Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: January 4, 2018

J. Arkko
Ericsson
J. Tantsura
Futurewei, Future Networks

July 3, 2017

Low Latency Applications and the Internet Architecture
draft-arkko-arch-low-latency-01

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 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 4, 2018.

Table of Contents

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, across the stack. 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.

Across the stack there are also many other 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 uses 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. 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:

- Application architectures for globally connected but low-latency services.
- What are the issues with inter-domain Quality-of-Service mechanisms? Are there approaches that would offer progress on this field?
- 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.
- The emergence of cross-layer optimisations and how that affects the Internet architecture and its future evolution.
- Inter-organisatorial 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, 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-quic-transport]
Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-04 (work in progress), June 2017.

[I-D.ietf-sfc-nsh]
Quinn, P. and U. Elzur, "Network Service Header", draft-ietf-sfc-nsh-13 (work in progress), June 2017.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-20 (work in progress), April 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, <http://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, <http://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, <http://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, <http://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, <http://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, <http://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, <http://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: Standards Track
Expires: January 2018

A. Bashandy
C. Filsfils
Cisco Systems
Bruno Decraene
Stephane Litkowski
Orange
Pierre Francois
Individual Contributor

July 17, 2017

Topology Independent Fast Reroute using Segment Routing
draft-bashandy-rtgwg-segment-routing-ti-lfa-01

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.

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.

This Internet-Draft will expire on August 16, 2017.

Table of Contents

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 [5].

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 applies to traffic which traverses the PLR. Traffic which does NOT traverse the PLR remains unaffected.

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) 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. This allows for a protection path selection approach meeting operational needs rather than a topologically constrained one.

Using SR, there is no need to create state in the network in order to enforce an explicit FRR path. As a result, we can use optimized detour paths for each specific destination and for each type of failure without creating additional forwarding state. Also, the mode of protection (link, node, SRLG) is not constrained to be network wide or node wide, but can be managed on a per interface basis.

Building on such an easier forwarding environment, 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.

As the capacity of the post-convergence path is typically planned by the operator to support the post-convergence routing of the traffic for any expected failure, there is much less need for the operator to tune the decision among which protection path to choose. The protection path will automatically follow the natural backup path that would be used after local convergence. This also helps to reduce the amount of path changes and hence service transients: one transition (pre-convergence to post-convergence) instead of two (pre-convergence to FRR and then post-convergence).

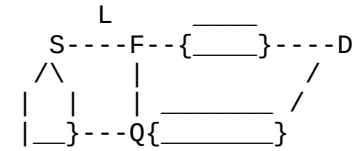

Figure 1 TI-LFA Protection

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, we describe how to compute protection lists that encode a loopfree post-convergence towards the destination, in Section 4.

Finally, we define the segment operations to be applied by the PLR to ensure consistency with the forwarding state of the repair node, in Section 5.

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 distance from node A to node B in SPT_old(A).

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

Similarly to [4], we rely on 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->X) for all S->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.

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.

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 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 [6].

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

As planned in the previous version of this document, 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}

We describe the protection behavior of S when

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}

We describe the protection behavior of S when

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.

6. Security Considerations

The techniques described in this document are 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.

7. IANA Considerations

No requirements for IANA

8. 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.

9. References

9.1. Normative References

9.2. 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] Bashandy, A., Filsfils, C., and Litkowski, S., " Loop avoidance using Segment Routing", draft-bashandy-rtgwg-segment-routing-uloop-00, (work in progress), May 2017

[6] 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

10. 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
pfrpfr@gmail.com

Ahmed Bashandy
Cisco Systems
170 West Tasman Dr, San Jose, CA 95134, USA
Email: bashandy@cisco.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

Network Working Group
Internet Draft
Intended status: Standards Track
Expires: January 2018

Ahmed Bashandy
Clarence Filsfils
Cisco Systems, Inc.
Stephane Litkowski
Orange
Pierre Francois
Individual Contributor

July 17, 2017

Loop avoidance using Segment Routing
draft-bashandy-rtgwg-segment-routing-uloop-01

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.

Table of Contents

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.

This document provides a mechanism leveraging Segment Routing to ensure loop-freeness during the IGP reconvergence process following a link-state change event.

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.

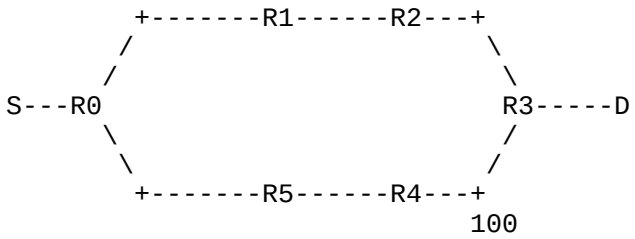

Figure 1 Illustrative scenario, failure of link R2-R3

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.

Using segment routing, a headend can enforce an explicit path without creating any state along the desired path. As a result, a converging node can enforce traffic on the post-convergence path in a loop-free manner, using a list of segments (typically short). We suggest that the converging node enforces its post-convergence path to the destination when applying this behavior to ease operation (predictability of path, less capacity planning issues ...); nodes converge over their new optimal path, but temporarily use an SR policy to ensure loop-freeness over that path.

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
Cisco Systems
170 West Tasman Dr, San Jose, CA 95134, USA
Email: bashandy@cisco.com

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

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

Pierre Francois
pfrpfr@gmail.com

Routing Area Working Group
Internet-Draft
Intended status: Informational
Expires: January 4, 2018

S. Bryant
J. Dong
Huawei

July 3, 2017

Enhanced Virtual Private Networks (VPN+)
draft-bryant-rtgwg-enhanced-vpn-00

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 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 4, 2018.

Copyright Notice

Table of Contents

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. The network slicing problem is described in [I-D.galis-netslices-revised-problem-statement] and the network slicing architecture is described in [I-D.geng-netslices-architecture]. A study of the new work needed in the IETF to address the gap between the requirements and existing IETF protocols is discussed in [I-D.qiang-netslices-gap-analysis].

Setting aside the details of the life-cycle management of a network slice instance (NSI), the underpinning technology in the transport network is a type of virtual network which provides the client with dedicated (private) networking, computing and storage resources drawn from a shared pool. The tenant of the NSI 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 the network slice, e.g. to customize the service paths in the network slice.

These new network layer properties are of interest as part of a network slicing solution, as a precursor to the full roll-out of network slicing, and in their own right. 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, and can achieve a greater isolation and lower latency than traditional VPN.

In this draft we identify the new and modified components that need to be provided in the network layer and their associated control and monitoring of an enhanced VPN. Specifically we are concerned with the technology needed to be provided by the enhanced VPN underlay, the enhanced VPN data-plane and the necessary protocols in both the underlay and the overlay of enhanced VPN. It is likely that these enhanced VPNs will be used to create network slices with different isolation requirements. Different service types, e.g. emergency services, enterprise service and broadband services etc. may be partitioned into different "hard" slices according to the isolation requirement. These "hard" slices might then be used to carry one or multiple VPNs. VPNs on such a hard slice may be only partially isolated (so called "soft" slices).

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 VPNs

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 and soft 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. An 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. A network slice that experiences only soft isolation is said to be soft sliced, and a network slice that has hard isolation is said to be hard sliced.

Although these isolation requirements are triggered by the needs of network slicing in support of 5G networks, they have general utility.

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. 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.

3.2. Guaranteed Performance

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 combine with the latency requirement.

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 IP/MPLS 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 and not provided through different virtual networks. The need for guaranteed maximum delay variation as a general requirement is for further study.

A possible mechanism to address these guarantees is to provide enhancement to the underlay network through technologies such as Flexible Ethernet [FLEXE].

3.3. 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.

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

4. Components of VPN+

4.1. Use of Segment Routing Constructs

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

Segment Routing [I-D.ietf-spring-segment-routing] is a method that prepends instructions to packets at entry and possibly at various points as it passes though 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 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. queues, processors, links, services etc. New instructions can also 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 enqueuing and dequeuing.

With SR, a path is 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.

The SIDs may be used to specify both network paths, or service functions as described in [I-D.xu-mpls-unified-source-routing-instruction].

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.

4.2. Latency Support

The IETF has ongoing work on support for a latency ceiling [I-D.ietf-detnet-architecture]. The provision of a latency ceiling is a requirement of the application seeking the use of enhanced virtual networks. The current design of DetNet assumes the design of the underlay network is unchanged. In this section we look at some changes that could be used to assist in achieving low latency ceiling across the wide area.

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, steering packets through specific queues may be required. This allows a much finer

control of which services wait for which, and a much finer granularity of queue management policy.

This may be introduced into traditional path construction techniques such as RSVP-TE and MPLS-TP, or it may be introduced by specifying the queue in an SR instruction list.

4.3. Support of an IP underlay

Where an underlay needs to be provided by IP, a number of options present themselves. We could allocate an IP address to that path and construct a path through the network for that IP address. The path could be laid in with RSVP signaling or through SDN controller. This path could have all of the required properties including specifying resources to use and functions to visit. Although this construct has been considered many time over the years, such a mechanism, at least to the author’s knowledge, has not found favor in deployment.

There are two ways that segment routing might be used to adapt the system described above to an IP context. One is to use the method described in [I-D.ietf-6man-segment-routing-header] in which each segment (instruction) is encoded as a normal IPv6 address. An alternative is to use the more compact representation considered in [I-D.xu-mpls-unified-source-routing-instruction] and [I-D.bryant-mpls-unified-ip-sr].

