MPLS Working Group
INTERNET-DRAFT
Intended Status: Proposed Standard
Expires: September 14, 2017

Santosh Esale
Raveendra Torvi
Juniper Networks

Luyuan Fang
Microsoft

Luay Jalil
Verizon

March 13, 2017

Fast Reroute for Node Protection in LDP-based LSPs
draft-esale-mpls-ldp-node-frr-05

Abstract

This document describes procedures to support node protection for unicast Label Switched Paths (LSPs) established by Label Distribution Protocol (LDP). In order to protect a node N, the Point of Local Repair (PLR) of N must discover the Merge Points (MPs) of node N such that traffic can be redirected to them in case of node N failure. Redirecting the traffic around the failed node N depends on existing point-to-point LSPs originated from the PLR to the MPs while bypassing the protected node N. The procedures described in this document are topology independent in a sense that they provide node protection in any topology so long as there is a alternate path in the network that avoids the protected node.

Status of this Memo

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

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

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

The list of current Internet-Drafts can be accessed at

http://www.ietf.org/1id-abstracts.html

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

Table of Contents

1. Introduction 2
1.1 Abbreviations 4
3. Merge Point (MP) Discovery 4
4. Constructing Bypass LSPs 5
5. Obtaining Label Mapping from MP 6
6. Forwarding Considerations 6
7. Synergy with node protection in mLDP 7
8. Security Considerations 7
9. IANA Considerations 7
10. Acknowledgements 7
11. Normative References 7
12. Informative References 7
Authors' Addresses 8

1. Introduction

This document describes procedures to support node protection for unicast Label Switched Paths (LSPs) established by Label Distribution Protocol (LDP) [RFC5036]. In order to protect a node N, the Point of Local Repair (PLR) of N must discover the Merge Points (MPs) of node N such that traffic can be redirected to them in case of node N

failure. Redirecting the traffic around the failed node N depends on existing explicit path Point-to-Point (P2P) LSPs originated from the PLR LSR to the MPs while bypassing node N. The procedures to setup these P2P LSPs are outside the scope of this document, but one option is to use RSVP-TE based techniques [RFC3209] to accomplish it. Finally, sending traffic from the PLR to the MPs requires the PLR to obtain FEC-label bindings from the MPs. The procedures described in this document relies on Targeted LDP (tLDP) session [RFC5036] for the PLR to obtain such FEC-Label bindings.

The procedure described in this document assumes the use of platform-wide label space. The procedures for node protection described in this document fall into the category of local protection. The procedures described in this document apply to LDP LSPs bound to either an IPv4 or IPv6 Prefix FEC element. The procedures described in this document are topology independent in a sense that they provide node protection in any topology so long as there is a alternate path in the network that avoids the protected node. Thus these procedures provide topology independent fast reroute.

1.1 Abbreviations

PLR: Point of Local Repair - the LSR that redirects the traffic to one or more Merge Point LSRs.

MP: Merge Point. Any LSR on the LDP-signaled (multi-point to point) LSP, provided that the path from that LSR to the egress of that LSP is not affected by the failure of the protected node.

tLDP: A targeted LDP session is an LDP session between non-directly connected LSRs, established using the LDP extended discovery mechanism.

FEC: Forwarding equivalence class.

IGP: Interior Gateway Protocol.

BR: Border Router.

3. Merge Point (MP) Discovery

For a given LSP that traverses the PLR, the protected node N, and a particular neighbor of the protected node, we'll refer to this neighbor as the "next next-hop". Note that from the PLR's perspective the protected node N is the next hop for the FEC associated with that LSP. Likewise, from the protected node's perspective the next next-hop is the next hop for that FEC. If for a given <LSP, PLR, N> triplet the next next-hop is in the same routing subdomain (area) as the PLR, then that next next-hop acts as the MP for that triplet. For a given LSP traversing a PLR and the node protected by the PLR, the PLR discovers its next next-hops (MPs) that are in the same routing subdomain (IGP area) as the PLR from IGP shortest path first (SPF) calculations. The discovery of next next-hop, depending on an implementation, may not involve any additional SPF, above and beyond what will be needed by either ISIS or OSPF anyway, as the next next-hop, just like the next-hop, is a by-product of SPF computation.

Also, the PLR may discover all possible MPs from either its traffic engineering database or link state database. Some implementations MAY need appropriate configuration to populate the traffic engineering database. The traffic engineering database is populated by routing protocols such as ISIS and OSPF or configured statically.

If for a given <LSP, PLR, N> triplet the node protected by the PLR is an Border Router (BR), then the PLR and the next next-hop may end up in different routing subdomain. This could happen when an LSP

traversing the PLR and the protected node does not terminate in the same routing subdomain as the PLR. In this situation the PLR may not be able to determine the next next-hop from shortest path first (SPF) calculations, and thus may not be able to use the next next-hop as the MP. In this scenario the PLR uses an "alternative" BR as the MP, where an alternative BR is defined as follows. For a given LSP that traverses the PLR and the (protected) BR, an alternative BR is defined as any BR that advertises into PLR's own routing subdomain reachability to the FEC associated with the LSP.

Note that even if a PLR protects an BR, for some of the LSPs traversing the PLR and the BR, the next next-hops may be in the same routing subdomain as the PLR, in which case these next next-hops act as MPs for these LSPs. Note that even if the protected node is not an BR, if an LSP traversing the PLR and the protected node does not terminate in the same routing subdomain as the PLR, then for this LSP the PLR MAY use an alternative BR (as defined earlier), rather than the next next-hop as the MP. When there are several candidate BRs for alternative BR, the LSR MUST select one BR. The algorithm used for the alternative BR selection is a local matter but one option is to select the BR per FEC based on shortest path from PLR to the BR.

4. Constructing Bypass LSPs

As mentioned before, redirecting traffic around the failed node N depends on existing explicit path Point-to-Point (P2P) LSPs originated from the PLR to the MPs while bypassing node N. Let's refer to these LSPs as "bypass LSPs". While the procedures to signal these bypass LSPs are outside the scope of this document, this document assumes use of RSVP-TE LSPs [RFC3209] to accomplish it. Once a PLR that protects a given node N discovers the set of MPs associated with itself and the protected node, at the minimum the PLR MUST (automatically) establish bypass LSPs to all these MPs. The bypass LSPs MUST be established prior to the failure of the protected node.

One could observe that if the protected node is not an BR and the PLR does not use alternative BR(s) as MP(s), then the set of all the IGP neighbors of the protected node forms a superset of the MPs. Thus it would be sufficient for the PLR to establish bypass LSPs with all the IGP neighbors of the protected node, even though some of these neighbors may not be MPs for any of the LSPs traversing the PLR and the protected node.

The bypass LSPs MUST avoid traversing the protected node, which means that the bypass LSPs are explicitly routed LSPs. Of course, using

RSVP-TE to establish bypass LSPs allows these LSPs to be explicitly routed. As a given router may act as an MP for more than one LSP traversing the PLR, the protected node, and the MP, the same bypass LSP will be used to protect all those LSPs.

5. Obtaining Label Mapping from MP

As mentioned before, sending traffic from the PLR to the MPs requires the PLR to obtain FEC-label bindings from the MPs. The solution described in this document relies on Targeted LDP (tLDP) session [RFC5036] for the PLR to obtain such mappings. Specifically, for a given PLR and the node protected by this PLR, at the minimum the PLR MUST (automatically) establish tLDP with all the MPs associated with this PLR and the protected node. These tLDP sessions MUST be established prior to the failure of the protected node. One could observe that if the protected node is not an BR and the PLR does not use alternative BR(s) as MP(s), then the set of all the IGP neighbors of the protected node forms a superset of the MPs. Thus it will be sufficient for the PLR to (automatically) establish tLDP session with all the IGP neighbors of the protected node - except the PLR - that are in the same area as the PLR, even though some of these neighbors may not be MPs for any of the LSPs traversing the PLR and the protected node.

At the minimum for a given tLDP peer the PLR MUST obtain FEC-label mapping for the FEC(s) for which the peer acts as an MP. The PLR MUST obtain this mapping before the failure of the protected node. To obtain this mapping for only these FECs and no other FECs that the peer may maintain, the PLR SHOULD rely on the LDP Downstream on Demand (DoD) procedures [RFC5036]. Otherwise, without relying on the DoD procedures, the PLR may end up receiving from a given tLDP peer FEC-label mappings for all the FECs maintained by the peer, even if the peer does not act as an MP for some of these FECs. If the LDP DoD procedures are not used, then for the purpose of the procedures specified in this draft the only label mappings that SHOULD be exchanged are for the Prefix FEC elements whose PreLen value is either 32 (IPv4), or 128 (IPv6); label mappings for the Prefix FEC elements with any other PreLen value SHOULD NOT be exchanged.

When a PLR has one or more BRs acting as MPs, the PLR MAY use the procedures specified in [draft-ietf-mpls-app-aware-tldp] to limit the set of FEC-label mappings received from non-BR MPs to only the mappings for the FECs associated with the LSPs that terminate in the PLR’s own routing subdomain (area).

6. Forwarding Considerations

When a PLR detects failure of the protected node then rather than

swapping an incoming label with a label that the PLR received from the protected node, the PLR swaps the incoming label with the label that the PLR receives from the MP, and then pushes the label associated with the bypass LSP to that MP.

To minimize micro-loop during the IGP global convergence PLR may continue to use the bypass LSP during network convergence by adding small delay before switching to a new path.

7. Synergy with node protection in mLDP

Both the bypass LSPs and tLDP sessions described in this document could also be used for the purpose of mLDP node protection, as described in [draft-ietf-mpls-mldp-node-protection].

