /** UDP port to which we can receive flooded TIEs */
    3: required common.UDPPortType flood_port =
        common.default_tie_udp_flood_port;
    /** layer 3 MTU */
    4: optional common.MTUSizeType link_mtu_size =
```

```

        common.default_mtu_size;
    /** this will reflect the neighbor once received to provid
        3-way connectivity */
    5: optional Neighbor                neighbor;
    6: optional common.PodType          pod = common.default_pod;
    /** optional nonce used for security computations */
    7: optional common.NonceType        nonce;
    /** optional node capabilities shown in the LIE. The capabilities
        MUST match the capabilities shown in the Node TIEs, otherwise
        the behavior is unspecified. A node detecting the mismatch
        SHOULD generate according error.
    */
    8: optional NodeCapabilities        capabilities;
    /** required holdtime of the adjacency, i.e. how much time
        MUST expire without LIE for the adjacency to drop
    */
    9: required common.HoldTimeInSecType holdtime =
        common.default_holdtime;
    /** indicates that the level on the LIE MUST NOT be used
        to derive a ZTP level by the receiving node. */
    10: optional bool                  not_a_ztp_offer =
        common.default_not_a_ztp_offer;
}

/** LinkID pair describes one of parallel links between two nodes */
struct LinkIDPair {
    /** node-wide unique value for the local link */
    1: required common.LinkIDType      local_id;
    /** received remote link ID for this link */
    2: required common.LinkIDType      remote_id;
    /** more properties of the link can go in here */
}

/** ID of a TIE

    @note: TIEID space is a total order achieved by comparing the elements
           in sequence defined and comparing each value as an
           unsigned integer of according length
    */
struct TIEID {
    /** indicates direction of the TIE */
    1: required common.TieDirectionType direction;
    /** indicates originator of the TIE */
    2: required common.SystemIDType      originator;
    3: required common.TIETypeType       tietype;
    4: required common.TIENrType         tie_nr;
}

```

```
/** Header of a TIE */
struct TIEHeader {
    2: required TIEID                tieid;
    3: required common.SeqNrType      seq_nr;
    /** lifetime expires down to 0 just like in ISIS */
    4: required common.LifeTimeInSecType lifetime;
}

/** A sorted TIDE packet, if unsorted, behavior is undefined */
struct TIDEPacket {
    /** all 00s marks starts */
    1: required TIEID                start_range;
    /** all FFs mark end */
    2: required TIEID                end_range;
    /** _sorted_ list of headers */
    3: required list<TIEHeader> headers;
}

/** A TIRE packet */
struct TIREPacket {
    1: required set<TIEHeader> headers;
}

/** Neighbor of a node */
struct NodeNeighborsTIEElement {
    2: required common.LevelType      level;
    /** Cost to neighbor.

        @note: All parallel links to same node
        incur same cost, in case the neighbor has multiple
        parallel links at different cost, the largest distance
        (highest numerical value) MUST be advertised
        @note: any neighbor with cost <= 0 MUST be ignored in computations */
    3: optional common.MetricType      cost = common.default_distance;
    /** can carry description of multiple parallel links in a TIE */
    4: optional set<LinkIDPair>        link_ids;

    /** total bandwidth to neighbor, this will be normally sum of the
     *   bandwidths of all the parallel links.
     */
    5: optional common.BandwidthInMegaBitsType bandwidth =
        common.default_bandwidth;
}

/** Flags the node sets */
struct NodeFlags {
    /** node is in overload, do not transit traffic through it */
    1: optional bool                  overload = common.overload_default;
```

```

}

/** Description of a node.

    It may occur multiple times in different TIEs but if either
    * capabilities values do not match or
    * flags values do not match or
    * neighbors repeat with different values or
    * visible in same level/having partition upper do not match
    the behavior is undefined and a warning SHOULD be generated.
    Neighbors can be distributed across multiple TIEs however if
    the sets are disjoint.

    @note: observe that absence of fields implies defined defaults
*/
struct NodeTIEElement {
    1: required common.LevelType          level;
    /** if neighbor systemID repeats in other node TIEs of same node
        the behavior is undefined. Equivalent to |A_(n,s)(N) in spec. */
    2: required map<common.SystemIDType,
        NodeNeighborsTIEElement>         neighbors;
    3: optional NodeCapabilities           capabilities;
    4: optional NodeFlags                  flags;
    /** optional node name for easier operations */
    5: optional string                     name;

    /** Nodes seen an the same level through reflection through nodes
        having backlink to both nodes. They are equivalent to |V(N) in
        future specifications. Ignored in Node S-TIEs if present.
    */
    6: optional set<common.SystemIDType>   visible_in_same_level
        = common.empty_set_of_nodeids;
    /** Non-overloaded nodes in |V seen as attached to another north
        * level partition due to the fact that some nodes in its |V have
        * adjacencies to higher level nodes that this node doesn't see.
        * This may be used in the computation at higher levels to prevent
        * blackholing. Ignored in Node S-TIEs if present.
        * Equivalent to |PUL(N) in spec. */
    7: optional set<common.SystemIDType>   same_level_unknown_north_partitions
        = common.empty_set_of_nodeids;
}

struct PrefixAttributes {
    /** Observe that in default metric case the node is supposed to advertise
        * metric calculated from comparison of bandwidths at all nodes at its
        * level. */
    2: required common.MetricType          metric = common.default_distance;
}

```

```

/** multiple prefixes */
struct PrefixTIEElement {
    /** prefixes with the associated attributes.
        if the same prefix repeats in multiple TIEs of same node
        behavior is unspecified */
    1: required map<common.IPPrefixType, PrefixAttributes> prefixes;
}

/** keys with their values */
struct KeyValueTIEElement {
    /** if the same key repeats in multiple TIEs of same node
        or with different values, behavior is unspecified */
    1: required map<common.KeyIDType, string> keyvalues;
}