4.4. Application Specific Network Types

Although the transport service that underpins the extended VPN is likely MPLS/IP based, it needs to be able to carry application specific non-MPLS/IP traffic. This can be accommodated through the use of pseudowires (PWs).

4.5. A Hybrid Control Plane

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+.
[I-D.king-teas-applicability-actn-slicing] describes the use of ACTN to network slicing. This approach may be considered as part of the centralized control plane of VPN+ in some applications.

5. Applicability to Network Slicing

In [I-D.geng-netslices-architecture] a network slice is defined to be:

"A managed group of subsets of resources, network functions / network virtual functions at the data, control, management/orchestration planes and services at a given time. Network slice is programmable and has the ability to expose its capabilities. The behaviour of the network slice realized via network slice instance(s)."

A network slice instance (NSI) is then defined as:

"An activated network slice. It is created based on network template.
A set of managed run-time network functions, and resources to run these network functions, forming a complete instantiated logical network to meet certain network characteristics required by the service instance(s).
It provides the network characteristics that are required by a service instance. A network slice instance may also be shared across multiple service instances provided by the network operator. The network slice instance may be composed by none, one or more sub-network instances, which may be shared by another network slice instance."

A network slice can thus be thought of as a customized set of logical network and compute resources required by the service the slice is supporting. These resources support both virtual services in the data path and operation of the slice, for example by providing routing services. The customization includes the connectivity, performance, and isolation characteristics. These characteristics can be provided by the enhanced VPN described in this draft.

6. 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.

7. IANA Considerations

There are no requested IANA actions.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

8.2. Informative References

[FLEXE] "Flex Ethernet Implementation Agreement", March 2016, <http://www.oiforum.com/wp-content/uploads/OIF-FLEXE-01.0.pdf>.

[I-D.bryant-mpls-unified-ip-sr] Bryant, S., Xu, X., Chen, M., Farrel, A., and J. Drake, "A Unified Approach to IP Segment Routing", draft-bryant-mpls-unified-ip-sr-00 (work in progress), June 2017.

[I-D.galis-netslices-revised-problem-statement] Galis, A., "Network Slicing - Revised Problem Statement", draft-galis-netslices-revised-problem-statement-00 (work in progress), June 2017.

[I-D.geng-netslices-architecture] 67, 4., Dong, J., Bryant, S., kiran.makhijani@huawei.com, k., Galis, A., Foy, X., and S. Kuklinski, "Network Slicing Architecture", draft-geng-netslices-architecture-01 (work in progress), June 2017.

[I-D.ietf-6man-segment-routing-header] Previdi, S., Filsfils, C., Raza, K., Leddy, J., Field, B., daniel.voyer@bell.ca, d., daniel.bernier@bell.ca, d., Matsushima, S., Leung, I., Linkova, J., Aries, E., Kosugi, T., Vyncke, E., Lebrun, D., Steinberg, D., and R. Raszuk, "IPv6 Segment Routing Header (SRH)", draft-ietf-6man-segment-routing-header-06 (work in progress), March 2017.

[I-D.ietf-detnet-architecture] Finn, N., Thubert, P., Varga, B., and J. Farkas, "Deterministic Networking Architecture", draft-ietf-detnet-architecture-02 (work in progress), June 2017.

[I-D.ietf-spring-segment-routing]
Filsfils, C., Previdi, S., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", draft-ietf-spring-segment-routing-12 (work in progress), June 2017.

[I-D.king-teas-applicability-actn-slicing]
King, D., "Applicability of Abstraction and Control of TE Networks (ACTN) to Network Slicing", draft-king-teas-applicability-actn-slicing-00 (work in progress), June 2017.

[I-D.qiang-netslices-gap-analysis]
Qiang, L., Martinez-Julia, P., 67, 4., Dong, J., kiran.makhijani@huawei.com, k., Galis, A., Hares, S., and S. Slawomir, "Gap Analysis for Network Slicing", draft-qiang-netslices-gap-analysis-00 (work in progress), June 2017.

[I-D.xu-mpls-unified-source-routing-instruction]
Xu, X., Bryant, S., Raszuk, R., Chunduri, U., Contreras, L., Jalil, L., Assarpour, H., Velde, G., Tantsura, J., and S. Ma, "Unified Source Routing Instruction using MPLS Label Stack", draft-xu-mpls-unified-source-routing-instruction-02 (work in progress), June 2017.

Authors' Addresses

Stewart Bryant
Huawei

Email: stewart.bryant@gmail.com

Jie Dong
Huawei

Email: jie.dong@huawei.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.

Table of Contents

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

Table of Contents

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].

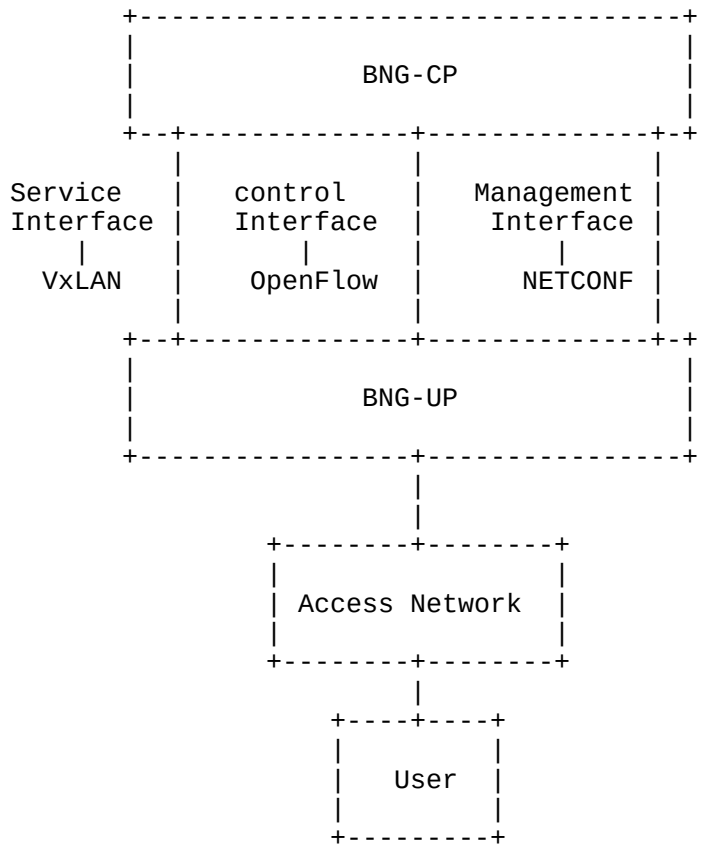

Figure 1: Architecture of C/U separated vBNG

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 seting 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 cofigure 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 tunnelis 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 Seperation 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: January 4, 2018

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

X. Liu
Jabil

July 3, 2017

YANG Logical Network Elements
draft-ietf-rtgwg-lne-model-03

Abstract

This document defines a logical network element 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.

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 4, 2018.

Table of Contents

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.

This document also defines the necessary augmentations for allocating host resources to a given LNE. As the interface management model [RFC7223] 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", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

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.

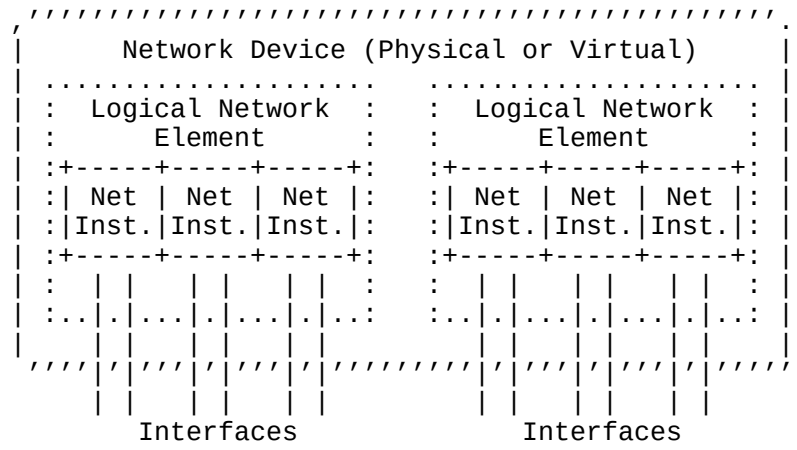

Figure 1: Module Element Relationships

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 [RFC7223] 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 [RFC7223]. The logical-network-element module augments existing interface management model by adding an identifier which is used on physical interface types 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 [RFC7223] 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 and 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 using the tree format defined in [I-D.ietf-netmod-yang-tree-diagrams] 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 [RFC7223] 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 [RFC7223].

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 'use-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 'use-schema' approach is expected to be be most useful in the case where all LNEs share the same schema. When 'use-schema' is used with an LNE mount point, the YANG library rooted in the LNE's mount point MUST match the associated schema defined within 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

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 [RFC7223], [RFC7317] and [RFC8022].

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.

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@2017-06-30.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 "RFC 7223: 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 2017-03-13 {
     description
       "Initial revision.";
     reference "RFC TBD";
   }

   // 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;
         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.";
       }
       yangmnt:mount-point "root" {
         description
           "Root for models supported per logical
            network element.  This mount point will
            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-schema-mount]
Bjorklund, M. and L. Lhotka, "YANG Schema Mount", draft-ietf-netmod-schema-mount-05 (work in progress), May 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>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://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, <http://www.rfc-editor.org/info/rfc6020>.