8. Security Considerations

The same security considerations apply as those for the base LDP specification, as described in [RFC5036].

9. IANA Considerations

This document introduces no new IANA Considerations.

10. Acknowledgements

We are indebted to Yakov Rekhter for many discussions on this topic. We like to thank Hannes Gredler, Aman Kapoor, Minto Jeyananth, Eric Rosen, Vladimir Blazhkun and Loa Andersson for through review of this document.

11. Normative References

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

[RFC3209] D. Awduche, et al., "RSVP-TE: Extensions to RSVP for LSP Tunnels", RFC3209, Decembet 2001.

[RFC5036] Andersson, L., Minei, I., and B. Thomas, "LDP Specification", RFC 5036, October 2007.

[draft-ietf-mpls-app-aware-tldp] Esale, S., et al.,"Application-aware Targeted LDP", draft-esale-mpls-app-aware-tldp, work in progress.

12. Informative References

[RFC7715], IJ. Wijnands, et al., "Multipoint LDP (mLDP) Node Protection", RFC7715, January 2016.

Authors' Addresses

Santosh Esale
Juniper Networks
EMail: sesale@juniper.net

Raveendra Torvi
Juniper Networks
EMail: rtorvi@juniper.net

Luyuan Fang
Microsoft
Email: lufang@microsoft.com

Luay Jalil
Verizon
Email: luay.jalil@verizon.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 13, 2018

B. Decraene
Orange
S. Litkowski
Orange Business Service
H. Gredler
RtBrick Inc
A. Lindem
Cisco Systems
P. Francois

C. Bowers
Juniper Networks, Inc.
December 10, 2017

SPF Back-off algorithm for link state IGPs
draft-ietf-rtgwg-backoff-algo-07

Abstract

This document defines a standard algorithm to back-off link-state IGP Shortest Path First (SPF) computations.

Having one standard algorithm improves interoperability by reducing the probability and/or duration of transient forwarding loops during the IGP convergence when the IGP reacts to multiple temporally close IGP events.

Requirements Language

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

Status of This Memo

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

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

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

This Internet-Draft will expire on June 13, 2018.

Table of Contents

1. Introduction 2
2. High level goals 3
3. Definitions and parameters 4
4. Principles of SPF delay algorithm 5
5. Specification of the SPF delay state machine 5
5.1. States 6
5.2. Timers 6
5.3. States Transitions 6
5.4. FSM Events 7
6. Parameters 9
7. Partial Deployment 10
8. Impact on micro-loops 10
9. IANA Considerations 11
10. Security considerations 11
11. Acknowledgements 11
12. References 11
12.1. Normative References 11
12.2. Informative References 11
Authors' Addresses 12

1. Introduction

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

In general, when the network is stable, there is a desire to compute a new Shortest Path First (SPF) as soon as a failure is detected in order to quickly route around the failure. However, when the network is experiencing multiple temporally close failures over a short period of time, there is a conflicting desire to limit the frequency of SPF computations. Indeed, this allows a reduction in control plane resources used by IGPs and all protocols/subsystems reacting on the attendant route change, such as LDP [RFC5036], RSVP-TE [RFC3209], BGP [RFC4271], Fast ReRoute computations (e.g. Loop Free Alternates (LFA) [RFC5286], FIB updates... This also reduces the churn on routers and in the network and, in particular, reduces the side effects such as micro-loops [RFC5715] that ensue during IGP convergence.

To allow for this, IGPs implement an SPF back-off algorithm. However, different implementations have choosen different algorithms. Hence, in a multi-vendor network, it’s not possible to ensure that all routers trigger their SPF computation after the same delay. This situation increases the average and maximum differential delay between routers completing their SPF computation. It also increases the probability that different routers compute their FIBs based on different LSDB versions. Both factors increase the probability and/or duration of micro-loops as discussed in Section 8.

To allow multi-vendor networks to have all routers delay their SPF computations for the same duration, this document specifies a standard algorithm. Optionally, implementations may also offer alternative algorithms.

2. High level goals

The high level goals of this algorithm are the following:

- Very fast convergence for a single event (e.g., link failure).
- Paced fast convergence for multiple temporally close IGP events while IGP stability is considered acceptable.
- Delayed convergence when IGP stability is problematic. This will allow the IGP and related processes to conserve resources during the period of instability.
- Always try to avoid different SPF_DELAY timers values across different routers in the area/level. Even though not all routers will receive IGP messages at the same time, due to differences both in the distance from the originator of the IGP event and in flooding implementations.

3. Definitions and parameters

IGP events: The reception or origination of an IGP LSDB change requiring a new routing table computation. Examples are a topology change, a prefix change, a metric change on a link or prefix... Note that locally triggering a routing table computation is not considered as an IGP event since other IGP routers are unaware of this occurrence.

Routing table computation: Computation of the routing table, by the IGP, using the IGP LSDB. No distinction is made between the type of computation performed. e.g., full SPF, incremental SPF, Partial Route Computation (PRC). The type of computation is a local consideration. This document may interchangeably use the terms routing table computation and SPF computation.

SPF_DELAY: The delay between the first IGP event triggering a new routing table computation and the start of that routing table computation. It can take the following values:

INITIAL_SPF_DELAY: A very small delay to quickly handle link failure, e.g., 0 milliseconds.

SHORT_SPF_DELAY: A small delay to have a fast convergence in case of a single failure (node, SRLG..), e.g., 50-100 milliseconds.

LONG_SPF_DELAY: A long delay when the IGP is unstable, e.g., 2 seconds. Note that this allows the IGP network to stabilize.

TIME_TO_LEARN_INTERVAL: This is the maximum duration typically needed to learn all the IGP events related to a single component failure (e.g., router failure, SRLG failure), e.g., 1 second. It’s mostly dependent on failure detection time variation between all routers that are adjacent to the failure. Additionally, it may depend on the different IGP implementations/parameters across the network, related to origination and flooding of their link state advertisements.

HOLDDOWN_INTERVAL: The time required with no received IGP events before considering the IGP to be stable again and allowing the SPF_DELAY to be restored to INITIAL_SPF_DELAY. e.g., 3 seconds. The HOLDDOWN_INTERVAL MUST be defaulted or configured to be longer than the TIME_TO_LEARN_INTERVAL.

4. Principles of SPF delay algorithm

For this first IGP event, we assume that there has been a single simple change in the network which can be taken into account using a single routing computation (e.g., link failure, prefix (metric) change) and we optimize for very fast convergence, delaying the routing computation by INITIAL_SPF_DELAY. Under this assumption, there is no benefit in delaying the routing computation. In a typical network, this is the most common type of IGP event. Hence, it makes sense to optimize this case.

If subsequent IGP events are received in a short period of time (TIME_TO_LEARN_INTERVAL), we then assume that a single component failed, but that this failure requires the knowledge of multiple IGP events in order for IGP routing to converge. Under this assumption, we want fast convergence since this is a normal network situation. However, there is a benefit in waiting for all IGP events related to this single component failure so that the IGP can compute the post-failure routing table in a single route computation. In this situation, we delay the routing computation by SHORT_SPF_DELAY.

If IGP events are still received after TIME_TO_LEARN_INTERVAL from the initial IGP event received in QUIET state, then the network is presumably experiencing multiple independent failures. In this case, while waiting for network stability, the computations are delayed for a longer time represented by LONG_SPF_DELAY. This SPF delay is kept until no IGP events are received for HOLDDOWN_INTERVAL.

Note that previously implemented SPF delay algorithms counted the number of SPF computations. However, as all routers may receive the IGP events at different times, we cannot assume that all routers will perform the same number of SPF computations or that they will schedule them at the same time. For example, assuming that the SPF delay is 50 ms, router R1 may receive 3 IGP events (E1, E2, E3) in those 50 ms and hence will perform a single routing computation. While another router R2 may only receive 2 events (E1, E2) in those 50 ms and hence will schedule another routing computation when receiving E3. That’s why this document uses a time (TIME_TO_LEARN_INTERVAL) from the initial event detection/reception as opposed to counting the number of SPF computations to determine when the IGP is unstable.

5. Specification of the SPF delay state machine

5.1. States

This section describes the state machine. The naming and semantics of each state corresponds directly to the SPF delay used for IGP events received in that state. Three states are defined:

QUIET: This is the initial state, when no IGP events have occured for at least HOLDDOWN_INTERVAL since the previous routing table computation. The state is meant to handle link failures very quickly.

SHORT_WAIT: State entered when an IGP event has been received in QUIET state. This state is meant to handle single component failure requiring multiple IGP events (e.g., node, SRLG).

LONG_WAIT: State reached after TIME_TO_LEARN_INTERVAL. In other words, state reached after TIME_TO_LEARN_INTERVAL in state SHORT_WAIT. This state is meant to handle multiple independent component failures during periods of IGP instability.

5.2. Timers

SPF_TIMER: The Finite State Machine (FSM) abstract timer that uses the computed SPF delay. Upon expiration, the Route Table Computation (as defined in Section 3) is performed.

HOLDDOWN_TIMER: The Finite State Machine (FSM) abstract timer that is (re)started whan an IGP event is received and set to HOLDDOWN_INTERVAL. Upon expiration, the FSM is moved to the QUIET state.

LEARN_TIMER: The Finite State Machine (FSM) abstract timer that is started when an IGP event is recevied while the FSM is in the QUIET state. Upon expiration, the FSM is moved to the LONG_WAIT state.

5.3. States Transitions

The FSM is initialized to the QUIET state with all three timers timers (SPF_TIMER, HOLDDOWN_TIMER, LEARN_TIMER) deactivated.

The events which may change the FSM states are an IGP event or the expiration of one timer (SPF_TIMER, HOLDDOWN_TIMER, LEARN_TIMER).