/** single element in a TIE. enum common.TIETimeType
    in TIEID indicates which elements MUST be present
    in the TIEElement. In case of mismatch the unexpected
    elements MUST be ignored.
    */
union TIEElement {
    /** in case of enum common.TIETimeType.NodeTIETimeType */
    1: optional NodeTIEElement node;
    /** in case of enum common.TIETimeType.PrefixTIETimeType */
    2: optional PrefixTIEElement prefixes;
    /** transitive prefixes (always southbound) which SHOULD be propagated
        * southwards towards lower levels to heal
        * pathological upper level partitioning, otherwise
        * blackholes may occur. MUST NOT be advertised within a North TIE.
        */
    3: optional PrefixTIEElement transitive_prefixes;
    4: optional KeyValueTIEElement keyvalues;
    /** @todo: policy guided prefixes */
}

/** @todo: flood header separately in UDP to allow caching to TIEs
    while changing lifetime?
    */
struct TIEPacket {
    1: required TIEHeader header;
    2: required TIEElement element;
}

union PacketContent {
    1: optional LIEPacket lie;
    2: optional TIDEPacket tide;
    3: optional TIREPacket tire;
    4: optional TIEPacket tie;
}

```

```
}

/** protocol packet structure */
struct ProtocolPacket {
    1: required PacketHeader header;
    2: required PacketContent content;
}
```

9. IANA Considerations

This specification will request at an opportune time multiple registry points to exchange protocol packets in a standardized way, amongst them multicast address assignments and standard port numbers. The schema itself defines many values and codepoints which can be considered registries themselves.

10. Acknowledgments

Many thanks to Naiming Shen for some of the early discussions around the topic of using IGPs for routing in topologies related to Clos. Russ White to be especially acknowledged for the key conversation on epistemology that allowed to tie current asynchronous distributed systems theory results to a modern protocol design presented here. Adrian Farrel, Joel Halpern and Jeffrey Zhang provided thoughtful comments that improved the readability of the document and found good amount of corners where the light failed to shine. Kris Price was first to mention single router, single arm default considerations. Jeff Tantsura helped out with some initial thoughts on BFD interactions while Jeff Haas corrected several misconceptions about BFD's finer points. Artur Makutunowicz pointed out many possible improvements and acted as sounding board in regard to modern protocol implementation techniques RIFT is exploring. Barak Gafni formalized first time clearly the problem of partitioned spine on a (clean) napkin in Singapore.

11. References

11.1. Normative References

- [ISO10589]
ISO "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.

- [RFC1142] Oran, D., Ed., "OSI IS-IS Intra-domain Routing Protocol", RFC 1142, DOI 10.17487/RFC1142, February 1990, <<https://www.rfc-editor.org/info/rfc1142>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2328] Moy, J., "OSPF Version 2", STD 54, RFC 2328, DOI 10.17487/RFC2328, April 1998, <<https://www.rfc-editor.org/info/rfc2328>>.
- [RFC2365] Meyer, D., "Administratively Scoped IP Multicast", BCP 23, RFC 2365, DOI 10.17487/RFC2365, July 1998, <<https://www.rfc-editor.org/info/rfc2365>>.
- [RFC4271] Rekhter, Y., Ed., Li, T., Ed., and S. Hares, Ed., "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, DOI 10.17487/RFC4271, January 2006, <<https://www.rfc-editor.org/info/rfc4271>>.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <<https://www.rfc-editor.org/info/rfc4291>>.
- [RFC4655] Farrel, A., Vasseur, J., and J. Ash, "A Path Computation Element (PCE)-Based Architecture", RFC 4655, DOI 10.17487/RFC4655, August 2006, <<https://www.rfc-editor.org/info/rfc4655>>.
- [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, <<https://www.rfc-editor.org/info/rfc5120>>.
- [RFC5303] Katz, D., Saluja, R., and D. Eastlake 3rd, "Three-Way Handshake for IS-IS Point-to-Point Adjacencies", RFC 5303, DOI 10.17487/RFC5303, October 2008, <<https://www.rfc-editor.org/info/rfc5303>>.
- [RFC5709] Bhatia, M., Manral, V., Fanto, M., White, R., Barnes, M., Li, T., and R. Atkinson, "OSPFv2 HMAC-SHA Cryptographic Authentication", RFC 5709, DOI 10.17487/RFC5709, October 2009, <<https://www.rfc-editor.org/info/rfc5709>>.

- [RFC5881] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD) for IPv4 and IPv6 (Single Hop)", RFC 5881, DOI 10.17487/RFC5881, June 2010, <<https://www.rfc-editor.org/info/rfc5881>>.
- [RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <<https://www.rfc-editor.org/info/rfc6234>>.
- [RFC6822] Previdi, S., Ed., Ginsberg, L., Shand, M., Roy, A., and D. Ward, "IS-IS Multi-Instance", RFC 6822, DOI 10.17487/RFC6822, December 2012, <<https://www.rfc-editor.org/info/rfc6822>>.
- [RFC7855] Previdi, S., Ed., Filsfils, C., Ed., Decraene, B., Litkowski, S., Horneffer, M., and R. Shakir, "Source Packet Routing in Networking (SPRING) Problem Statement and Requirements", RFC 7855, DOI 10.17487/RFC7855, May 2016, <<https://www.rfc-editor.org/info/rfc7855>>.
- [RFC7938] Lapukhov, P., Premji, A., and J. Mitchell, Ed., "Use of BGP for Routing in Large-Scale Data Centers", RFC 7938, DOI 10.17487/RFC7938, August 2016, <<https://www.rfc-editor.org/info/rfc7938>>.
- [RFC7987] Ginsberg, L., Wells, P., Decraene, B., Przygienda, T., and H. Gredler, "IS-IS Minimum Remaining Lifetime", RFC 7987, DOI 10.17487/RFC7987, October 2016, <<https://www.rfc-editor.org/info/rfc7987>>.

11.2. Informative References

- [CLOS] Yuan, X., "On Nonblocking Folded-Clos Networks in Computer Communication Environments", IEEE International Parallel & Distributed Processing Symposium, 2011.
- [DIJKSTRA] Dijkstra, E., "A Note on Two Problems in Connexion with Graphs", Journal Numer. Math. , 1959.
- [DYNAMO] De Candia et al., G., "Dynamo: amazon's highly available key-value store", ACM SIGOPS symposium on Operating systems principles (SOSP '07), 2007.