[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>.

7.2. Informative References

[I-D.ietf-netmod-yang-tree-diagrams]
Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-01 (work in progress), June 2017.

[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., and D. Bogdanovic, "YANG Network Instances", draft-ietf-rtgwg-ni-model-02 (work in progress), March 2017.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <http://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, <http://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, <http://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, <http://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].

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

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 [RFC7223], routing information base [RFC8022], and OSPF protocol. Each of these features is specified by a YANG model, and they are combined using YANG Schema Mount as follows:

```
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/
            |     ...
            +--ro rt:routing-state/
            |  +--ro router-id?                 yang:dotted-quad
            |  +--ro control-plane-protocols
            |     +--ro control-plane-protocol* [type name]
            |        +--ro ospf:ospf/
            |           +--ro instance* [af]
            |           ...
            +--rw rt:routing/
            |  +--rw router-id?                 yang:dotted-quad
            |  +--rw control-plane-protocols
            |     +--rw control-plane-protocol* [type name]
```

```
               |        +--rw ospf:ospf/
               |           +--rw instance* [af]
               |              +--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]
               |     |  ...
               +--ro if:interfaces-state/
                  +--ro interface* [name]
                     +--ro name               string
                     +--ro ip:ipv4!/
                     |  +--ro address* [ip]
                     |  ...

   module: ietf-interfaces
      +--rw interfaces
      |  +--rw interface* [name]
      |     +--rw name                         string
      |     +--rw lne:bind-lne-name?           string
      +--ro interfaces-state
         +--ro interface* [name]
            +--ro 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",
      "name": "root",
      "use-schema": [
        {
          "name": "lne-schema"
        }
      ]
    }
  ],
  "schema": [
    {
      "name": "lne-schema",
      "module": [
        {
          "name": "ietf-yang-library",
          "revision": "2016-06-21",
          "namespace":
            "urn:ietf:params:xml:ns:yang:ietf-yang-library",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-system",
          "revision": "2014-08-06",
          "namespace":
            "urn:ietf:params:xml:ns:yang:ietf-system",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-routing",
          "revision": "2016-11-04",
          "namespace":
            "urn:ietf:params:xml:ns:yang:ietf-routing",
          "conformance-type": "implement"
        },
        {
```

```
            "name": "ietf-ospf",
            "revision": "2017-03-12",
            "namespace":
              "urn:ietf:params:xml:ns:yang:ietf-ospf",
            "conformance-type": "implement"
          },
          {
            "name": "ietf-interfaces",
            "revision": "2014-05-08",
            "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"
          }
        ]
      }
    ]
  }
```

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
      +--ro routing-state
      |  +--ro router-id?                 yang:dotted-quad
      |  +--ro control-plane-protocols
      |  |  +--ro control-plane-protocol* [type name]
      |  |     +--ro ospf:ospf/
      |  |        +--ro instance* [af]
      +--rw routing
         +--rw router-id?                 yang:dotted-quad
         +--rw control-plane-protocols
            +--rw control-plane-protocol* [type name]
               +--rw ospf:ospf/
                  +--rw instance* [af]
                     +--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 interfaces-state
         +--ro interface* [name]
         +--ro name               string
         +--ro oper-status        enumeration
```

B.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": {
                      "instance": [
                        {
                        "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,
                        }
                      ]
                    }
                  }
                ]
              }
            }
          }
        },
        {
          "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": {
                      "instance": [
                        {
                          "af": "ipv4",
                          "areas": {
                            "area": [
                              {
                                "area-id": "203.0.113.1",
                                "interfaces": {
                                  "interface": [
```

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

    "ietf-interfaces:interfaces": {
      "interfaces": {
        "interface": [
          {
            "name": "eth0",
            "ip:ipv4": {
              "address": [
                {
                  "ip": "192.0.2.10",
                  "prefix-length": 24,
                }
              ]
```

```
            }
          },
          {
            "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": "2017-03-12",
      "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-ospf",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-routing",
      "revision": "2016-11-04",
      "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-state": {
              "router-id": "192.0.2.1",
              "control-plane-protocols": {
                "control-plane-protocol": [
                  {
                    "type": "ietf-routing:ospf",
                    "name": "1",
                    "ietf-ospf:ospf": {
                      "instance": [
                        {
                        "af": "ipv4",
                          "areas": {
                            "area": [
                              {
                                "area-id": "203.0.113.1",
                                "interfaces": {
                                  "interface": [
                                    {
                                      "name": "eth1",
                                      "cost": 10
                                    }
                                  ]
                                }
                              }
                            ]
                          }
                        }
                      ]
```

```
                    }
                  }
                ]
              }
            },
            "ietf-interfaces:interfaces-state": {
              "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": "2017-03-12",
      "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-ospf",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-routing",
      "revision": "2016-11-04",
      "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-state": {
              "router-id": "192.0.2.2",
              "control-plane-protocols": {
                "control-plane-protocol": [
                  {
                    "type": "ietf-routing:ospf",
                    "name": "1",
                    "ietf-ospf:ospf": {
                      "instance": [
                        {
                          "af": "ipv4",
                          "areas": {
                            "area": [
                              {
                                "area-id": "203.0.113.1",
                                "interfaces": {
                                  "interface": [
                                    {
                                      "name": "eth1",
                                      "cost": 10
                                    }
                                  ]
                                }
                              }
                            ]
                          }
                        }
                      ]
                    }
                  }
                ]
              }
            }
```

```
            }
            "ietf-interfaces:interfaces-state": {
              "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-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,
                }
              ]
            }
          },
          {
```

```
            "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-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": "2017-03-12",
          "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ospf",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-routing",
          "revision": "2016-11-04",
          "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-system:system-state": {
      "platform": {
        "os-name": "NetworkOS"
      }
    }
  }
```

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 [RFC7223], routing information base [RFC8022], 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 parament 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 interfaces-state
         +--ro interface* [name]
            +--ro 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",
      "name": "root",
      "inline": [null]
    }
  ]
}
```

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
   +--ro routing-state
   |  +--ro router-id?                 yang:dotted-quad
   |  +--ro control-plane-protocols
```

```
      |  |  +--ro control-plane-protocol* [type name]
      |  |     +--ro ospf:ospf/
      |  |        +--ro instance* [af]
      +--rw routing
         +--rw router-id?                 yang:dotted-quad
         +--rw control-plane-protocols
            +--rw control-plane-protocol* [type name]
               +--rw ospf:ospf/
                  +--rw instance* [af]
                     +--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 interfaces-state
         +--ro interface* [name]
         +--ro 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. 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": {
              "instance": [
                {
                  "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": {
            "instance": [
              {
                "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-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,
                 }
               ]
             }
           },
           {
             "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-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-system:system-state": {
      "platform": {
        "os-name": "NetworkOS"
      }
    }
  }
```

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": "2017-03-12",
          "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ospf",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-routing",
          "revision": "2016-11-04",
          "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-state": {
      "router-id": "192.0.2.1",
      "control-plane-protocols": {
        "control-plane-protocol": [
          {
            "type": "ietf-routing:ospf",
            "name": "1",
            "ietf-ospf:ospf": {
              "instance": [
                {
                  "af": "ipv4",
                  "areas": {
                    "area": [
                      {
                        "area-id": "203.0.113.1",
                        "interfaces": {
                          "interface": [
                            {
                              "name": "eth1",
                              "cost": 10
                            }
                          ]
                        }
                      }
                    ]
                  }
                }
              ]
            }
          }
        ]
```

```
      }
    },

    "ietf-interfaces:interfaces-state": {
      "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": "2017-03-12",
          "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ospf",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-routing",
          "revision": "2016-11-04",
          "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-state": {
      "router-id": "192.0.2.2",
      "control-plane-protocols": {
        "control-plane-protocol": [
          {
            "type": "ietf-routing:ospf",
            "name": "1",
            "ietf-ospf:ospf": {
              "instance": [
                {
                  "af": "ipv4",
                  "areas": {
                    "area": [
                      {
                        "area-id": "203.0.113.1",
                        "interfaces": {
                          "interface": [
                            {
                              "name": "eth1",
                              "cost": 10
                            }
                          ]
                        }
                      }
                    ]
                  }
                }
              ]
            }
          }
        ]
      }
```

```
    },

    "ietf-interfaces:interfaces-state": {
      "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: January 4, 2018

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

X. Liu
Jabil

July 3, 2017

YANG Network Instances
draft-ietf-rtgwg-ni-model-03

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).

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 4, 2018.

Table of Contents

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 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", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

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].

1.2. Status of Work and Open Issues

The top open issues are:

1. Schema mount currently doesn’t allow parent-reference filtering on the instance of the mount point, but rather just the schema. This means it is not possible to filter based on actual data, e.g., bind-network-instance-name="green". In the schema mount definition, the text and examples should be updated to cover this case.

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.

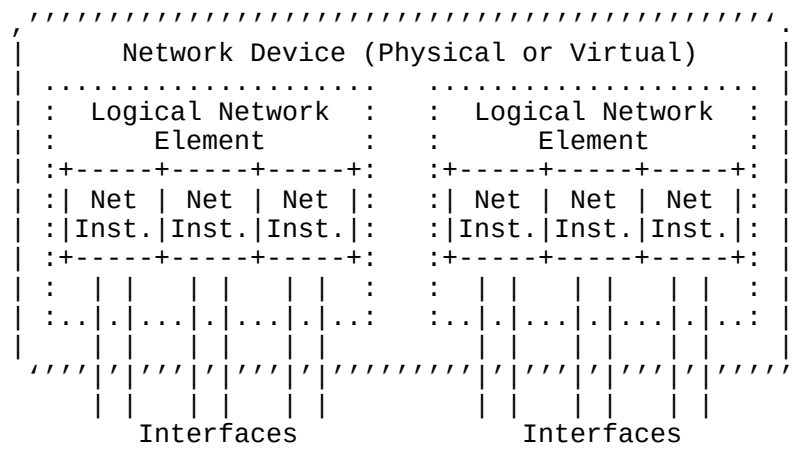

Figure 1: Module Element Relationships

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

The current interface management model [RFC7223] 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 [RFC8022] 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.

The NI model can be represented using the tree format defined in [I-D.ietf-netmod-yang-tree-diagrams] as:

```
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
                 +--ro rt:routing-state/
                 |  +--ro router-id?                yang:dotted-quad
                 |  +--ro control-plane-protocols
                 |     +--ro control-plane-protocol* [type name]
                 |        +--ro ospf:ospf/
                 |           +--ro instance* [af]
                 +--rw rt:routing/
                 |  +--rw router-id?                yang:dotted-quad
                 |  +--rw control-plane-protocols
                 |     +--rw control-plane-protocol* [type name]
                 |     +--rw ospf:ospf/
                 |        +--rw instance* [af]
                 |           +--rw areas
                 |              +--rw area* [area-id]
                 |                 +--rw interfaces
                 |                    +--rw interface* [name]
                 |                       +--rw name if:interface-ref
                 |                       +--rw cost?   uint16
                 +--ro if:interfaces@
                 |  ...
                 +--ro if:interfaces-state@
                 |  ...
```

This shows YANG Routing Management [RFC8022] 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 [RFC7223].

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 [RFC7223].

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 [RFC7277].

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 [RFC7223] 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 use-schema mechanism defined as part of the Schema Mount module [I-D.ietf-netmod-schema-mount] MUST be used with the module defined in this document to identify accessible modules. A future version of this document could relax this requirement. Mounted modules in the non-inline case SHOULD be defined with access, via the appropriate schema mount parent-references [I-D.ietf-netmod-schema-mount], to device resources such as interfaces.

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",
          "name": "vrf-root",
          "use-schema": [
            {
              "name": "ni-schema",
              "parent-reference": [
                "/*[namespace-uri() = 'urn:ietf:...:ietf-interfaces']"
              ]
            }
          ]
        }
      ],
      "schema": [
        {
          "name": "ni-schema",
          "module": [
            {
               "name": "ietf-routing",
               "revision": "2016-11-04",
               "namespace":
                 "urn:ietf:params:xml:ns:yang:ietf-routing",
               "conformance-type": "implement"
             },
            {
               "name": "ietf-ospf",
               "revision": "2017-03-12",
               "namespace":
                 "urn:ietf:params:xml:ns:yang:ietf-ospf",
               "conformance-type": "implement"
            }
          ]
        }
      ]
    }
```

Module data identified under "schema" 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.

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

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 [RFC8022], 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 [RFC7223] and [RFC8022].

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@2017-07-03.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 "RFC 7223: A YANG Data Model for Interface
               Management";
  }
  import ietf-ip {
    prefix ip;
    reference "RFC 7277: 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 2017-07-02 {
    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 "RFC 8022 - A YANG Data Model for Routing
                  Management";
       list network-instance {
         key "name";
         description
           "List of network-instances.";
         leaf name {
           type string;
           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 {
           description
             "Well known mount points.";
           yangmnt:mount-point "vrf-root" {
             description
               "Root for L3VPN type models. This will typically
```

```
              not be an inline type mount point.";
          }
          yangmnt:mount-point "vsi-root" {
            description
              "Root for L2VPN type models. This will typically
               not be an inline type 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-schema-mount]
Bjorklund, M. and L. Lhotka, "YANG Schema Mount", draft-ietf-netmod-schema-mount-05 (work in progress), May 2017.

[I-D.ietf-netmod-yang-tree-diagrams]
Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-01 (work in progress), June 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>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://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, <http://www.rfc-editor.org/info/rfc6020>.

[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>.

[RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <http://www.rfc-editor.org/info/rfc7277>.

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-05 (work in progress), March 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-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-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., and D. Bogdanovic, "YANG Logical Network Elements", draft-ietf-rtgwg-lne-model-02 (work in progress), March 2017.

[RFC4026] Andersson, L. and T. Madsen, "Provider Provisioned Virtual Private Network (VPN) Terminology", RFC 4026, DOI 10.17487/RFC4026, March 2005, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc4664>.

[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>.

[RFC8022] Lhotka, L. and A. Lindem, "A YANG Data Model for Routing Management", RFC 8022, DOI 10.17487/RFC8022, November 2016, <http://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].

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": {
                    "instance": [
                      {
                        "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": {
                      "instance": [
                        {
                          "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,
                  }
                ]
              }
            },
            {
              "name": "eth1",
              "ip:ipv4": {
                "address": [
                  {
                    "ip": "192.0.2.11",
                    "prefix-length": 24,
                  }
                ]
              },
              "ni:bind-network-instance-name": "vrf-red"
            },
            {
              "name": "eth2",
              "ip:ipv4": {
                "address": [
                  {
                    "ip": "192.0.2.11",
                    "prefix-length": 24,
                  }
                ]
              },
              "ni:bind-network-instance-name": "vrf-blue"
```

```
          }
        ]
      }
    },

    "ietf-system:system": {
      "authentication": {
        "user": [
          {
            "name": "john",
            "password": "$0$password"
          }
        ]
      }
    }
  }
```

B.2. State Data

The following shows state data for the example above.

```
  {
    "ietf-network-instance:network-instances": {
      "network-instance": [
        {
          "name": "vrf-red",
          "vrf-root": {
            "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": {
                      "instance": [
                        {
                          "af": "ipv4",
                          "areas": {
                            "area": [
                              {
                                "area-id": "203.0.113.1",
                                "interfaces": {
                                  "interface": [
                                    {
                                      "name": "eth1",
                                      "cost": 10
                                    }
```

```
                                ]
                              }
                            }
                          ]
                        }
                      }
                    ]
                  }
                }
              ]
            }
          }
        }
      },
      {
        "name": "vrf-blue",
        "vrf-root": {
          "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": {
                    "instance": [
                      {
                        "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,
              }
            ]
          }
        },
        {
          "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": "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,
                }
              ]
            }
          }
        ]
      }
    },