The following diagram briefly describes the state transitions.

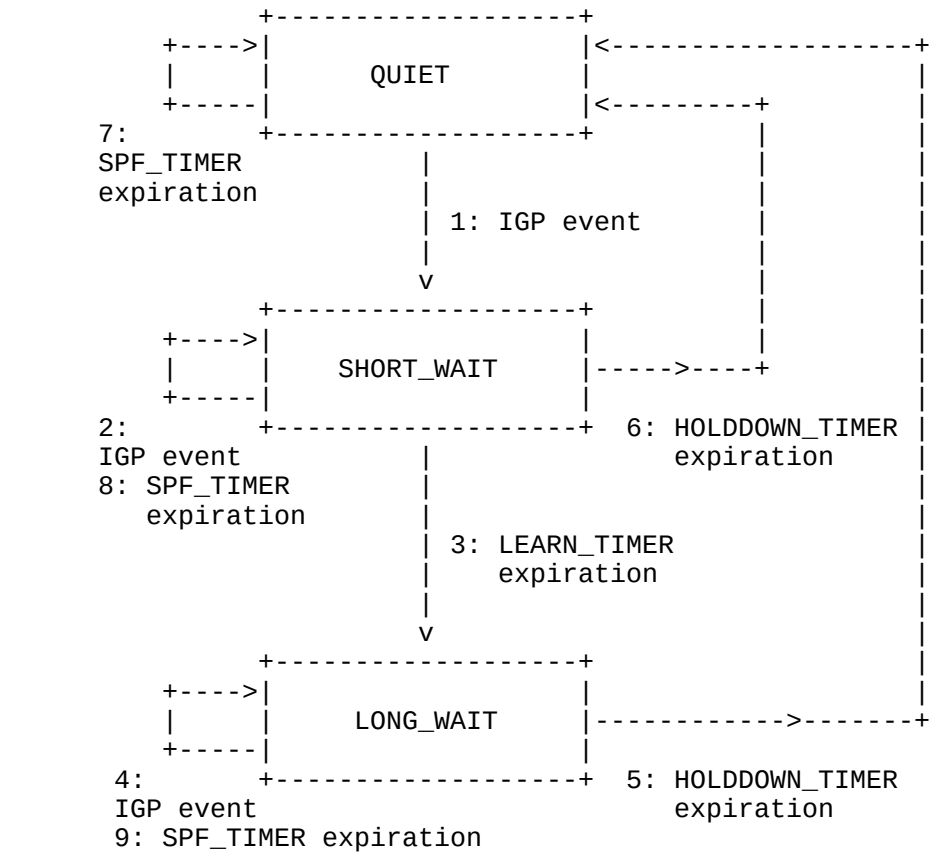


Figure 1: State Machine

5.4. FSM Events

This section describes the events and the actions performed in response.

Transition 1: IGP event, while in QUIET_STATE.

Actions on event 1:

- If SPF_TIMER is not already running, start it with value INITIAL_SPF_DELAY.
- Start LEARN_TIMER with TIME_TO_LEARN_INTERVAL.
- Start HOLDDOWN_TIMER with HOLDDOWN_INTERVAL.

- Transition to SHORT_WAIT state.

Transition 2: IGP event, while in SHORT_WAIT.

Actions on event 2:

- Reset HOLDDOWN_TIMER to HOLDDOWN_INTERVAL.
- If SPF_TIMER is not already running, start it with value SHORT_SPF_DELAY.
- Remain in current state.

Transition 3: LEARN_TIMER expiration.

Actions on event 3:

- Transition to LONG_WAIT state.

Transition 4: IGP event, while in LONG_WAIT.

Actions on event 4:

- Reset HOLDDOWN_TIMER to HOLDDOWN_INTERVAL.
- If SPF_TIMER is not already running, start it with value LONG_SPF_DELAY.
- Remain in current state.

Transition 5: HOLDDOWN_TIMER expiration, while in LONG_WAIT.

Actions on event 5:

- Transition to QUIET state.

Transition 6: HOLDDOWN_TIMER expiration, while in SHORT_WAIT.

Actions on event 6:

- Deactivate LEARN_TIMER.
- Transition to QUIET state.

Transition 7: SPF_TIMER expiration, while in QUIET.

Actions on event 7:

- Compute SPF.
- Remain in current state.

Transition 8: SPF_TIMER expiration, while in SHORT_WAIT.

Actions on event 8:

- Compute SPF.
- Remain in current state.

Transition 9: SPF_TIMER expiration, while in LONG_WAIT.

Actions on event 9:

- Compute SPF.
- Remain in current state.

6. Parameters

All the parameters MUST be configurable [I-D.ietf-isis-yang-isis-cfg] [I-D.ietf-ospf-yang] at the protocol instance granularity. They MAY be configurable at the area/level granularity. All the delays (INITIAL_SPF_DELAY, SHORT_SPF_DELAY, LONG_SPF_DELAY, TIME_TO_LEARN_INTERVAL, HOLDDOWN_INTERVAL) SHOULD be configurable at the millisecond granularity. They MUST be configurable at least at the tenth of second granularity. The configurable range for all the parameters SHOULD at least be from 0 milliseconds to 60 seconds.

This document does not propose default values for the parameters because these values are expected to be context dependent. Implementations are free to propose their own default values.

In order to satisfy the goals stated in Section 2, operators are RECOMMENDED to configure delay intervals such that SPF_INITIAL_DELAY <= SPF_SHORT_DELAY and SPF_SHORT_DELAY <= SPF_LONG_DELAY.

When setting (default) values, one SHOULD consider the customers and their application requirements, the computational power of the

routers, the size of the network, and, in particular, the number of IP prefixes advertised in the IGP, the frequency and number of IGP events, the number of protocols reactions/computations triggered by IGP SPF (e.g., BGP, PCEP, Traffic Engineering CSPF, Fast ReRoute computations).

Note that some or all of these factors may change over the life of the network. In case of doubt, it's RECOMMENDED to play it safe and start with safe, i.e., longer timers.

For the standard algorithm to be effective in mitigating micro-loops, it is RECOMMENDED that all routers in the IGP domain, or at least all the routers in the same area/level, have exactly the same configured values.

7. Partial Deployment

In general, the SPF delay algorithm is only effective in mitigating micro-loops if it is deployed, with the same parameters, on all routers, in the IGP domain or, at least, all routers in an IGP area/level. The impact of partial deployment is based on the particular event, topology, and the SPF algorithm(s) used on other routers in the IGP area/level. In cases where the previous SPF algorithm was implemented uniformly, partial deployment will increase the frequency and duration of micro-loops. Hence, it is RECOMMENDED that all routers in the IGP domain or at least within the same area/level be migrated to the SPF algorithm described herein at roughly the same time.

Note that this is not a new consideration as over times, network operators have changed SPF delay parameters in order to accommodate new customer requirements for fast convergence, as permitted by new software and hardware. They may also have progressively replaced an implementation with a given SPF delay algorithm by another implementation with a different one.

8. Impact on micro-loops

Micro-loops during IGP convergence are due to a non-synchronized or non-ordered update of the forwarding information tables (FIB) [RFC5715] [RFC6976] [I-D.ietf-rtgwg-spf-uloop-pb-statement]. FIBs are installed after multiple steps such as flooding of the IGP event across the network, SPF wait time, SPF computation, FIB distribution across line cards, and FIB update. This document only addresses the first contribution. This standardized procedure reduces the probability and/or duration of micro-loops when IGPs experience multiple temporally close events. It does not prevent all micro-loops. However, it is beneficial and is less complex and costly to

implement when compared to full solutions such as [RFC5715] or [RFC6976].

9. IANA Considerations

No IANA actions required.

10. Security considerations

The algorithm presented in this document does not compromise IGP security. An attacker having the ability to generate IGP events would be able to delay the IGP convergence time. The LONG_SPF_DELAY state may help mitigate the effects of Denial-of-Service (DOS) attacks generating many IGP events.

11. Acknowledgements

We would like to acknowledge Les Ginsberg, Uma Chunduri, Mike Shand and Alexander Vainshtein for the discussions and comments related to this document.

12. References

12.1. Normative References

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

12.2. Informative References

[I-D.ietf-isis-yang-isis-cfg]
Litkowski, S., Yeung, D., Lindem, A., Zhang, Z., and L. Lhotka, "YANG Data Model for IS-IS protocol", draft-ietf-isis-yang-isis-cfg-19 (work in progress), November 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-09 (work in progress), October 2017.

[I-D.ietf-rtgwg-spf-uloop-pb-statement]
Litkowski, S., Decraene, B., and M. Horneffer, "Link State protocols SPF trigger and delay algorithm impact on IGP micro-loops", draft-ietf-rtgwg-spf-uloop-pb-statement-05 (work in progress), December 2017.

[ISO10589-Second-Edition]
International Organization for Standardization, "Intermediate system to Intermediate system intra-domain routeing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode Network Service (ISO 8473)", ISO/IEC 10589:2002, Second Edition, Nov 2002.

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

[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>.

[RFC4271] Rekhter, Y., Ed., Li, T., Ed., and S. Hares, Ed., "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, DOI 10.17487/RFC4271, January 2006, <https://www.rfc-editor.org/info/rfc4271>.

[RFC5036] Andersson, L., Ed., Minei, I., Ed., and B. Thomas, Ed., "LDP Specification", RFC 5036, DOI 10.17487/RFC5036, October 2007, <https://www.rfc-editor.org/info/rfc5036>.

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

[RFC5715] Shand, M. and S. Bryant, "A Framework for Loop-Free Convergence", RFC 5715, DOI 10.17487/RFC5715, January 2010, <https://www.rfc-editor.org/info/rfc5715>.