- [EPPSTEIN] Eppstein, D., "Finding the k-Shortest Paths", 1997.

- [FATTREE] Leiserson, C., "Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing", 1985.
- [I-D.ietf-spring-segment-routing]
Filsfils, C., Previdi, S., Ginsberg, L., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", draft-ietf-spring-segment-routing-15 (work in progress), January 2018.
- [MAKSIC2013]
Maksic et al., N., "Improving Utilization of Data Center Networks", IEEE Communications Magazine, Nov 2013.
- [PROTOBUF]
Google, Inc., "Protocol Buffers", <https://developers.google.com/protocol-buffers>.
- [QUIC] Iyengar et al., J., "QUIC: A UDP-Based Multiplexed and Secure Transport", 2016.
- [VAHDAT08]
Al-Fares, M., Loukissas, A., and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture", SIGCOMM , 2008.

Authors' Addresses

Tony Przygienda (editor)
Juniper Networks
1194 N. Mathilda Ave
Sunnyvale, CA 94089
US

Email: prz@juniper.net

Alankar Sharma
Comcast
1800 Bishops Gate Blvd
Mount Laurel, NJ 08054
US

Email: Alankar_Sharma@comcast.com

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

Email: akatlas@juniper.net

John Drake
Juniper Networks
1194 N. Mathilda Ave
Sunnyvale, CA 94089
US

Email: jdrake@juniper.net

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 9, 2019

R. White, Ed.
S. Zandi, Ed.
LinkedIn
November 5, 2018

IS-IS Support for Openfabric
draft-white-openfabric-07

Abstract

Spine and leaf topologies are widely used in hyperscale and cloud scale networks. In most of these networks, configuration is automated, but difficult, and topology information is extracted through broad based connections. Policy is often integrated into the control plane, as well, making configuration, management, and troubleshooting difficult. Openfabric is an adaptation of an existing, widely deployed link state protocol, Intermediate System to Intermediate System (IS-IS) that is designed to:

- o Provide a full view of the topology from a single point in the network to simplify operations
- o Minimize configuration of each Intermediate System (IS) (also called a router or switch) in the network
- o Optimize the operation of IS-IS within a spine and leaf fabric to enable scaling

This document begins with an overview of openfabric, including a description of what may be removed from IS-IS to enable scaling. The document then describes an optimized adjacency formation process; an optimized flooding scheme; some thoughts on the operation of openfabric, metrics, and aggregation; and finally a description of the changes to the IS-IS protocol required for openfabric.

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 May 9, 2019.

Copyright Notice

Copyright (c) 2018 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. Goals	3
1.2. Contributors	3
1.3. Simplification	3
1.4. Additions and Requirements	4
1.5. Sample Network	4
2. Modified Adjacency Formation	6
2.1. Level 2 Adjacencies Only	6
2.2. Point-to-point Adjacencies	6
2.3. Three Way Handshake Support	7
2.4. Adjacency Formation Optimization	7
3. Advertisement of Reachability Information	8
4. Determining and Advertising Location on the Fabric	9
5. Flooding Optimization	10
5.1. Flooding Failures	11
6. Other Optimizations	12
6.1. Transit Link Reachability	12
6.2. Transiting T0 Intermediate Systems	12
7. Openfabric and Route Aggregation	13
8. Security Considerations	13
9. References	13
9.1. Normative References	13
9.2. Informative References	15
Appendix A. Flooding Optimization Operation	17
Appendix B. Fabric Location Calculation	19
Authors' Addresses	20

1. Introduction

1.1. Goals

Spine and leaf fabrics are often used in large scale data centers; in this application, they are commonly called a fabric because of their regular structure and predictable forwarding and convergence properties. This document describes modifications to the IS-IS protocol to enable it to run efficiently on a large scale spine and leaf fabric, openfabric. The goals of this control plane are:

- o Provide a full view of the topology from a single point in the network to simplify operations
- o Minimize configuration of each IS in the network
- o Optimize the operation of IS-IS within a spine and leaf fabric to enable scaling

1.2. Contributors

The following people have contributed to this draft: Nikos Triantafyllis (reflected flooding optimization), Ivan Pepelnjak (fabric locality calculation modifications), Christian Franke (fabric locality calculation modification), Hannes Gredler (do not reflood optimizations), Les Ginsberg (capabilities encoding, circuit local reflooding), Naiming Shen (capabilities encoding, circuit local reflooding), Uma Chunduri (failure mode suggestions, flooding), Nick Russo, and Rodny Molina.

See [RFC5449], [RFC5614], and [RFC7182] for similar solutions in the Mobile Ad Hoc Networking (MANET) solution space.

1.3. Simplification

In building any scalable system, it is often best to begin by removing what is not needed. In this spirit, openfabric implementations MAY remove the following from IS-IS:

- o External metrics. There is no need for external metrics in large scale spine and leaf fabrics; it is assumed that metrics will be properly configured by the operator to account for the correct order of route preference at any route redistribution point.
- o Tags and traffic engineering processing. Openfabric is only designed to provide topology and reachability information. It is not designed to provide for traffic engineering, route preference through tags, or other policy mechanisms. It is assumed that all

routing policy will be provided through an overlay system which communicates directly with each IS in the fabric, such as PCEP [RFC5440] or I2RS [RFC7921]. Traffic engineering is assumed to be provided through Segment Routing (SR) [I-D.ietf-spring-segment-routing].

1.4. Additions and Requirements

To create a scalable link state fabric, openfabric includes the following:

- o A slightly modified adjacency formation process.
- o Mechanisms for determining which tier within a spine and leaf fabric in which the IS is located.
- o A mechanism that reduces flooding to the minimum possible, while still ensuring complete database synchronization among the intermediate systems within the fabric.

Three general requirements are placed here; more specific requirements are considered in the following sections. Openfabric implementations:

- o MUST support [RFC5301] and enable hostname advertisement by default if a hostname is configured on the intermediate system.
- o SHOULD support [RFC6232], purge originator identification for IS-IS.
- o MUST NOT be mixed with standard IS-IS implementations in operational deployments. Openfabric and standard IS-IS implementations SHOULD be treated as two separate protocols.

1.5. Sample Network

The following spine and leaf fabric will be used to describe these modifications.

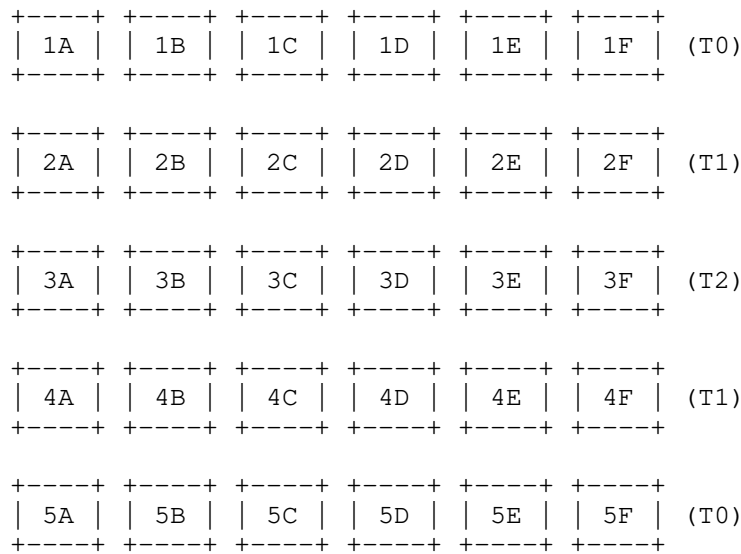


Figure 1

To reduce confusion (spine and leaf fabrics are difficult to draw in plain text art), this diagram does not contain the connections between devices. The reader should assume that each device in a given layer is connected to every device in the layer above it. For instance:

- o 5A is connected to 4A, 4B, 4C, 4D, 4E, and 4F
- o 5B is connected to 4A, 4B, 4C, 4D, 4E, and 4F
- o 4A is connected to 3A, 3B, 3C, 3D, 3E, 3F, 5A, 5B, 5C, 5D, 5E, and 5F
- o 4B is connected to 3A, 3B, 3C, 3D, 3E, 3F, 5A, 5B, 5C, 5D, 5E, and 5F
- o etc.

The tiers or stages of the fabric are also marked for easier reference. T0 is assumed to be connected to application servers, or rather they are Top of Rack (ToR) intermediate systems. The remaining tiers, T1 and T2, are connected only to the fabric itself. Note there are no "cross links," or "east west" links in the illustrated fabric. The fabric locality detection mechanism described here will not work if there are cross links running east/

west through the fabric. Locality detection may be possible in such a fabric; this is an area for further study.

2. Modified Adjacency Formation

Because Openfabric operates in a tightly controlled data center environment, various modifications can be made to the IS-IS neighbor formation process to increase efficiency and simplify the protocol. Specifically, Openfabric implementations SHOULD support [RFC3719], section 4, hello padding for IS-IS. Variable hello padding SHOULD NOT be used, as data center fabrics are built using high speed links on which padded hellos will have little performance impact. Further modifications to the neighbor formation process are considered in the following sections.

2.1. Level 2 Adjacencies Only

Openfabric is designed to work in a single flooding domain over a single data center fabric at the scale of thousands of routers with hundreds of thousands of routes (so a moderate scale in router and route count terms). Because of the way Openfabric optimizes operation in this environment, it is not necessary nor desirable to build multiple flooding domains. For instance, the flooding optimizations described later in this document require a full view of the topology, as does any proposed overlay to inject policy into the forwarding plane. In light of this, the following changes SHOULD BE to IS-IS implementations to support Openfabric:

- o IIH PDU 17 (level 2 point-to-point circuit hello) should be the only IIH PDU type transmitted (see section 9.7 of ISO 10589)
- o In IIH PDU 17 (level 2 point-to-point circuit hello), the Circuit Type field should be set to 2 (see section 9.7 of ISO 10589)
- o Support for IIH PDU 15 (level 1 broadcast hello) should be removed (see section 9.5 of ISO 10589)
- o Support for IIH PDU 16 (level 2 broadcast hello) should be removed (see section 9.6 of ISO 10589)

2.2. Point-to-point Adjacencies

Data center network fabrics only contain point-to-point links; because of this, there is no reason to support any broadcast link types, nor to support the Designated Intermediate System processing, including pseudonode creation. In light of this, processing related to sections 7.2.3 (broadcast networks), 7.3.8 (generation of level 1 pseudonode LSPs), 7.3.10 (generation of level 2 pseudonode LSPs), and

section 8.4.5 (LAN designated intermediate systems) in [ISO10589] SHOULD BE removed.

2.3. Three Way Handshake Support

It is important that two way connectivity be established before synchronizing the link state database, or routing through a link in a data center fabric. To reject optical failures that cause a one way connection between two routers, fabricDC must support the three way handshake mechanism described in [RFC5303].

2.4. Adjacency Formation Optimization

While adjacency formation is not considered particularly burdensome in IS-IS, it may still be useful to reduce the amount of state transferred across the network when connecting a new IS to the fabric. In its simplest form, the process is:

- o An IS connected to the fabric will send hellos on all links.
- o The IS will only complete the three-way handshake with one newly discovered neighbor; this would normally be the first neighbor which sends the newly connected intermediate system's ID back in the three-way handshake process.
- o The IS will complete its database exchange with this one newly adjacent neighbor.
- o Once this process is completed, the IS will continue processing the remaining neighbors as normal.
- o If synchronization is not achieved within twice the dead timer on the local interface, the newly connected IS will repeat this process with the second neighbor with which it forms a three-way adjacency.

This process allows each IS newly added to the fabric to exchange a full table once; a very minimal amount of information will be transferred with the remaining neighbors to reach full synchronization.