    "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": "2017-03-13",
          "feature": [
            "bind-network-instance-name"
          ],
          "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-network-instance",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-ospf",
          "revision": "2017-03-12",
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-ospf",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-routing",
          "revision": "2016-11-04",
          "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-system:system-state": {
      "platform": {
        "os-name": "NetworkOS"
      }
    }
  }
```

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: March 23, 2018

X. Liu
Jabil
Y. Qu
Futurewei Technologies, Inc.
A. Lindem
Cisco Systems
C. Hopps
Deutsche Telekom
L. Berger
LabN Consulting, L.L.C.
September 19, 2017

Routing Area Common YANG Data Types
draft-ietf-rtgwg-routing-types-14

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 March 23, 2018.

Copyright Notice

Table of Contents

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 target="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-te].

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@2017-09-19.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 2017-09-19 {
    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}|'
        +     '[0-5]?[0-9]{0,3}[0-9]):(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}|'
          +     '[0-3]?[0-9]{0,8}[0-9]))|'
          + '(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}|'
          +     '[0-5]?[0-9]{0,3}[0-9]))|'
          + '(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}|'
          +     '[0-3]?[0-9]{0,8}[0-9]):'
          +     '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|'
          +     '6[0-4][0-9]{3}|'
          +     '[0-5]?[0-9]{0,3}[0-9]))|'
          + '(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]|[01]?[0-9]?[0-9])\.){3}'
            + '(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])))'
            + ':'
            + '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|'
            + '6[0-4][0-9]{3}|'
            + '[0-5]?[0-9]{0,3}[0-9])';
        pattern '((([^:]+:){6}(([^:]+:[^:]+)|(.*\..*)))|'
            + '((([^:]+:)*[^:]+)?::(([^:]+:)*[^:]+)?))'
            + ':'
            + '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|'
            + '6[0-4][0-9]{3}|'
            + '[0-5]?[0-9]{0,3}[0-9])';
      }
      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}|'
        +     '[0-5]?[0-9]{0,3}[0-9]):(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}|'
        +     '[0-3]?[0-9]{0,8}[0-9]))|'
        + '(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}|'
        +     '[0-5]?[0-9]{0,3}[0-9]))|'
        + '(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}|'
        +     '[0-3]?[0-9]{0,8}[0-9]):'
        +     '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|'
        +     '6[0-4][0-9]{3}|'
        +     '[0-5]?[0-9]{0,3}[0-9]))|'
        + '(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}|'
        +     '[0-5]?[0-9]{0,3}[0-9]):(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}|'
        +     '[0-3]?[0-9]{0,8}[0-9]))|'
        + '(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}|'
        +     '[0-5]?[0-9]{0,3}[0-9]))|'
```

```
          + '(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}|'
          +     '[0-3]?[0-9]{0,8}[0-9]):'
          +     '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|'
          +     '6[0-4][0-9]{3}|'
          +     '[0-5]?[0-9]{0,3}[0-9]))|'
          + '(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]|[01]?[0-9]?[0-9])\.){3}'
            + '(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])))'
            + ':'
            + '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|'
            + '6[0-4][0-9]{3}|'
            + '[0-5]?[0-9]{0,3}[0-9])';
        pattern '((([^:]+:){6}(([^:]+:[^:]+)|(.*\..*)))|'
            + '((([^:]+:)*[^:]+)?::(([^:]+:)*[^:]+)?))'
            + ':'
            + '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|'
            + '6[0-4][0-9]{3}|'
            + '[0-5]?[0-9]{0,3}[0-9])';
      }
      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 or 0X1.HHHHHHP{+}D
         where 'h' and 'H' are hexadecimal digits, 'd' and 'D' are
         integers in the range of [0..127].
         When six hexadecimal digits are used for 'hhhhhh' or
         'HHHHHH', 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 e164-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.

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:

"val": 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 data types using the YANG data modeling language. The definitions themselves have no security 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-08 (work in progress), April 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-06 (work in progress), June 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

Networking Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 28, 2017

T. Przygienda
Juniper Networks
A. Sharma
Comcast
J. Drake
A. Atlas
Juniper Networks
June 26, 2017

RIFT: Routing in Fat Trees
draft-przygienda-rift-02

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 and ultimately (6) 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 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 December 28, 2017.

Table of Contents

1. Introduction

Clos [CLOS] and Fat-Tree [FATTREE] have gained prominence in today's networking, primarily as a result of a 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. The existing set of dynamic routing protocols was geared originally towards a network with an irregular topology and low degree of connectivity and consequently several attempts to adapt those 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 fulfill certain of the resulting requirements.

In looking at the problem through the very lens of its requirements an optimal approach does not seem 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’. The balance of this document details the resulting protocol.

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.

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 at level 0.

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.

2.2. Topology

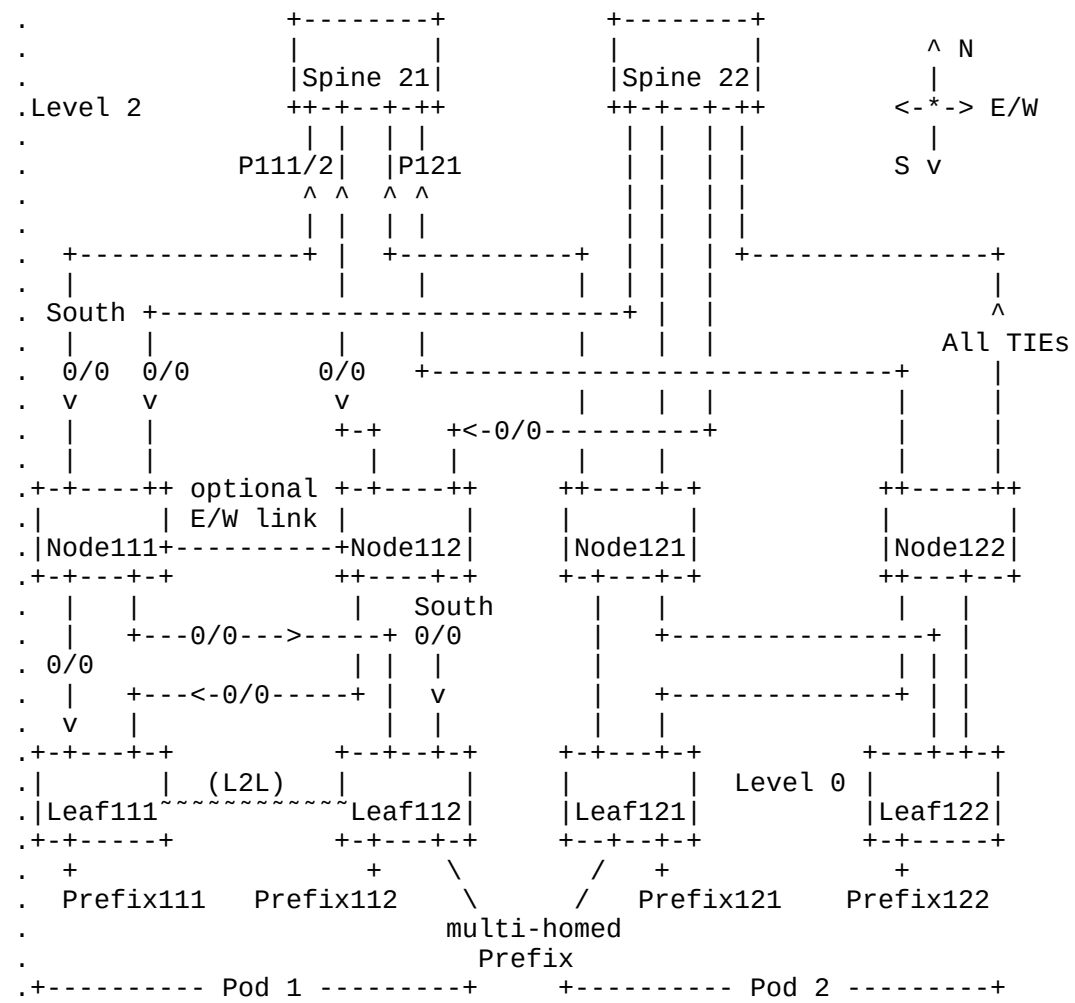

Figure 1: A two level spine-and-leaf topology

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.

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 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.

REQ2: Very high degree of ECMP (and ideally non equal cost) must be supported. Maximum ECMP is currently understood as the most efficient routing approach to maximize the throughput of switching fabrics [MAKSIC2013].

REQ3: Traffic engineering should be allowed by modification of prefixes and/or their next-hops.

REQ4: 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.

REQ5: 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].

REQ6: 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.

REQ7: 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.

REQ8: 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.

REQ9: A node without any configuration beside default values should come up as leaf 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.

REQ10: 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.

REQ11: 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.

REQ12: Parallel links between same set of nodes must be distinguishable for SPF, failure and traffic engineering purposes.

REQ13: 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.

REQ14: Optionally, the protocol should allow to provision data centers where the individual switches carry no configuration information and are all determining the topology 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.

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 novel 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.

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.

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 link in the topology. LIEs are exchanged over all links running RIFT.

Each node is provisioned with the level at which it is operating and its PoD. A default level and PoD of zero are assumed, meaning that leafs do not need to be configured with a level (or even PoD). Nodes in the spine are configured with a PoD of zero. This information is propagated in the LIEs exchanged. Adjacencies are formed if and only if

1. the node is in the same PoD or either the node or the neighbor advertises "any" PoD membership (PoD# = 0) AND

2. the neighboring node is at most one level away AND

3. the neighboring node is running the same MAJOR schema version AND

4. the neighbor is not member of some PoD while the node has a northbound adjacency already joining another PoD.

A node configured with "any" 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.

LIE exchange uses three-way handshake mechanism [RFC5303]. Precise final 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 side of the scale 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 south or to the north/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 neighbors 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.

As an example illustrating a databases holding both representations, consider the topology in Figure 1 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-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)))
SouthPrefixesElement(prefixes(0/0, cost 1), (::/0, cost 1))

Node111 S-TIE:
NodeElement(layer=1, neighbors((Spine21,layer 2,cost 1),
(Spine22, layer 2, cost 1), (Node112, layer 1, cost 1),
(Leaf111, layer 0, cost 1), (Leaf112, layer 0, cost 1)))
SouthPrefixesElement(prefixes(0/0, cost 1), (::/0, cost 1))

Node111 N-TIE:
NodeLinkElement(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(...))))
NorthPrefixesElement(prefixes(Node111.loopback)

Node121 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)))
SouthPrefixesElement(prefixes(0/0, cost 1), (::/0, cost 1))

Node121 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(...))))
NorthPrefixesElement(prefixes(Node121.loopback)

Leaf112 N-TIE:
NodeLinkElement(layer=0,
neighbors((Node111, layer 1, cost 1, links(...)),
(Node112, layer 1, cost 1, links(...))))
NorthPrefixesElement(prefixes(Leaf112.loopback, Prefix112,
Prefix_MH))
```

Figure 2: 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.

Precise final 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 more southern level, e.g. in case of leaf level. The precise flooding scopes are given in Table 1. Those rules govern in a symmetric fashion what SHOULD be included in TIDEs towards neighbors.

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	East-West
node S-TIE	flood own only	flood always	same as South
other S-TIE	flood own only	flood only if TIE originator is equal peer	same as South
all N-TIEs	never flood	flood always	same as South
TIDE	include TIEs in flooding scope	include TIEs in flooding scope	same as South
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	same as South

Table 1: Flooding Scopes

As an example to illustrate these rules, consider using the topology in Figure 1, with the optional link between node 111 and node 112, and the associated TIEs given in Figure 2. The flooding from particular nodes of the TIEs is given in Table 2.

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, Node112 N-TIEs, Leaf111 N-TIEs, Leaf112 N-TIEs, Spine22 node S-TIE
Node111	Spine22	Node111 N-TIEs, Node112 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 2: 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 1.

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 through many years of experience to be complex and fragile. 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 looses 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.

4.2.3.7. Southbound Default Route Origination

Under certain conditions nodes issue a default route in their South Prefix TIEs.

A node X that is NOT overloaded AND has southbound or east-west adjacencies originates in its south prefix TIE such a default route IIF all other nodes at X's' level are overloaded OR have NO northbound adjacencies OR X has computed reachability to a default route during N-SPF.