[RFC6976] Shand, M., Bryant, S., Previdi, S., Filsfils, C., Francois, P., and O. Bonaventure, "Framework for Loop-Free Convergence Using the Ordered Forwarding Information Base (oFIB) Approach", RFC 6976, DOI 10.17487/RFC6976, July 2013, <https://www.rfc-editor.org/info/rfc6976>.

Authors' Addresses

Bruno Decraene
Orange

Email: bruno.decraene@orange.com

Stephane Litkowski
Orange Business Service

Email: stephane.litkowski@orange.com

Hannes Gredler
RtBrick Inc

Email: hannes@rtbrick.com

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

Email: acee@cisco.com

Pierre Francois

Email: pfrpfr@gmail.com

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

Email: cbowers@juniper.net

Routing Working Group
Internet-Draft
Intended status: Informational
Expires: May 2, 2018

F. Baker
C. Bowers
Juniper Networks
J. Linkova
Google
October 29, 2017

Enterprise Multihoming using Provider-Assigned Addresses without Network Prefix Translation: Requirements and Solution draft-ietf-rtgwg-enterprise-pa-multihoming-02

Abstract

Connecting an enterprise site to multiple ISPs using provider-assigned addresses is difficult without the use of some form of Network Address Translation (NAT). Much has been written on this topic over the last 10 to 15 years, but it still remains a problem without a clearly defined or widely implemented solution. Any multihoming solution without NAT requires hosts at the site to have addresses from each ISP and to select the egress ISP by selecting a source address for outgoing packets. It also requires routers at the site to take into account those source addresses when forwarding packets out towards the ISPs.

This document attempts to define a complete solution to this problem. It covers the behavior of routers to forward traffic taking into account source address, and it covers the behavior of host to select appropriate source addresses. It also covers any possible role that routers might play in providing information to hosts to help them select appropriate source addresses. In the process of exploring potential solutions, this documents also makes explicit requirements for how the solution would be expected to behave from the perspective of an enterprise site network administrator .

Status of This Memo

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

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

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

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

This Internet-Draft will expire on May 2, 2018.

Table of Contents

1. Introduction 3
2. Enterprise Multihoming Requirements 6
2.1. Simple ISP Connectivity with Connected SERs 6
2.2. Simple ISP Connectivity Where SERs Are Not Directly Connected 7
2.3. Enterprise Network Operator Expectations 8
2.4. More complex ISP connectivity 11
2.5. ISPs and Provider-Assigned Prefixes 13
2.6. Simplified Topologies 14
3. Generating Source-Prefix-Scoped Forwarding Tables 14
4. Mechanisms For Hosts To Choose Good Source Addresses In A Multihomed Site 21
4.1. Source Address Selection Algorithm on Hosts 23
4.2. Selecting Source Address When Both Uplinks Are Working . 26
4.2.1. Distributing Address Selection Policy Table with DHCPv6 26
4.2.2. Controlling Source Address Selection With Router Advertisements 26
4.2.3. Controlling Source Address Selection With ICMPv6 . . 28
4.2.4. Summary of Methods For Controlling Source Address Selection To Implement Routing Policy 30
4.3. Selecting Source Address When One Uplink Has Failed . . . 31
4.3.1. Controlling Source Address Selection With DHCPv6 . . 32
4.3.2. Controlling Source Address Selection With Router Advertisements 33
4.3.3. Controlling Source Address Selection With ICMPv6 . . 34

4.3.4. Summary Of Methods For Controlling Source Address Selection On The Failure Of An Uplink 34
4.4. Selecting Source Address Upon Failed Uplink Recovery . . 35
4.4.1. Controlling Source Address Selection With DHCPv6 . . 35
4.4.2. Controlling Source Address Selection With Router Advertisements 35
4.4.3. Controlling Source Address Selection With ICMP . . . 36
4.4.4. Summary Of Methods For Controlling Source Address Selection Upon Failed Uplink Recovery 36
4.5. Selecting Source Address When All Uplinks Failed 37
4.5.1. Controlling Source Address Selection With DHCPv6 . . 37
4.5.2. Controlling Source Address Selection With Router Advertisements 37
4.5.3. Controlling Source Address Selection With ICMPv6 . . 38
4.5.4. Summary Of Methods For Controlling Source Address Selection When All Uplinks Failed 38
4.6. Summary Of Methods For Controlling Source Address Selection 38
4.7. Other Configuration Parameters 40
4.7.1. DNS Configuration 40
5. Other Solutions 41
5.1. Shim6 41
5.2. IPv6-to-IPv6 Network Prefix Translation 42
6. IANA Considerations 42
7. Security Considerations 42
7.1. Privacy Considerations 42
8. Acknowledgements 42
9. References 42
9.1. Normative References 42
9.2. Informative References 44
Appendix A. Change Log 47
Authors' Addresses 47

1. Introduction

Site multihoming, the connection of a subscriber network to multiple upstream networks using redundant uplinks, is a common enterprise architecture for improving the reliability of its Internet connectivity. If the site uses provider-independent (PI) addresses, all traffic originating from the enterprise can use source addresses from the PI address space. Site multihoming with PI addresses is commonly used with both IPv4 and IPv6, and does not present any new technical challenges.

It may be desirable for an enterprise site to connect to multiple ISPs using provider-assigned (PA) addresses, instead of PI addresses. Multihoming with provider-assigned addresses is typically less expensive for the enterprise relative to using provider-independent

addresses. PA multihoming is also a practice that should be facilitated and encouraged because it does not add to the size of the Internet routing table, whereas PI multihoming does. Note that PA is also used to mean "provider-aggregatable". In this document we assume that provider-assigned addresses are always provider-aggregatable.

With PA multihoming, for each ISP connection, the site is assigned a prefix from within an address block allocated to that ISP by its National or Regional Internet Registry. In the simple case of two ISPs (ISP-A and ISP-B), the site will have two different prefixes assigned to it (prefix-A and prefix-B). This arrangement is problematic. First, packets with the "wrong" source address may be dropped by one of the ISPs. In order to limit denial of service attacks using spoofed source addresses, BCP38 [RFC2827] recommends that ISPs filter traffic from customer sites to only allow traffic with a source address that has been assigned by that ISP. So a packet sent from a multihomed site on the uplink to ISP-B with a source address in prefix-A may be dropped by ISP-B.

However, even if ISP-B does not implement BCP38 or ISP-B adds prefix-A to its list of allowed source addresses on the uplink from the multihomed site, two-way communication may still fail. If the packet with source address in prefix-A was sent to ISP-B because the uplink to ISP-A failed, then if ISP-B does not drop the packet and the packet reaches its destination somewhere on the Internet, the return packet will be sent back with a destination address in prefix-A. The return packet will be routed over the Internet to ISP-A, but it will not be delivered to the multihomed site because its link with ISP-A has failed. Two-way communication would require some arrangement for ISP-B to advertise prefix-A when the uplink to ISP-A fails.

Note that the same may be true with a provider that does not implement BCP 38, if his upstream provider does, or has no corresponding route. The issue is not that the immediate provider implements ingress filtering; it is that someone upstream does, or lacks a route.

With IPv4, this problem is commonly solved by using [RFC1918] private address space within the multi-homed site and Network Address Translation (NAT) or Network Address/Port Translation (NAPT) on the uplinks to the ISPs. However, one of the goals of IPv6 is to eliminate the need for and the use of NAT or NAPT. Therefore, requiring the use of NAT or NAPT for an enterprise site to multihome with provider-assigned addresses is not an attractive solution.

[RFC6296] describes a translation solution specifically tailored to meet the requirements of multi-homing with provider-assigned IPv6 addresses. With the IPv6-to-IPv6 Network Prefix Translation (NPTv6) solution, within the site an enterprise can use Unique Local Addresses [RFC4193] or the prefix assigned by one of the ISPs. As traffic leaves the site on an uplink to an ISP, the source address gets translated to an address within the prefix assigned by the ISP on that uplink in a predictable and reversible manner. [RFC6296] is currently classified as Experimental, and it has been implemented by several vendors. See Section 5.2, for more discussion of NPTv6.

This document defines routing requirements for enterprise multihoming using provider-assigned IPv6 addresses. We have made no attempt to write these requirements in a manner that is agnostic to potential solutions. Instead, this document focuses on the following general class of solutions.

Each host at the enterprise has multiple addresses, at least one from each ISP-assigned prefix. Each host, as discussed in Section 4.1 and [RFC6724], is responsible for choosing the source address applied to each packet it sends. A host SHOULD be able respond dynamically to the failure of an uplink to a given ISP by no longer sending packets with the source address corresponding to that ISP. Potential mechanisms for the communication of changes in the network to the host are Neighbor Discovery Router Advertisements, DHCPv6, and ICMPv6.

The routers in the enterprise network are responsible for ensuring that packets are delivered to the "correct" ISP uplink based on source address. This requires that at least some routers in the site network are able to take into account the source address of a packet when deciding how to route it. That is, some routers must be capable of some form of Source Address Dependent Routing (SADR), if only as described in [RFC3704]. At a minimum, the routers connected to the ISP uplinks (the site exit routers or SERs) must be capable of Source Address Dependent Routing. Expanding the connected domain of routers capable of SADR from the site exit routers deeper into the site network will generally result in more efficient routing of traffic with external destinations.

The document first looks in more detail at the enterprise networking environments in which this solution is expected to operate. It then discusses existing and proposed mechanisms for hosts to select the source address applied to packets. Finally, it looks at the requirements for routing that are needed to support these enterprise network scenarios and the mechanisms by which hosts are expected to select source addresses dynamically based on network state.