Any such optimization is bound to present a tradeoff between several factors; the mechanism described here increases the amount of time required to form adjacencies slightly in order to reduce the total state carried across the network. An alternative mechanism could provide a better balance of the amount of information carried across the network for initial synchronization and the time required to synchronize a new IS. For instance, an IS could choose to

synchronize its database with two or three adjacent intermediate systems, which could speed the synchronization process up at the cost of carrying additional data on the network. A locally determined balance between the speed of synchronization and the amount of data carried on the network can be achieved by adjusting the number of adjacent intermediate systems the newly attached IS synchronizes with.

3. Advertisement of Reachability Information

IS-IS describes the topology in two different sets of TLVs; the first describes the set of neighbors connected to an IS, the second describes the set of reachable destination connected to an IS. There are two different forms of both of these descriptions, one of which carries what are widely called narrow metrics, the other of which carries what are widely called wide metrics. In a tightly controlled data center fabric implementation, such as the ones Openfabric is designed to support, no IS that supports narrow metrics will ever be deployed or supported; hence there is no reason to support any metric type other than wide metrics.

- o The Level 2 Link State PDU (type 20 in section 9.9 of [ISO10589]) and the scoped flooding PDU (type 10 in section 3.1 of [RFC7356]) SHOULD BE the only PDU types used to carry link state information in a Openfabric implementation
- o Processing related to the Level 1 Link State PDU (type 18) MAY BE removed from Openfabric implementations (see section 9.8 of [ISO10589])
- o Neighbor reachability MUST BE carried in TLV type 22 (see section 3 of [RFC5305])
- o IPv4 reachability SHOULD BE carried in TLV type 135 (see section 4 of [RFC5305]), or TLV type 235 for multitopology implementations (see [RFC5120])
- o IPv6 reachability SHOULD BE carried in TLV type 236 (see [RFC5308]), or TLV type 237 for multitopology implemenations (see [RFC5120])
- o Processing related to the neighbor reachability TLV (type 2, see sections 9.8 and 9.9 of [ISO10589]) SHOULD BE removed
- o Processing related to the narrow metric IP reachability TLV (types 128 and 130) SHOULD BE removed

Further, if segment routing support is desired, Openfabric MAY support the Prefix Segment Identifier sub-TLV and other TLVs as required in [I-D.ietf-isis-segment-routing-extensions].

4. Determining and Advertising Location on the Fabric

The tier to which a IS is connected is useful to enable autoconfiguration of intermediate systems connected to the fabric and to reduce flooding. Once the tier of an intermediate system within the fabric has been determined, it MUST be advertised using the 4 bit Tier field described in section 3.3 of [I-D.shen-isis-spine-leaf-ext]. This section describes a method of calculating the tier number, assuming the tier numbers rise in value from the edge of the fabric.

This method begins with two of the T0 intermediate systems advertising their location in the fabric. This information can either be obtained through:

- o Two T0 intermediate systems are manually configured to advertise 0x00 in their IS reachability tier sub-TLV, indicating they are at the edge of the fabric (a ToR IS).
- o The T0 intermediate systems detect they are T0 through the presence connected hosts (i.e. through a request for address assignment or some other means). If such detection is used, and the IS determines it is located at T0, it should advertise 0x00 in its IS reachability tier sub-TLV.

If the first method is used, the two T0 routers MUST be "maximally separated" on the fabric. They must be a maximal number of hops apart, or rather they MUST NOT be connected to the same T1 device as their "upstream" towards the superspines in a 5 ary fabric.

The second method above SHOULD be used with care, as it may not be secure, and it may not work in all data center environments. For instance, if a host is mistakenly (or intentionally, as a form of attack) attached to a spine IS, or a request for address assignment is transmitted to a spine IS during the bootup phase of the device or fabric, it is possible to cause a spine IS to advertise itself as a T0. Unless the autodetection of the T0 devices is secured, the manual mechanism SHOULD BE used (configuring at least one T0 device manually).

Given the correct configuration of two T0 devices, maximally spaced on the fabric, the remaining intermediate systems calculate their tier number as follows:

- o The local IS calculates an SPT (using SPF) setting the cost of every link to 1; this effectively calculates a topology only view of the network, without considering any configured link costs
- o Ensure that at least two T0 are in the calculated SPT; otherwise abort
- o Find the furthest T0; call this node A and set LD to the cost; the "furthest T0" is the T0 with the largest metric, or the furthest distance from the local calculating node
- o Calculate an SPT (using SPF) from the perspective of A (above) setting the cost of every link to 1
- o Find the furthest IS in A's SPT; call this node B and set RD to the cost from A to B
- o Calculate the tier number of the local IS by subtracting LD from RD

In the example network, assume 5A and 1C are manually configured as a T0, and are advertising their tier numbers. From here:

- o From 1A the path to 5A is 4 hops; this is LD
- o Run SPF from the perspective of 5A with all link metrics set to 1
- o From 5A the path length to 1C is 4; this is RD
- o $RD - LD$ is 0 at 1A, so 1A is T0, or a ToR

This process will work for any spine and leaf fabric without "cross links."

5. Flooding Optimization