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 node S-TIEs have been received and which have precisely the same non-empty sets of respectively north and south neighbor adjacencies and 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 XOR(TIE-ORIGINATOR-ID<<1,~OWN-SYSTEM-ID)), whereas << is a non-circular shift and ~ 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 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 adjacencies in north node TIEs when progressing Dijkstra. N-SPF uses E-W adjacencies during N-SPF ONLY if the node itself does NOT have any northbound adjacencies and the adjacent node has one or more northbound adjacencies. This forms a "one-hop split-horizon" and prevents looping over default routes while allowing for "one-hop protection" of nodes that lost all northbound adjacencies. A detailed example can be found in Section 4.3.8.

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, 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 the 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).

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 3: 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 4: 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 5: 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.

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

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 TIEs (which are always S-TIEs) can arrive from multiple nodes and need tie-breaking per key uses 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 S-TIE with the highest level and within the same level highest originator ID is preferred.

Observe that if a node goes down, the node south of it looses 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 is not result of independent computation of every node but a diffused computation.

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 looses 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. Leaf to Leaf Procedures

RIFT can be optionally relaxed to allow leaf East-West adjacencies under additional set of rules. The leaf supporting those procedures MUST:

Only nodes supporting Leaf to Leaf Procedures CAN advertise LIEs on E-W links at level 0 and MUST in such a case advertise the according flag in node capabilities as "true".

The overload bit MUST be set on all leaf’s node TIEs.

Only node’s own north and south TIEs are flooded over E-W leaf adjacency.

E-W leaf adjacency is always used in both north as well as south computation.

Any advertised aggregate in leaf’s south TIE MUST install a discard route.

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).

4.3.6. 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.

4.3.7. 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.

4.3.8. 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 looses 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.

4.3.9. 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. Section 5.4 explains the resulting behavior based on an 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

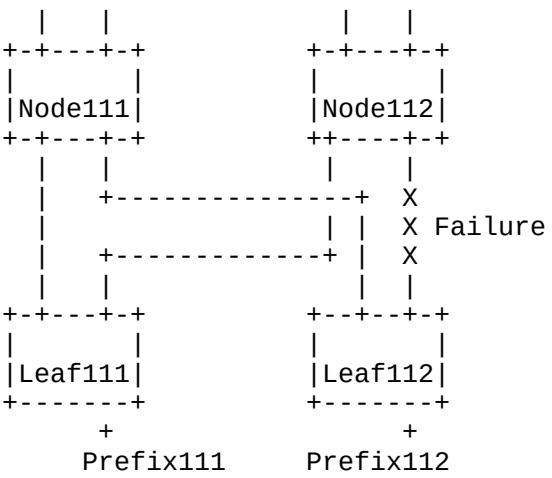

Figure 6: Single Leaf link failure

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

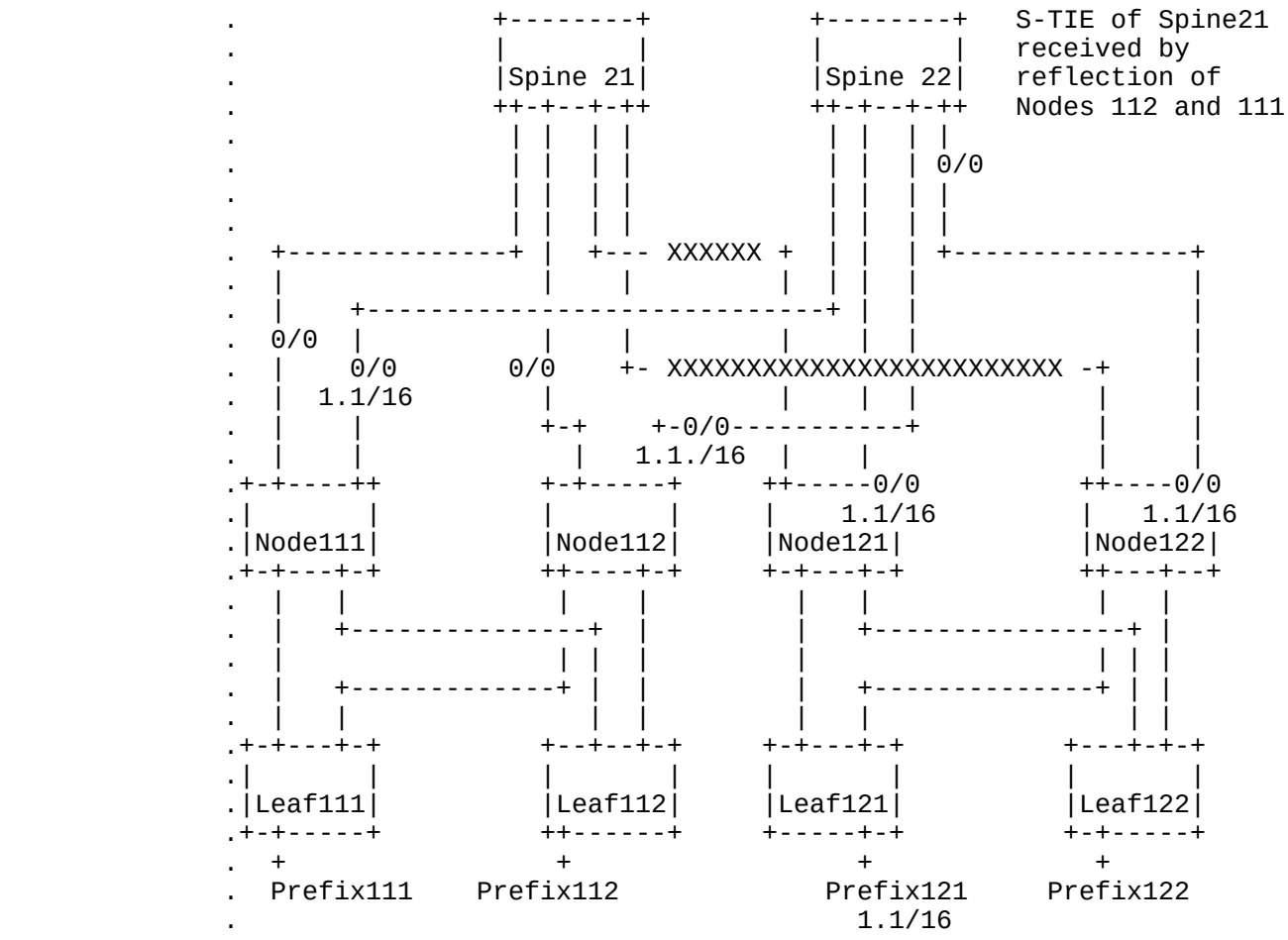

Figure 7: Fabric partition

Figure 7 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

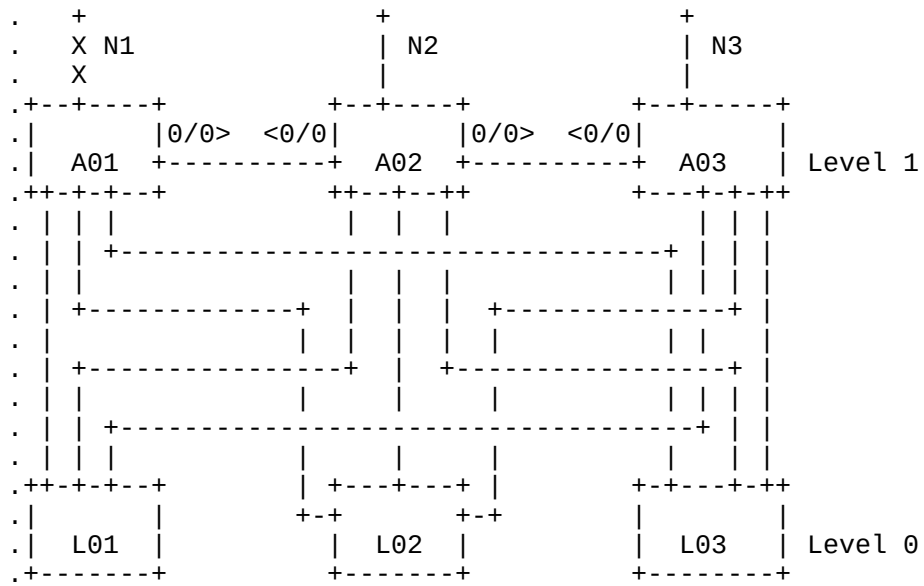

Figure 8: North Partitioned Router

Figure 8 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 4.3.8 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 looses 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

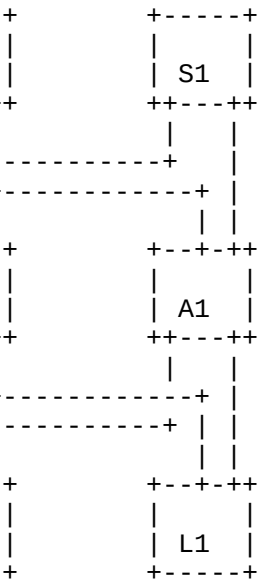

Figure 9: Level Shortcut

Strictly speaking, RIFT is not limited to Clos variations only. The protocol preconditions only a sense of 'compass rose direction' achieved by configuration 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 9 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.

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. change lists into sets, unions into structures or

4. change multiplicity of fields or

5. change datatypes of any field or

6. 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.