2. Enterprise Multihoming Requirements

2.1. Simple ISP Connectivity with Connected SERs

We start by looking at a scenario in which a site has connections to two ISPs, as shown in Figure 1. The site is assigned the prefix 2001:db8:0:a000::/52 by ISP-A and prefix 2001:db8:0:b000::/52 by ISP-B. We consider three hosts in the site. H31 and H32 are on a LAN that has been assigned subnets 2001:db8:0:a010::/64 and 2001:db8:0:b010::/64. H31 has been assigned the addresses 2001:db8:0:a010::31 and 2001:db8:0:b010::31. H32 has been assigned 2001:db8:0:a010::32 and 2001:db8:0:b010::32. H41 is on a different subnet that has been assigned 2001:db8:0:a020::/64 and 2001:db8:0:b020::/64.

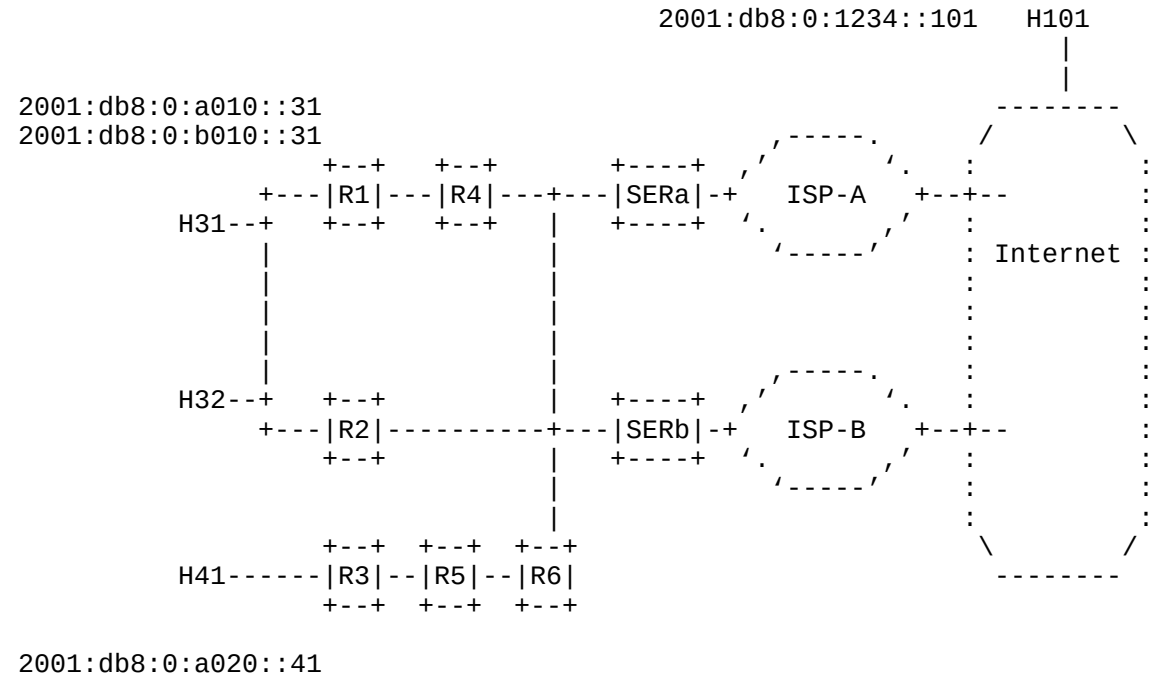


Figure 1: Simple ISP Connectivity With Connected SERs

We refer to a router that connects the site to an ISP as a site edge router(SER). Several other routers provide connectivity among the internal hosts (H31, H32, and H41), as well as connecting the internal hosts to the Internet through SERa and SERb. In this

example SERa and SERb share a direct connection to each other. In Section 2.2, we consider a scenario where this is not the case.

For the moment, we assume that the hosts are able to make good choices about which source addresses through some mechanism that doesn't involve the routers in the site network. Here, we focus on primary task of the routed site network, which is to get packets efficiently to their destinations, while sending a packet to the ISP that assigned the prefix that matches the source address of the packet. In Section 4, we examine what role the routed network may play in helping hosts make good choices about source addresses for packets.

With this solution, routers will need form of Source Address Dependent Routing, which will be new functionality. It would be useful if an enterprise site does not need to upgrade all routers to support the new SADR functionality in order to support PA multi-homing. We consider if this is possible and what are the tradeoffs of not having all routers in the site support SADR functionality.

In the topology in Figure 1, it is possible to support PA multihoming with only SERa and SERb being capable of SADR. The other routers can continue to forward based only on destination address, and exchange routes that only consider destination address. In this scenario, SERa and SERb communicate source-scoped routing information across their shared connection. When SERa receives a packet with a source address matching prefix 2001:db8:0:b000::/52 , it forwards the packet to SERb, which forwards it on the uplink to ISP-B. The analogous behaviour holds for traffic that SERb receives with a source address matching prefix 2001:db8:0:a000::/52.

In Figure 1, when only SERa and SERb are capable of source address dependent routing, PA multi-homing will work. However, the paths over which the packets are sent will generally not be the shortest paths. The forwarding paths will generally be more efficient as more routers are capable of SADR. For example, if R4, R2, and R6 are upgraded to support SADR, then can exchange source-scoped routes with SERa and SERb. They will then know to send traffic with a source address matching prefix 2001:db8:0:b000::/52 directly to SERb, without sending it to SERa first.

2.2. Simple ISP Connectivity Where SERs Are Not Directly Connected

In Figure 2, we modify the topology slightly by inserting R7, so that SERa and SERb are no longer directly connected. With this topology, it is not enough to just enable SADR routing on SERa and SERb to support PA multi-homing. There are two solutions to ways to enable PA multihoming in this topology.

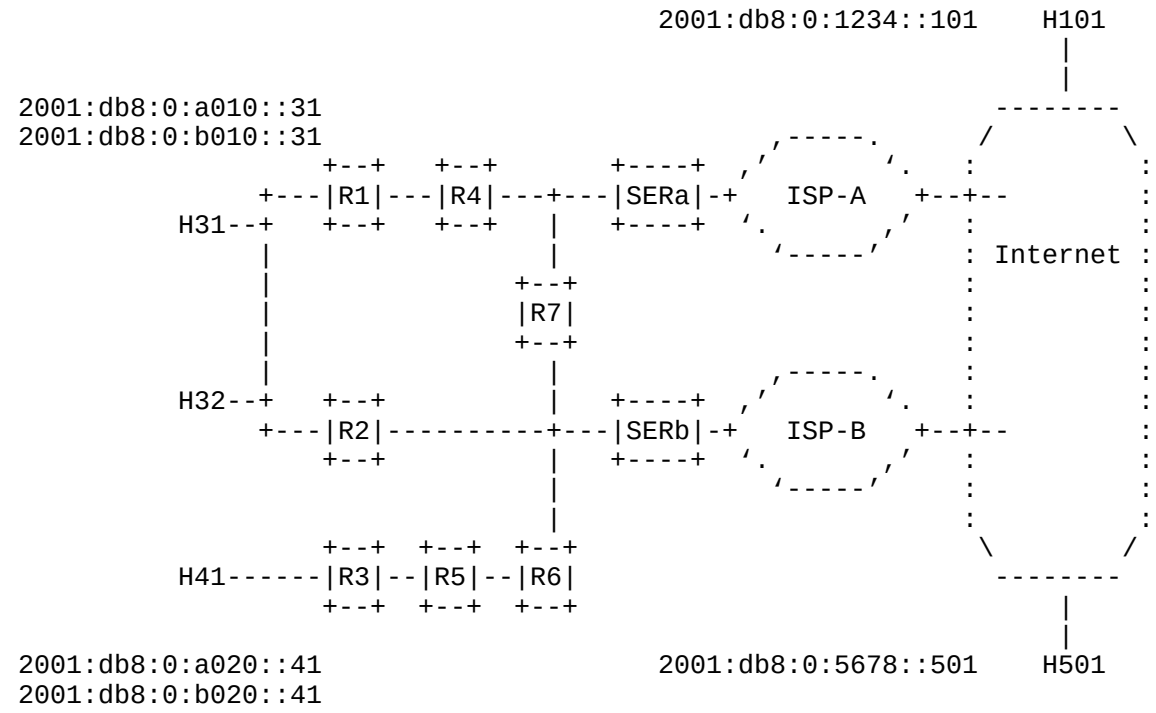


Figure 2: Simple ISP Connectivity Where SERs Are Not Directly Connected

One option is to effectively modify the topology by creating a logical tunnel between SERa and SERb, using GRE for example. Although SERa and SERb are not directly connected physically in this topology, they can be directly connected logically by a tunnel.

The other option is to enable SADR functionality on R7. In this way, R7 will exchange source-scoped routes with SERa and SERb, making the three routers act as a single SADR domain. This illustrates the basic principle that the minimum requirement for the routed site network to support PA multi-homing is having all of the site exit routers be part of a connected SADR domain. Extending the connected SADR domain beyond that point can produce more efficient forwarding paths.

2.3. Enterprise Network Operator Expectations

Before considering a more complex scenario, let’s look in more detail at the reasonably simple multihoming scenario in Figure 2 to understand what can reasonably be expected from this solution. As a

general guiding principle, we assume an enterprise network operator will expect a multihomed network to behave as close as to a single-homed network as possible. So a solution that meets those expectations where possible is a good thing.