Flooding is perhaps the most challenging scaling issue for a link state protocol running on a dense, large scale fabric. To reduce the flooding of link state information in the form of Link State Protocol Data Units (LSPs), Openfabric takes advantage of information already available in the link state protocol, the list of the local intermediate system's neighbor's neighbors, and the fabric locality computed above. The following tables are required to compute a set of reflooders:

- o Neighbor List (NL) list: The set of neighbors

- o Neighbor's Neighbors (NN) list: The set of neighbor's neighbors; this can be calculated by running SPF truncated to two hops
- o Do Not Reflood (DNR) list: The set of neighbors who should have LSPs (or fragments) who should not reflood LSPs
- o Reflood (RF) list: The set of neighbors who should flood LSPs (or fragments) to their adjacent neighbors to ensure synchronization

NL is set to contain all neighbors, and sorted deterministically (for instance, from the highest IS identifier to the lowest). All intermediate systems within a single fabric SHOULD use the same mechanism for sorting the NL list. NN is set to contain all neighbor's neighbors, or all intermediate systems that are two hops away, as determined by performing a truncated SPF. The DNR and RF tables are initially empty. To begin, the following steps are taken to reduce the size of NN and NL:

- o Move any IS in NL with its tier (or fabric location) set to T0 to DNR
- o Remove all intermediate systems from NL and NN that in the shortest path to the IS that originated the LSP

Then, for every IS in NL:

- o If the current entry in NL is connected to any entries in NN:
 - * Move the IS to RF
 - * Remove the intermediate systems connected to the IS from NN
- o Else move the IS to DNR

The calculation terminates when the NL is empty.

When flooding, LSPs transmitted to adjacent neighbors on the RF list will be transmitted normally. Adjacent intermediate systems on this list will reflood received LSPs into the next stage of the topology, ensuring database synchronization. LSPs transmitted to adjacent neighbors on the DNR list, however, MUST be transmitted using a circuit scope PDU as described in [RFC7356].

5.1. Flooding Failures

It is possible in some failure modes for flooding to be incomplete because of the flooding optimizations outlined. Specifically, if a reflooder fails, or is somehow disconnected from all the links across

which it should be reflooding, it is possible an LSP is only partially flooded through the fabric. To prevent such situations, any IS receiving an LSP transmitted using DNR SHOULD:

- o Set a short timer; the default should be less than one second
- o When the timer expires, send a Complete Sequence Number Packet (CSNP) to all neighbors
- o Process any Partial Sequence Number Packets (PSNPs) as required to resynchronize
- o If a resynchronization is required, notify the network operator through a network management system

6. Other Optimizations

6.1. Transit Link Reachability

In order to reduce the amount of control plane state carried on large scale spine and leaf fabrics, openfabric implementations SHOULD NOT advertise reachability for transit links. These links MAY remain unnumbered, as IS-IS does not require layer 3 IP addresses to operate. Each IS SHOULD be configured with a single loopback address, which is assigned an IPv6 address, to provide reachability to intermediate systems which make up the fabric.

[RFC3277] SHOULD be supported on devices supporting openfabric with unnumbered interface in order to support traceability and network management.

6.2. Transiting T0 Intermediate Systems

In data center fabrics, ToR intermediate systems SHOULD NOT be used to transit between two T1 (or above) spine intermediate systems. The simplest way to prevent this is to set the overload bit [RFC3277] for all the LSPs originated from T0 intermediate systems. However, this solution would have the unfortunate side effect of causing all reachability beyond any T0 IS to have the same metric, and many implementations treat a set overload bit as a metric of 0xFFFF in calculating the Shortest Path Tree (SPT). This document proposes an alternate solution which preserves the leaf node metric, while still avoiding transiting T0 intermediate systems.

Specifically, all T0 intermediate systems SHOULD advertise their metric to reach any T1 adjacent neighbor with a cost of 0XFFE. T1 intermediate systems, on the other hand, will advertise T0 intermediate systems with the actual interface cost used to reach the

T0 IS. Hence, links connecting T0 and T1 intermediate systems will be advertised with an asymmetric cost that discourages transiting T0 intermediate systems, while leaving reachability to the destinations attached to T0 devices the same.

7. Openfabric and Route Aggregation

While schemes may be designed so reachability information can be aggregated in Openfabric deployments, this is not a recommended configuration.

8. Security Considerations

This document outlines modifications to the IS-IS protocol for operation on large scale data center fabrics. While it does add new TLVs, and some local processing changes, it does not add any new security vulnerabilities to the operation of IS-IS. However, openfabric implementations SHOULD implement IS-IS cryptographic authentication, as described in [RFC5304], and should enable other security measures in accordance with best common practices for the IS-IS protocol.

If T0 intermediate systems are auto-detected using information outside Openfabric, it is possible to attack the calculations used for flooding reduction and auto-configuration of intermediate systems. For instance, if a request for an address pool is used as an indicator of an attached host, and hence receiving such a request causes an intermediate system to advertise itself as T0, it is possible for an attacker (or a simple mistake) to cause auto-configuration to fail. Any such auto-detection mechanisms SHOULD BE secured using appropriate techniques, as described by any protocols or mechanisms used.