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 for common definitions in RIFT
*/

namespace * models

typedef i64    SystemID
typedef i32    IPv4Address
/** this has to be of length long enough to accomodate prefix */
typedef binary IPv6Address
typedef i16    UDPPortType
typedef i32    TIENrType
typedef i16    MTUSizeType
typedef i32    SeqNrType
/** lifetime in seconds */
typedef i32    LifeTimeType
typedef i16    LevelType
typedef i16    PodType
typedef i16    VersionType
typedef i32    MetricType
typedef string KeyIDType
```

```
   /** node local, unique identification for a link, this is kept
     * at 32 bits so it aligns with BFD [RFC5880] discriminator size
     */
   typedef i32    LinkIDType
   typedef string KeyNameType
   /** indicates whether the direction is northbound/east-west
     * or southbound */
   typedef bool   TieDirectionType
   typedef byte   PrefixLenType
   /** timestamp in seconds since the epoch */
   typedef i64    TimestampInSecsType
   /** security nonce */
   typedef i64    NonceType

   const LevelType   default_level           = 0
   const PodType     default_pod             = 0
   const LinkIDType  undefined_linkid        = 0
   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 bool leaf_2_leaf_procedures_default = false;

   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 IPPrefixType {
       1: optional IPv4PrefixType   ipv4prefix;
```

```
        2: optional IPv6PrefixType   ipv6prefix;
    }

    enum TIETypeType {
        Illegal              =   0,
        TIETypeMinValue      =   1,
        /** first legal value */
        NodeTIEType          =   2,
        PrefixTIEType        =   3,
        PGPrefixTIEType      =   4,
        KeyValueTIEType      =   5,
        TIETypeMaxValue      =   6,
    }

    /** @note: route types which MUST be ordered on preference
     *  PGP prefixes are most preferred attracting
     *  traffic north (towards spine)
     *  normal prefixes are attracting traffic south (towards leafs),
     *  i.e. prefix in NORTH PREFIX TIE is preferred
     */
    enum RouteType {
        Illegal           = 0,
        RouteTypeMinValue = 1,
        /** First legal value. Local prefixes are
         *  locally hosted routes on the system.
         */
        LocalPrefix       = 2,
        SouthPGPPrefix    = 3,
        NorthPGPPrefix    = 4,
        NorthPrefix       = 5,
        SouthPrefix       = 6,
        /** advertised in N-TIEs */
        RouteTypeMaxValue = 7
    }
```

8.2. encoding.thrift

```
/**
    Thrift file for packet encodings for RIFT
*/

include "common.thrift"

namespace * models

/** represents protocol major version */
const i32 current_major_version = 2
/** represents protocol minor version */
```

```
const i32 current_minor_version = 1

/** coomon RIFT packet header */
struct PacketHeader {
    1: required common.VersionType major_version = current_major_version;
    2: required common.VersionType minor_version = current_minor_version;
    /** this is the node sending the packet, in case of LIE/TIRE/TIDE
        also the originator of it */
    3: required common.SystemID  sender;
    /** level of the node sending the packet */
    4: required common.LevelType level;
}

/** protocol packet structure */
struct ProtocolPacket {
    1: required PacketHeader header;
    2: required Content content;
    }

union Content {
    1: optional LIE          hello;
    2: optional TIDEPacket   tide;
    3: optional TIREPacket   tire;
    4: optional TIEPacket    tie;
}

/** Community serves as community for PGP purposes */
struct Community {
    1: required i32          top;
    2: required i32          bottom;
}

/** @todo: flood header separately in UDP to allow caching to TIEs
           while changing lifetime?
 */
struct TIEPacket {
    1: required TIEHeader  header;
    2: required TIEElement element;
}

/** RIFT LIE packet

    @note this node's level is already included on the packet header */
struct LIE {
    1: optional string                name;
    /** UDP port to which we can flood TIEs, same address
        as the hello TX this hello has been received on */
    /** local link ID */
```

```
    2: required common.LinkIDType      local_id;
    3: required common.UDPPortType     flood_port;
    /** this will reflect the neighbor once received */
    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 capabilies
        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;
}

/** Neighbor structure */
struct Neighbor {
    1: required common.SystemID        originator;
    2: required common.LinkIDType      remote_id;
}

/** 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 whether N or S-TIE, True > False */
    1: required common.TieDirectionType    northbound;
    2: required common.SystemID            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;
    4: required common.LifeTimeType        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;
    3: optional common.MetricType      cost = common.default_distance;
    /** can carry description of multiple parallel links in a TIE
     *  all parallel links to same node incur same cost
     **/
    4: optional set<LinkIDPair>            link_ids;
}

/** Capabilities the node supports */
struct NodeCapabilities {
    /** can this node participate in flood reduction,
        only relevant at level > 0 */
    1: required bool         flood_reduction;
    /** does this node support leaf-2-leaf procedures,
        only relevant at level 0 */
    2: optional bool         leaf_2_leaf_procedures;
}

/** Flags the node sets */
struct NodeFlags {
    /** node is in overload, do not transit traffic through it */
    1: required bool         overload;
}

/** 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
        * neighbors repeat with different values
```

```
    the behavior is undefined and a warning
    SHOULD be generated.

    @note: observe that absence of fields implies defined defaults
*/
struct NodeTIEElement {
    1: required common.LevelType                              level;
    2: optional NodeCapabilities                              capabilities =
            {
              "flood_reduction": common.flood_reduction_default,
              "leaf_2_leaf_procedures": common.leaf_2_leaf_procedures_default
            };
    3: optional NodeFlags                                     flags =
            {
              "overload": common.overload_default
            };
    /** if neighbor systemID repeats in other node TIEs of same node
        the behavior is undefined */
    4: required map<common.SystemID,NodeNeighborsTIEElement> neighbors;
}