For traffic between internal hosts and traffic from outside the site to internal hosts, an enterprise network operator would expect there to be no visible change in the path taken by this traffic, since this traffic does not need to be routed in a way that depends on source address. It is also reasonable to expect that internal hosts should be able to communicate with each other using either of their source addresses without restriction. For example, H31 should be able to communicate with H41 using a packet with S=2001:db8:0:a010::31, D=2001:db8:0:b010::41, regardless of the state of uplink to ISP-B.

These goals can be accomplished by having all of the routers in the network continue to originate normal unscoped destination routes for their connected networks. If we can arrange so that these unscoped destination routes get used for forwarding this traffic, then we will have accomplished the goal of keeping forwarding of traffic destined for internal hosts, unaffected by the multihoming solution.

For traffic destined for external hosts, it is reasonable to expect that traffic with an source address from the prefix assigned by ISP-A to follow the path to that the traffic would follow if there is no connection to ISP-B. This can be accomplished by having SERa originate a source-scoped route of the form (S=2001:db8:0:a000::/52, D=::/0) . If all of the routers in the site support SADR, then the path of traffic exiting via ISP-A can match that expectation. If some routers don’t support SADR, then it is reasonable to expect that the path for traffic exiting via ISP-A may be different within the site. This is a tradeoff that the enterprise network operator may decide to make.

It is important to understand how this multihoming solution behaves when an uplink to one of the ISPs fails. To simplify this discussion, we assume that all routers in the site support SADR. We first start by looking at how the network operates when the uplinks to both ISP-A and ISP-B are functioning properly. SERa originates a source-scoped route of the form (S=2001:db8:0:a000::/52, D=::/0), and SERb is originates a source-scoped route of the form (S=2001:db8:0:b000::/52, D=::/0). These routes are distributed through the routers in the site, and they establish within the routers two set of forwarding paths for traffic leaving the site. One set of forwarding paths is for packets with source address in 2001:db8:0:a000::/52. The other set of forwarding paths is for packets with source address in 2001:db8:0:b000::/52. The normal destination routes which are not scoped to these two source prefixes

play no role in the forwarding. Whether a packet exits the site via SERa or via SERb is completely determined by the source address applied to the packet by the host. So for example, when host H31 sends a packet to host H101 with (S=2001:db8:0:a010::31, D=2001:db8:0:1234::101), the packet will only be sent out the link from SERa to ISP-A.

Now consider what happens when the uplink from SERa to ISP-A fails. The only way for the packets from H31 to reach H101 is for H31 to start using the source address for ISP-B. H31 needs to send the following packet: (S=2001:db8:0:b010::31, D=2001:db8:0:1234::101).

This behavior is very different from the behavior that occurs with site multihoming using PI addresses or with PA addresses using NAT. In these other multi-homing solutions, hosts do not need to react to network failures several hops away in order to regain Internet access. Instead, a host can be largely unaware of the failure of an uplink to an ISP. When multihoming with PA addresses and NAT, existing sessions generally need to be re-established after a failure since the external host will receive packets from the internal host with a new source address. However, new sessions can be established without any action on the part of the hosts.

Another example where the behavior of this multihoming solution differs significantly from that of multihoming with PI address or with PA addresses using NAT is in the ability of the enterprise network operator to route traffic over different ISPs based on destination address. We still consider the fairly simple network of Figure 2 and assume that uplinks to both ISPs are functioning. Assume that the site is multihomed using PA addresses and NAT, and that SERa and SERb each originate a normal destination route for D=::/0, with the route origination dependent on the state of the uplink to the respective ISP.

Now suppose it is observed that an important application running between internal hosts and external host H101 experience much better performance when the traffic passes through ISP-A (perhaps because ISP-A provides lower latency to H101.) When multihoming this site with PI addresses or with PA addresses and NAT, the enterprise network operator can configure SERa to originate into the site network a normal destination route for D=2001:db8:0:1234::/64 (the destination prefix to reach H101) that depends on the state of the uplink to ISP-A. When the link to ISP-A is functioning, the destination route D=2001:db8:0:1234::/64 will be originated by SERa, so traffic from all hosts will use ISP-A to reach H101 based on the longest destination prefix match in the route lookup.

Implementing the same routing policy is more difficult with the PA multihoming solution described in this document since it doesn't use NAT. By design, the only way to control where a packet exits this network is by setting the source address of the packet. Since the network cannot modify the source address without NAT, the host must set it. To implement this routing policy, each host needs to use the source address from the prefix assigned by ISP-A to send traffic destined for H101. Mechanisms have been proposed to allow hosts to choose the source address for packets in a fine grained manner. We will discuss these proposals in Section 4. However, interacting with host operating systems in some manner to ensure a particular source address is chosen for a particular destination prefix is not what an enterprise network administrator would expect to have to do to implement this routing policy.

2.4. More complex ISP connectivity

The previous sections considered two variations of a simple multihoming scenario where the site is connected to two ISPs offering only Internet connectivity. It is likely that many actual enterprise multihoming scenarios will be similar to this simple example. However, there are more complex multihoming scenarios that we would like this solution to address as well.

It is fairly common for an ISP to offer a service in addition to Internet access over the same uplink. Two variation of this are reflected in Figure 3. In addition to Internet access, ISP-A offers a service which requires the site to access host H51 at 2001:db8:0:5555::51. The site has a single physical and logical connection with ISP-A, and ISP-A only allows access to H51 over that connection. So when H32 needs to access the service at H51 it needs to send packets with (S=2001:db8:0:a010::32, D=2001:db8:0:5555::51) and those packets need to be forward out the link from SERa to ISP-A.

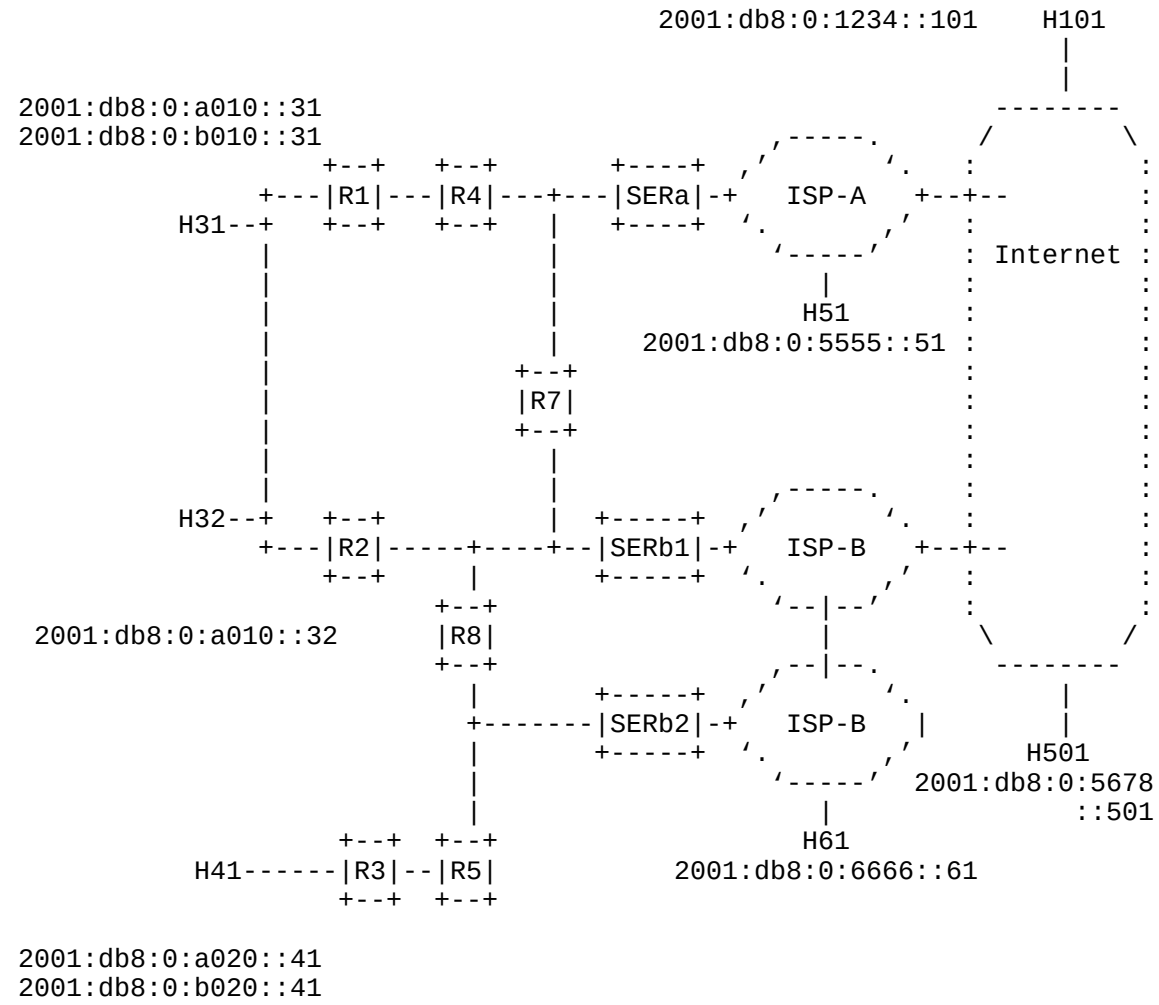


Figure 3: Internet access and services offered by ISP-A and ISP-B

ISP-B illustrates a variation on this scenario. In addition to Internet access, ISP-B also offers a service which requires the site to access host H61. The site has two connections to two different parts of ISP-B (shown as SERb1 and SERb2 in Figure 3). ISP-B expects Internet traffic to use the uplink from SERb1, while it expects it expects traffic destined for the service at H61 to use the uplink from SERb2. For either uplink, ISP-B expects the ingress traffic to have a source address matching the prefix it assigned to the site, 2001:db8:0:b000::/52.