9. References

9.1. Normative References

[I-D.shen-isis-spine-leaf-ext]

Shen, N., Ginsberg, L., and S. Thyamagundalu, "IS-IS Routing for Spine-Leaf Topology", draft-shen-isis-spine-leaf-ext-07 (work in progress), October 2018.

- [ISO10589] 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.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, DOI 10.17487/RFC2629, June 1999, <<https://www.rfc-editor.org/info/rfc2629>>.
- [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, <<https://www.rfc-editor.org/info/rfc5120>>.
- [RFC5301] McPherson, D. and N. Shen, "Dynamic Hostname Exchange Mechanism for IS-IS", RFC 5301, DOI 10.17487/RFC5301, October 2008, <<https://www.rfc-editor.org/info/rfc5301>>.
- [RFC5303] Katz, D., Saluja, R., and D. Eastlake 3rd, "Three-Way Handshake for IS-IS Point-to-Point Adjacencies", RFC 5303, DOI 10.17487/RFC5303, October 2008, <<https://www.rfc-editor.org/info/rfc5303>>.
- [RFC5305] Li, T. and H. Smit, "IS-IS Extensions for Traffic Engineering", RFC 5305, DOI 10.17487/RFC5305, October 2008, <<https://www.rfc-editor.org/info/rfc5305>>.
- [RFC5308] Hopps, C., "Routing IPv6 with IS-IS", RFC 5308, DOI 10.17487/RFC5308, October 2008, <<https://www.rfc-editor.org/info/rfc5308>>.
- [RFC5309] Shen, N., Ed. and A. Zinin, Ed., "Point-to-Point Operation over LAN in Link State Routing Protocols", RFC 5309, DOI 10.17487/RFC5309, October 2008, <<https://www.rfc-editor.org/info/rfc5309>>.

- [RFC5311] McPherson, D., Ed., Ginsberg, L., Previdi, S., and M. Shand, "Simplified Extension of Link State PDU (LSP) Space for IS-IS", RFC 5311, DOI 10.17487/RFC5311, February 2009, <<https://www.rfc-editor.org/info/rfc5311>>.
- [RFC5316] Chen, M., Zhang, R., and X. Duan, "ISIS Extensions in Support of Inter-Autonomous System (AS) MPLS and GMPLS Traffic Engineering", RFC 5316, DOI 10.17487/RFC5316, December 2008, <<https://www.rfc-editor.org/info/rfc5316>>.
- [RFC7356] Ginsberg, L., Previdi, S., and Y. Yang, "IS-IS Flooding Scope Link State PDUs (LSPs)", RFC 7356, DOI 10.17487/RFC7356, September 2014, <<https://www.rfc-editor.org/info/rfc7356>>.
- [RFC7981] Ginsberg, L., Previdi, S., and M. Chen, "IS-IS Extensions for Advertising Router Information", RFC 7981, DOI 10.17487/RFC7981, October 2016, <<https://www.rfc-editor.org/info/rfc7981>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

9.2. Informative References

- [I-D.ietf-isis-segment-routing-extensions]
Previdi, S., Ginsberg, L., Filsfils, C., Bashandy, A., Gredler, H., Litkowski, S., Decraene, B., and J. Tantsura, "IS-IS Extensions for Segment Routing", draft-ietf-isis-segment-routing-extensions-19 (work in progress), July 2018.
- [I-D.ietf-spring-segment-routing]
Filsfils, C., Previdi, S., Ginsberg, L., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", draft-ietf-spring-segment-routing-15 (work in progress), January 2018.
- [RFC3277] McPherson, D., "Intermediate System to Intermediate System (IS-IS) Transient Blackhole Avoidance", RFC 3277, DOI 10.17487/RFC3277, April 2002, <<https://www.rfc-editor.org/info/rfc3277>>.
- [RFC3719] Parker, J., Ed., "Recommendations for Interoperable Networks using Intermediate System to Intermediate System (IS-IS)", RFC 3719, DOI 10.17487/RFC3719, February 2004, <<https://www.rfc-editor.org/info/rfc3719>>.

- [RFC4271] Rekhter, Y., Ed., Li, T., Ed., and S. Hares, Ed., "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, DOI 10.17487/RFC4271, January 2006, <<https://www.rfc-editor.org/info/rfc4271>>.
- [RFC5304] Li, T. and R. Atkinson, "IS-IS Cryptographic Authentication", RFC 5304, DOI 10.17487/RFC5304, October 2008, <<https://www.rfc-editor.org/info/rfc5304>>.
- [RFC5440] Vasseur, JP., Ed. and JL. Le Roux, Ed., "Path Computation Element (PCE) Communication Protocol (PCEP)", RFC 5440, DOI 10.17487/RFC5440, March 2009, <<https://www.rfc-editor.org/info/rfc5440>>.
- [RFC5449] Baccelli, E., Jacquet, P., Nguyen, D., and T. Clausen, "OSPF Multipoint Relay (MPR) Extension for Ad Hoc Networks", RFC 5449, DOI 10.17487/RFC5449, February 2009, <<https://www.rfc-editor.org/info/rfc5449>>.
- [RFC5614] Ogier, R. and P. Spagnolo, "Mobile Ad Hoc Network (MANET) Extension of OSPF Using Connected Dominating Set (CDS) Flooding", RFC 5614, DOI 10.17487/RFC5614, August 2009, <<https://www.rfc-editor.org/info/rfc5614>>.
- [RFC5820] Roy, A., Ed. and M. Chandra, Ed., "Extensions to OSPF to Support Mobile Ad Hoc Networking", RFC 5820, DOI 10.17487/RFC5820, March 2010, <<https://www.rfc-editor.org/info/rfc5820>>.
- [RFC5837] Atlas, A., Ed., Bonica, R., Ed., Pignataro, C., Ed., Shen, N., and JR. Rivers, "Extending ICMP for Interface and Next-Hop Identification", RFC 5837, DOI 10.17487/RFC5837, April 2010, <<https://www.rfc-editor.org/info/rfc5837>>.
- [RFC6232] Wei, F., Qin, Y., Li, Z., Li, T., and J. Dong, "Purge Originator Identification TLV for IS-IS", RFC 6232, DOI 10.17487/RFC6232, May 2011, <<https://www.rfc-editor.org/info/rfc6232>>.
- [RFC7182] Herberg, U., Clausen, T., and C. Dearlove, "Integrity Check Value and Timestamp TLV Definitions for Mobile Ad Hoc Networks (MANETs)", RFC 7182, DOI 10.17487/RFC7182, April 2014, <<https://www.rfc-editor.org/info/rfc7182>>.
- [RFC7921] Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", RFC 7921, DOI 10.17487/RFC7921, June 2016, <<https://www.rfc-editor.org/info/rfc7921>>.

Appendix A. Flooding Optimization Operation

Recent testing has shown that flooding is largely a "non-issue" in terms of scaling when using high speed links connecting intermediate systems with reasonable processing power and memory. However, testing has also shown that flooding will impact convergence speed even in such environments, and flooding optimization has a major impact on the performance of a link state protocol in resource constrained environments. Some thoughts on flooding optimization in general, and the flooding optimization contained in this document, follow.