/** multiple prefixes */
struct PrefixTIEElement {
    /** prefixes with the associated cost.
        if the same prefix repeats in multiple TIEs of same node
        or with different metrics, behavior is unspecified */
    1: required map<common.IPPrefixType,common.MetricType> 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.TIETypeType
    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.TIETypeType.NodeTIEType */
    1: optional NodeTIEElement            node;
    /** in case of enum common.TIETypeType.PrefixTIEType */
    2: optional PrefixTIEElement          prefixes;
    3: optional KeyValueTIEElement        keyvalues;
    /** @todo: policy guided prefixes */
```

}

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. Security Considerations

Security mechanisms will be addressed in upcoming versions of this specification.

11. 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. Adrian Farrel 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.

12. References

12.1. 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, <http://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, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2328] Moy, J., "OSPF Version 2", STD 54, RFC 2328, DOI 10.17487/RFC2328, April 1998, <http://www.rfc-editor.org/info/rfc2328>.

[RFC2365] Meyer, D., "Administratively Scoped IP Multicast", BCP 23, RFC 2365, DOI 10.17487/RFC2365, July 1998, <http://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, <http://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, <http://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, <http://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, <http://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, <http://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, <http://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, <http://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, <http://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, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc7987>.

12.2. 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.

[MAKSIC2013] Maksic et al., N., "Improving Utilization of Data Center Networks", IEEE Communications Magazine, Nov 2013.

[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
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

John Drake
Juniper Networks
1194 N. Mathilda Ave
Sunnyvale, CA 94089
US

Email: jdrake@juniper.net

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

Email: akatlas@juniper.net

Network Working Group
Internet-Draft
Intended status: Informational
Expires: October 4, 2017

R. White, Ed.
S. Zandi, Ed.
LinkedIn

April 2, 2017

Openfabric
draft-white-openfabric-02

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:

- Provide a full view of the topology from a single point in the network to simplify operations
- Minimize configuration of each router (or switch) in the network
- 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 http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 4, 2017.

Table of Contents

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:

- Provide a full view of the topology from a single point in the network to simplify operations
- Minimize configuration of each router (or switch) in the network
- 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 Triantafillis (reflected flooding optimization), Ivan Pepelnjak (three stage fabric modifications), 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:

- Multilevel flooding domain support. The modifications described in this document will not work across multiple flooding domains. It is assumed that multiple fabrics will be connected through an Exterior Gateway Protocol (EGP), specifically BGP [RFC4271].
- All mutliaccess link processing, including Designated Intermediate Systems (DIS). Spine and leaf fabrics are normally built using only point-to-point links, so multiaccess link processing is not required in openfabric.

- 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.

- 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 router 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:

- A slightly modified adjacency formation process. This is largely a matter of forming adjacencies in a specific order, rather than forming an adjacency with every discovered neighbor at the same time.

- A mechanism for determining which tier within a spine and leaf fabric in which the router is located.

- A mechanism that reduces flooding to the minimum possible, while still ensuring complete database synchronization among the routers within the fabric.

- New sub-TLVs to carry openfabric specific information; specifically a new IS reachability tier sub-TLV.

Openfabric implementations:

- MUST support [RFC5301] and enable hostname advertisement by default if a hostname is configured on the intermediate system.

- MUST support [RFC5311], simplified extension of the link state PDU space for IS-IS.

- MUST support [RFC5303] and enable three-way handshakes by default.

- MUST use TLV type 135 for carrying IPv4 reachability information, as defined in [RFC5305].

- MUST use TLV type 236 for carrying IPv6 reachability information, as defined in [RFC5308].

- MUST use TLV type 22 for carrying IS reachability information, as defined in [RFC5305].

- SHOULD support [RFC6232], purge originator identification for IS-IS.

- SHOULD support Segment Routing (SR). [I-D.ietf-spring-segment-routing]

- SHOULD support [I-D.ietf-isis-segment-routing-extensions].

- 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.

Openfabric implementations 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:

- 5A is connected to 4A, 4B, 4C, 4D, 4E, and 4F
- 5B is connected to 4A, 4B, 4C, 4D, 4E, and 4F
- 4A is connected to 3A, 3B, 3C, 3D, 3E, 3F, 5A, 5B, 5C, 5D, 5E, and 5F
- 4B is connected to 3A, 3B, 3C, 3D, 3E, 3F, 5A, 5B, 5C, 5D, 5E, and 5F
- 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) routers. 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

While adjacency formation is not considered particularly burdensome in IS-IS, it is still useful to reduce the amount of state transferred across the network when connecting a new router to the fabric. 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. The process is:

- An IS connected to the fabric will send hellos on all links.
- 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.
- The IS will complete its database exchange with this one newly adjacent neighbor.
- Once this process is completed, the IS will continue processing the remaining neighbors as normal.

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.

3. Determining Location on the Fabric

The tier to which a router is connected is useful to enable autoconfiguration of routers connected to the fabric, and to reduce flooding. This section describes mechanisms for determining the tier at which a router is connected in the fabric in several steps. The first step is to find the Farthest Distance (FD) and the Total Distance (TD), which are useful in this process. To find the FD and TD:

- Calculate a Shortest Path Tree (SPT) for the entire network with all link metrics set to 1; this has the effect of calculating a tree based only on hop count
- Find one node that is the farthest from the local node in the resulting tree; call this node F, and the distance to this node FD

- Calculate an SPT for the entire network with all link metrics set to 1 from the perspective of F; call this TD

3.1. Determining T0

If FD == TD == 2, this is a three stage fabric; it is not possible to determine the tier at which the local node is located based on any calculation, because the topology is perfectly symmetric. In this case:

- The T0 routers MAY be manually configured to advertise 0x00 in their IS reachability tier sub-TLV, indicating they are at the edge of the fabric (a ToR router).

- The T0 routers MAY detect that they are T0 through the presence connected hosts (i.e. through a request for address assignment or some other means). This means of detection may not be reliable in all operational environments, and SHOULD be used with care. If such detection is used, and the router determines it is located at T0, it should advertise 0x00 in its IS reachability tier sub-TLV.

- The router MAY examine the IS reachability tier sub-TLV of directly connected neighbors and determine one or more is advertising 0x1 in its IS reachability tier sub-TLVs. This would be the case if the spine routers in a three stage spine and leaf fabric are manually configured to advertise their tier as 0x1.

- If there is no way to determine whether or not the local device is in T0 or T1, it MUST advertise 0xFF in its IS reachability tier sub-TLV.

If FD == TD, and TD >= 4, this is a greater than three stage fabric; the local device SHOULD advertise 0x00 in its IS reachability tier sub-TLV.

For instance, in the diagram above, 1A would:

- Calculate an SPT with all link metrics set to 1; on this SPT, 5A through 5F would all have a distance of 4

- Select one of these nodes as F; assume 5F is chosen as F

- Set FD to 4, the distance to 5F

- Run SPF from the perspective of 5F with all link metrics set to 1

- Set TD to 4, the cost from 5F to 1A

- TD - FD == 0, so 1A is at T0, and is a ToR

3.2. Determining T1 and above

If FD == TD == 2, this is a three stage fabric; it is not possible to determine the tier at which the local node is located based on any calculation, because the topology is perfectly symmetric. In this case:

- The T1 routers MAY be manually configured to advertise 0x01 in their IS reachability tier sub-TLV.
- The router MAY examine the IS reachability tier sub-TLV of directly connected neighbors and determine that one or more is advertising 0x00 in its IS reachability tier sub-TLVs. This would be the case if the ToR routers in a three stage spine and leaf fabric are manually configured to advertise their tier as 0x00.
- If there is no way to determine whether or not the local device is in T0 or T1, it should advertise 0xFF in its IS reachability tier sub-TLV.

If TD != FD, this is a greater than three stage fabric; the local device SHOULD advertise (TD - FD) in its IS reachability tier sub-TLV.

For example, in the above five stage fabric, 3B would:

- Calculate an SPT with all link metrics set to 1; on this SPT, 5A through 5F and 1A through 1F would all have a cost of 2
- Select one of these nodes as F; assume 5F is chosen as F
- Set FD to 2, the distance to 5F
- Run SPF from the perspective of 5F with all link metrics set to 1
- Set TD to 4, the cost from 5F to 1A
- TD - FD == 2, so 1A is at T2, and is a spine switch

4. Flooding Optimization

Flooding is perhaps the most challenging scaling issue for a link state protocol running on a dense, large scale fabric. To reduce flooding, 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:

- NL list: The set of neighbors
- NN list: The set of neighbor’s neighbors; this can be calculated by running SPF truncated to two hops
- DNR list: The set of neighbors who should have LSPs (or fragments) marked Do Not Reflood (DNR)
- 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 router ID 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:

- Move any IS in NL with its tier (or fabric location) set to T0 to DNR
- 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:

- 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
- Else move the IS to DNR

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, will be transmitted to the DNR address (see modifications to the IS-IS protocol, below).

Any IS receiving a link state packet transmitted to the DNR address SHOULD NOT set the Send Route Message (SRM) flag on any interface for

this LSP; hence the LSP will not be reflooded by this IS to any adjacent neighbor. This reduces flooding to the minimum possible while retaining full Link State Database (LSDB) synchronization.

4.1. 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:

- Set a short timer; the default should be less than one second
- When the timer expires, send a Complete Sequence Number Packet (CSNP) to all neighbors
- Process any Partial Sequence Number Packets (PSNPs) as required to resynchronize
- If a resynchronization is required, notify the network operator through a network management system

5. Other Optimizations

5.1. 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 router SHOULD be configured with a single loopback address, which is assigned an IPv6 address, to provide reachability to routers which make up the fabric.

5.2. Transiting T0 Routers

In data center fabrics, ToR routers SHOULD NOT be used to transit between two T1 (or above) spine routers. The simplest way to prevent this is to set the overload bit [RFC3277] for all the LSPs originated from T0 routers. However, this solution would have the unfortunate side effect of causing all reachability beyond any T0 router 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 routers.

Specifically, all T0 routers SHOULD advertise their metric to reach any T1 adjacent neighbor with a cost of 0XFFE. T1 routers, on the other hand, will advertise T0 routers with the actual interface cost used to reach the T0 router. Hence, links connecting T0 and T1 routers will be advertised with an asymmetric cost that discourages transiting T0 routers, while leaving reachability to the destinations attached to T0 devices the same.

6. Openfabric and Route Aggregation

While aggregation is not recommended in openfabric deployments, aggregation MAY take place when routing information is being transmitted from higher level tiers to lower level tiers. For instance, in the example network, 2A through 2F could advertise a single default route to 1A through 1F. 2A through 2F would simply advertise the default as if it were an attached to each router locally using either a type 135 or 236 TLV, and then block TLVs that contain reachability information (such as types 135 and 236). Type 22 TLVs, however, MUST be flooded through this boundary, so that every router in the network shares a common view of the topology.

Note that aggregation in a DC fabric can result in routing black holes in some cases, and also possibly reduce the efficiency of traffic engineering in the network.

7. Openfabric Modifications to the IS-IS protocol

7.1. Advertising Tier Level

The tier level is carried as a subTLV of the router capabilities TLV described in [RFC7981]. The subTLV number is TBA; the TLV will contain one octet encoding the tier number as an integer. If the openfabric IS only supports IPv6, the procedure described in section 3 of [RFC7981] MUST be followed, and an IPv6 address carried in the IPv6 TE Router ID sub-TLV according to [RFC5316].

7.2. Do Not Reflood (DNR)

Link state packets MUST be encoded in a circuit scope PDU as described in [RFC7356]

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.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, DOI 10.17487/RFC2629, June 1999, <http://www.rfc-editor.org/info/rfc2629>.

[RFC5301] McPherson, D. and N. Shen, "Dynamic Hostname Exchange Mechanism for IS-IS", RFC 5301, DOI 10.17487/RFC5301, October 2008, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc5305>.

[RFC5308] Hopps, C., "Routing IPv6 with IS-IS", RFC 5308, DOI 10.17487/RFC5308, October 2008, <http://www.rfc-editor.org/info/rfc5308>.

[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, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc7981>.

9.2. Informative References

[I-D.ietf-isis-segment-routing-extensions] Previdi, S., Filsfils, C., Bashandy, A., Gredler, H., Litkowski, S., Decraene, B., and j. jefftant@gmail.com, "IS-IS Extensions for Segment Routing", draft-ietf-isis-segment-routing-extensions-11 (work in progress), March 2017.

[I-D.ietf-spring-segment-routing] Filsfils, C., Previdi, S., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", draft-ietf-spring-segment-routing-11 (work in progress), February 2017.

[RFC3277] McPherson, D., "Intermediate System to Intermediate System (IS-IS) Transient Blackhole Avoidance", RFC 3277, DOI 10.17487/RFC3277, April 2002, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc4271>.

[RFC5304] Li, T. and R. Atkinson, "IS-IS Cryptographic Authentication", RFC 5304, DOI 10.17487/RFC5304, October 2008, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc5614>.

[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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc7921>.

Authors' Addresses

Russ White (editor)
LinkedIn

Email: russ@riw.us

Shawn Zandi (editor)
LinkedIn

Email: szandi@linkedin.com