As discussed before, we rely completely on the internal host to set the source address of the packet properly. In the case of a packet sent by H31 to access the service in ISP-B at H61, we expect the packet to have the following addresses: (S=2001:db8:0:b010::31, D=2001:db8:0:6666::61). The routed network has two potential ways of distributing routes so that this packet exits the site on the uplink at SERb2.

We could just rely on normal destination routes, without using source-prefix scoped routes. If we have SERb2 originate a normal unscoped destination route for D=2001:db8:0:6666::/64, the packets from H31 to H61 will exit the site at SERb2 as desired. We should not have to worry about SERa needing to originate the same route, because ISP-B should choose a globally unique prefix for the service at H61.

The alternative is to have SERb2 originate a source-prefix-scoped destination route of the form (S=2001:db8:0:b000::/52, D=2001:db8:0:6666::/64). From a forwarding point of view, the use of the source-prefix-scoped destination route would result in traffic with source addresses corresponding only to ISP-B being sent to SERb2. Instead, the use of the unscoped destination route would result in traffic with source addresses corresponding to ISP-A and ISP-B being sent to SERb2, as long as the destination address matches the destination prefix. It seems like either forwarding behavior would be acceptable.

However, from the point of view of the enterprise network administrator trying to configure, maintain, and trouble-shoot this multihoming solution, it seems much clearer to have SERb2 originate the source-prefix-scoped destination route correspond to the service offered by ISP-B. In this way, all of the traffic leaving the site is determined by the source-prefix-scoped routes, and all of the traffic within the site or arriving from external hosts is determined by the unscoped destination routes. Therefore, for this multihoming solution we choose to originate source-prefix-scoped routes for all traffic leaving the site.

2.5. ISPs and Provider-Assigned Prefixes

While we expect that most site multihoming involves connecting to only two ISPs, this solution allows for connections to an arbitrary number of ISPs to be supported. However, when evaluating scalable implementations of the solution, it would be reasonable to assume that the maximum number of ISPs that a site would connect to is five.

It is also useful to note that the prefixes assigned to the site by different ISPs will not overlap. This must be the case , since the provider-assigned addresses have to be globally unique.

2.6. Simplified Topologies

The topologies of many enterprise sites using this multihoming solution may in practice be simpler than the examples that we have used. The topology in Figure 1 could be further simplified by having all hosts directly connected to the LAN connecting the two site exit routers, SERa and SERb. The topology could also be simplified by having the uplinks to ISP-A and ISP-B both connected to the same site exit router. However, it is the aim of this draft to provide a solution that applies to a broad a range of enterprise site network topologies, so this draft focuses on providing a solution to the more general case. The simplified cases will also be supported by this solution, and there may even be optimizations that can be made for simplified cases. This solution however needs to support more complex topologies.

We are starting with the basic assumption that enterprise site networks can be quite complex from a routing perspective. However, even a complex site network can be multihomed to different ISPs with PA addresses using IPv4 and NAT. It is not reasonable to expect an enterprise network operator to change the routing topology of the site in order to deploy IPv6.

3. Generating Source-Prefix-Scoped Forwarding Tables

So far we have described in general terms how the routers in this solution that are capable of Source Address Dependent Routing will forward traffic using both normal unscoped destination routes and source-prefix-scoped destination routes. Here we give a precise method for generating a source-prefix-scoped forwarding table on a router that supports SADR.

1. Compute the next-hops for the source-prefix-scoped destination prefixes using only routers in the connected SADR domain. These are the initial source-prefix-scoped forwarding table entries.

2. Compute the next-hops for the unscoped destination prefixes using all routers in the IGP. This is the unscoped forwarding table.

3. Augment each less specific source-prefix-scoped forwarding table with all more specific source-prefix-scoped forwarding tables entries based on the following rule. If the destination prefix of the less specific source-prefix-scoped forwarding entry exactly matches the destination prefix of an existing more

specific source-prefix-scoped forwarding entry (including destination prefix length), then do not add the less specific source-prefix-scoped forwarding entry. If the destination prefix does NOT match an existing entry, then add the entry to the more source-prefix-scoped forwarding table. As the unscoped forwarding table is considered to be scoped to ::/0 this process starts with propagating routes from the unscoped forwarding table to source-prefix-scoped forwarding tables and then continues with propagating routes to more-specific-source-prefix-scoped forwarding tables should they exist.

The forward tables produced by this process are used in the following way to forward packets.

1. Select the most specific (longerst prefix match) source-prefix-scoped forwarding table that matches the source address of the packet (again, the unscoped forwarding table is considered to be scoped to ::/0).

2. Look up the destination address of the packet in the selected forwarding table to determine the next-hop for the packet.

The following example illustrates how this process is used to create a forwarding table for each provider-assigned source prefix. We consider the multihomed site network in Figure 3. Initially we assume that all of the routers in the site network support SADR. Figure 4 shows the routes that are originated by the routers in the site network.

```
Routes originated by SERa:
(S=2001:db8:0:a000::/52, D=2001:db8:0:5555/64)
(S=2001:db8:0:a000::/52, D=::/0)
(D=2001:db8:0:5555::/64)
(D=::/0)

Routes originated by SERb1:
(S=2001:db8:0:b000::/52, D=::/0)
(D=::/0)

Routes originated by SERb2:
(S=2001:db8:0:b000::/52, D=2001:db8:0:6666::/64)
(D=2001:db8:0:6666::/64)

Routes originated by R1:
(D=2001:db8:0:a010::/64)
(D=2001:db8:0:b010::/64)

Routes originated by R2:
(D=2001:db8:0:a010::/64)
(D=2001:db8:0:b010::/64)

Routes originated by R3:
(D=2001:db8:0:a020::/64)
(D=2001:db8:0:b020::/64)
```

Figure 4: Routes Originated by Routers in the Site Network

Each SER originates destination routes which are scoped to the source prefix assigned by the ISP that the SER connects to. Note that the SERs also originate the corresponding unscoped destination route. This is not needed when all of the routers in the site support SADR. However, it is required when some routers do not support SADR. This will be discussed in more detail later.

We focus on how R8 constructs its source-prefix-scoped forwarding tables from these route advertisements. R8 computes the next hops for destination routes which are scoped to the source prefix 2001:db8:0:a000::/52. The results are shown in the first table in Figure 5. (In this example, the next hops are computed assuming that all links have the same metric.) Then, R8 computes the next hops for destination routes which are scoped to the source prefix 2001:db8:0:b000::/52. The results are shown in the second table in Figure 5 . Finally, R8 computes the next hops for the unscoped destination prefixes. The results are shown in the third table in Figure 5.

```
forwarding entries scoped to
source prefix = 2001:db8:0:a000::/52
==========================================
D=2001:db8:0:5555/64      NH=R7
D=::/0                    NH=R7

forwarding entries scoped to
source prefix = 2001:db8:0:b000::/52
==========================================
D=2001:db8:0:6666/64      NH=SERb2
D=::/0                    NH=SERb1

unscoped forwarding entries
==========================================
D=2001:db8:0:a010::/64    NH=R2
D=2001:db8:0:b010::/64    NH=R2
D=2001:db8:0:a020::/64    NH=R5
D=2001:db8:0:b020::/64    NH=R5
D=2001:db8:0:5555::/64    NH=R7
D=2001:db8:0:6666::/64    NH=SERb2
D=::/0                    NH=SERb1
```

Figure 5: Forwarding Entries Computed at R8

The final step is for R8 to augment the less specific source-prefix-scoped forwarding entries with more specific source-prefix-scoped forwarding entries. As unscoped forwarding table is considered being scoped to ::/0 and both 2001:db8:0:a000::/52 and 2001:db8:0:b000::/52 are more specific prefixes of ::/0, the unscoped (scoped to ::/0) forwarding table needs to be augmented with both more specific source-prefix-scoped tables. If an less specific scoped forwarding entry has the exact same destination prefix as an more specific source-prefix-scoped forwarding entry (including destination prefix length), then the more specific source-prefix-scoped forwarding entry wins.

As as an example of how the source scoped forwarding entries are augmented, we consider how the two entries in the first table in Figure 5 (the table for source prefix = 2001:db8:0:a000::/52) are augmented with entries from the third table in Figure 5 (the table of unscoped or scoped for ::/0 forwarding entries). The first four unscoped forwarding entries (D=2001:db8:0:a010::/64, D=2001:db8:0:b010::/64, D=2001:db8:0:a020::/64, and D=2001:db8:0:b020::/64) are not an exact match for any of the existing entries in the forwarding table for source prefix 2001:db8:0:a000::/52. Therefore, these four entries are added to the final forwarding table for source prefix 2001:db8:0:a000::/52. The

result of adding these entries is reflected in first four entries the first table in Figure 6.

The next less specific scoped (scope is ::/0) forwarding table entry is for D=2001:db8:0:5555::/64. This entry is an exact match for the existing entry in the forwarding table for the more specific source prefix 2001:db8:0:a000::/52. Therefore, we do not replace the existing entry with the entry from the unscoped forwarding table. This is reflected in the fifth entry in the first table in Figure 6. (Note that since both scoped and unscoped entries have R7 as the next hop, the result of applying this rule is not visible.)

The next less specific prefix scoped (scope is ::/0) forwarding table entry is for D=2001:db8:0:6666::/64. This entry is not an exact match for any existing entries in the forwarding table for source prefix 2001:db8:0:a000::/52. Therefore, we add this entry. This is reflected in the sixth entry in the first table in Figure 6.