There are two general classes of flooding optimization available for link state protocols. The first class of optimization relies on a centralized service or server to gather the link state information and redistribute it back into the intermediate systems making up the fabric. Such solutions are attractive in many, but not all, environments; hence these systems compliment, rather than compete with, the system described here. Systems relying on a service or server necessarily also rely on connectivity to that service or server, either through an out-of-band network or connectivity through the fabric itself. Because of this, these mechanisms do not apply to all deployments; some deployments require underlying reachability regardless of connectivity to an outside service or server.

The second possibility is to create a fully distributed system that floods the minimal amount of information possible to every intermediate system. The system described in this draft is an example of such a system. Again, there are many ways to accomplish this goal, but simplicity is a primary goal of the system described in this draft.

The system described here divides the work into two different parts; forward and reverse optimization. The forward optimization begins by finding the set of intermediate systems two hops away from the flooding device, and choosing a subset of connected neighbors that will successfully reach this entire set of intermediate systems, as shown in the diagram below.

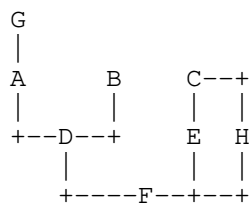


Figure 2

If F is flooding some piece of information, then it will find the entire set of intermediate systems within two hops by discovering its neighbors and their neighbors from the local LSDB. This will include A, B, C, D, and E--but not G. From this set, F can determine that D can reach A and B, while a single flood to either E or H will reach C. Hence F can flood to D and either E or H to reach C. F can choose to flood to D and E normally. Because H still needs to receive this new LSP (or fragment!), but does not need to reflood to C, F can send the LSP using link local signaling. In this case, H will receive and process the new LSP, but not reflood it.

Rather than carrying the information necessary through hello extensions, as is done in [RFC5820], the neighbors are allowed to complete initial synchronization, and then a truncated shortest path tree is built to determine the "two hop neighborhood." This has the advantage of using mechanisms already used in IS-IS, rather than adding new processes. The risk with this process is any LSPs flooded through the network before this initial calculation takes place will be suboptimal. This "two hop neighborhood" process has been used in OSPF deployments for a number of years, and has proven stable in practice.

Rather than setting a timer for reflooding, the implementation described here uses IS-IS' ability to describe the entire database using a CSNP to ensure flooding is successful. This adds some small amount of overhead, so there is some balance between optimal flooding and ensuring flooding is complete.

The reverse optimization is simpler. It relies on the observation that any intermediate system between the local IS and the origin of the LSP, other than in the case of floods removing an LSP from the shared LSDB, should have already received a copy of the LSP. For instance, if F originates an LSP in the figure above, and E refloods the LSP to C, C does not need to reflood back to F if F is on its shortest path tree towards F. It is obvious this is not a "perfect" optimization. A perfect optimization would block flooding back along a directed acyclic graph towards the originator. Using the SPT, however, is a quick way to reduce flooding without performing more calculations.

The combination of these two optimizations have been seen, in testing, to reduce the number of copies any IS receives from the tens to precisely one.

Appendix B. Fabric Location Calculation

Determining the location of a device in a symmetric topology is quite challenging. The authors of this draft worked through a number of possible solutions to this problem, each of which was found to either not work in some topology, or was found to be liable to unacceptable errors. For instance:

- o Method 1:

- * Caculate the maximum distance through the fabric, and the distance from one of those points to the local intermediate system
- * This works in a five stage Clos spine and leaf, but not in a three stage, nor in some other five stage spine and leaf fabrics, such as the common butterfly or Benes fabric

- o Method 2:

- * Manually mark one edge leaf node in the fabric as T0
- * Calculate maximum distance through the fabric from this point
- * Calculate local position based on this maximum distance the distance to the single marked device
- * This works in three and five stage Clod fabrics, but does not work from every location in other spine and leaf fabrics, such as the common butterfly or Benes fabric

In the end, marking two devices located as far from one another topologically as possible provides the anchor points necessary to calculate the total distance through the fabric, and then from those points to the location of the calculating device.

The information obtained in this way can also be combined with other forms of location calculation, such as whether a device requesting an address through some mechanism is attached to the local device, or other indications of fabric locality. It generally true that having more than one method to determine fabric location will be better than any single method to account for errors, failures, and other problems that can arise with any mechanism.

Authors' Addresses

Russ White (editor)
LinkedIn

Email: russ@riw.us

Shawn Zandi (editor)
LinkedIn

Email: szandi@linkedin.com