The next less specific prefix scoped (scope is ::/0) forwarding table entry is for D=::/0. This entry is an exact match for the existing entry in the forwarding table for more specific source prefix 2001:db8:0:a000::/52. Therefore, we do not overwrite the existing source-prefix-scoped entry, as can be seen in the last entry in the first table in Figure 6.

```
if source address matches 2001:db8:0:a000::/52
then use this forwarding table
==========================================
D=2001:db8:0:a010::/64    NH=R2
D=2001:db8:0:b010::/64    NH=R2
D=2001:db8:0:a020::/64    NH=R5
D=2001:db8:0:b020::/64    NH=R5
D=2001:db8:0:5555::/64    NH=R7
D=2001:db8:0:6666::/64    NH=SERb2
D=::/0                    NH=R7

else if source address matches 2001:db8:0:b000::/52
then use this forwarding table
==========================================
D=2001:db8:0:a010::/64    NH=R2
D=2001:db8:0:b010::/64    NH=R2
D=2001:db8:0:a020::/64    NH=R5
D=2001:db8:0:b020::/64    NH=R5
D=2001:db8:0:5555::/64    NH=R7
D=2001:db8:0:6666::/64    NH=SERb2
D=::/0                    NH=SERb1

else if source address matches ::/0 use this forwarding table
==========================================
D=2001:db8:0:a010::/64    NH=R2
D=2001:db8:0:b010::/64    NH=R2
D=2001:db8:0:a020::/64    NH=R5
D=2001:db8:0:b020::/64    NH=R5
D=2001:db8:0:5555::/64    NH=R7
D=2001:db8:0:6666::/64    NH=SERb2
D=::/0                    NH=SERb1
```

Figure 6: Complete Forwarding Tables Computed at R8

The forwarding tables produced by this process at R8 have the desired properties. A packet with a source address in 2001:db8:0:a000::/52 will be forwarded based on the first table in Figure 6. If the packet is destined for the Internet at large or the service at D=2001:db8:0:5555/64, it will be sent to R7 in the direction of SERa. If the packet is destined for an internal host, then the first four entries will send it to R2 or R5 as expected. Note that if this packet has a destination address corresponding to the service offered by ISP-B (D=2001:db8:0:5555::/64), then it will get forwarded to SERb2. It will be dropped by SERb2 or by ISP-B, since it the packet has a source address that was not assigned by ISP-B. However, this is expected behavior. In order to use the service offered by ISP-B, the host needs to originate the packet with a source address assigned by ISP-B.

In this example, a packet with a source address that doesn't match 2001:db8:0:a000::/52 or 2001:db8:0:b000::/52 must have originated from an external host. Such a packet will use the unscoped forwarding table (the last table in Figure 6). These packets will flow exactly as they would in absence of multihoming.

We can also modify this example to illustrate how it supports deployments where not all routers in the site support SADR. Continuing with the topology shown in Figure 3, suppose that R3 and R5 do not support SADR. Instead they are only capable of understanding unscoped route advertisements. The SADR routers in the network will still originate the routes shown in Figure 4. However, R3 and R5 will only understand the unscoped routes as shown in Figure 7.

```
Routes originated by SERa:
(D=2001:db8:0:5555::/64)
(D=::/0)

Routes originated by SERb1:
(D=::/0)

Routes originated by SERb2:
(D=2001:db8:0:6666::/64)

Routes originated by R1:
(D=2001:db8:0:a010::/64)
(D=2001:db8:0:b010::/64)

Routes originated by R2:
(D=2001:db8:0:a010::/64)
(D=2001:db8:0:b010::/64)

Routes originated by R3:
(D=2001:db8:0:a020::/64)
(D=2001:db8:0:b020::/64)
```

Figure 7: Routes Advertisements Understood by Routers that do no Support SADR

With these unscoped route advertisements, R5 will produce the forwarding table shown in Figure 8.

```
forwarding table
=========================================
D=2001:db8:0:a010::/64    NH=R8
D=2001:db8:0:b010::/64    NH=R8
D=2001:db8:0:a020::/64    NH=R3
D=2001:db8:0:b020::/64    NH=R3
D=2001:db8:0:5555::/64    NH=R8
D=2001:db8:0:6666::/64    NH=SERb2
D=::/0                    NH=R8
```

Figure 8: Forwarding Table For R5, Which Doesn't Understand Source-Prefix-Scoped Routes

Any traffic that needs to exit the site will eventually hit a SADR-capable router. Once that traffic enters the SADR-capable domain, then it will not leave that domain until it exits the site. This property is required in order to guarantee that there will not be routing loops involving SADR-capable and non-SADR-capable routers.

Note that the mechanism described here for converting source-prefix-scoped destination prefix routing advertisements into forwarding state is somewhat different from that proposed in [I-D.ietf-rtgwg-dst-src-routing]. The method described in this document is intended to be easy to understand for network enterprise operators while at the same time being functionally correct. Another difference is that the method in this document assumes that source prefix will not overlap. Other differences between the two approaches still need to be understood and reconciled.

An interesting side-effect of deploying SADR is if all routers in a given network support SADR and have a scoped forwarding table, then the unscoped forwarding table can be eliminated which ensures that packets with legitimate source addresses only can leave the network (as there are no scoped forwarding tables for spoofed/bogon source addresses). It would prevent accidental leaks of ULA/reserved/link-local sources to the Internet as well as ensures that no spoofing is possible from the SADR-enabled network.

4. Mechanisms For Hosts To Choose Good Source Addresses In A Multihomed Site

Until this point, we have made the assumption that hosts are able to choose the correct source address using some unspecified mechanism. This has allowed us to just focus on what the routers in a multihomed site network need to do in order to forward packets to the correct ISP based on source address. Now we look at possible mechanisms for hosts to choose the correct source address. We also look at what

role, if any, the routers may play in providing information that helps hosts to choose source addresses.

Any host that needs to be able to send traffic using the uplinks to a given ISP is expected to be configured with an address from the prefix assigned by that ISP. The host will control which ISP is used for its traffic by selecting one of the addresses configured on the host as the source address for outgoing traffic. It is the responsibility of the site network to ensure that a packet with the source address from an ISP is now sent on an uplink to that ISP.

If all of the ISP uplinks are working, the choice of source address by the host may be driven by the desire to load share across ISP uplinks, or it may be driven by the desire to take advantage of certain properties of a particular uplink or ISP. If any of the ISP uplinks is not working, then the choice of source address by the host can determine if packets get dropped.

How a host should make good decisions about source address selection in a multihomed site is not a solved problem. We do not attempt to solve this problem in this document. Instead we discuss the current state of affairs with respect to standardized solutions and implementation of those solutions. We also look at proposed solutions for this problem.

An external host initiating communication with a host internal to a PA multihomed site will need to know multiple addresses for that host in order to communicate with it using different ISPs to the multihomed site. These addresses are typically learned through DNS. (For simplicity, we assume that the external host is single-homed.) The external host chooses the ISP that will be used at the remote multihomed site by setting the destination address on the packets it transmits. For a sessions originated from an external host to an internal host, the choice of source address used by the internal host is simple. The internal host has no choice but to use the destination address in the received packet as the source address of the transmitted packet.

For a session originated by a host internal to the multi-homed site, the decision of what source address to select is more complicated. We consider three main methods for hosts to get information about the network. The two proactive methods are Neighbor Discovery Router Advertisements and DHCPv6. The one reactive method we consider is ICMPv6. Note that we are explicitly excluding the possibility of having hosts participate in or even listen directly to routing protocol advertisements.

First we look at how a host is currently expected to select the source and destination address with which it sends a packet.

4.1. Source Address Selection Algorithm on Hosts

[RFC6724] defines the algorithms that hosts are expected to use to select source and destination addresses for packets. It defines an algorithm for selecting a source address and a separate algorithm for selecting a destination address. Both of these algorithms depend on a policy table. [RFC6724] defines a default policy which produces certain behavior.

The rules in the two algorithms in [RFC6724] depend on many different properties of addresses. While these are needed for understanding how a host should choose addresses in an arbitrary environment, most of the rules are not relevant for understanding how a host should choose among multiple source addresses in mulitihomed envinronment when sending a packet to a remote host. Returning to the example in Figure 3, we look at what the default algorithms in [RFC6724] say about the source address that internal host H31 should use to send traffic to external host H101, somewhere on the Internet. Let’s look at what rules in [RFC6724] are actually used by H31 in this case.

There is no choice to be made with respect to destination address. H31 needs to send a packet with D=2001:db8:0:1234::101 in order to reach H101. So H31 have to choose between using S=2001:db8:0:a010::31 or S=2001:db8:0:b010::31 as the source address for this packet. We go through the rules for source address selection in Section 5 of [RFC6724]. Rule 1 (Prefer same address) is not useful to break the tie between source addresses, because neither the candidate source addresses equals the destination address. Rule 2 (Prefer appropriate scope) is also not used in this scenario, because both source addresses and the destination address have global scope.

Rule 3 (Avoid deprecated addresses) applies to an address that has been autoconfigured by a host using stateless address autoconfiguration as defined in [RFC4862]. An address autoconfigured by a host has a preferred lifetime and a valid lifetime. The address is preferred until the preferred lifetime expires, after which it becomes deprecated. A deprecated address is not used if there is a preferred address of the appropriate scope available. When the valid lifetime expires, the address cannot be used at all. The preferred and valid lifetimes for an autoconfigured address are set based on the corresponding lifetimes in the Prefix Information Option in Neighbor Discovery Router Advertisements. So a possible tool to control source address selection in this scenario would be for a host to make an address deprecated by having routers on that link, R1 and

R2 in Figure 3, send a Router Advertisement message contaning a Prefix Information Option for the source prefix to be discouraged (or prohibited) with the preferred lifetime set to zero. This is a rather blunt tool, because it discourages or prohibits the use of that source prefix for all destinations. However, it may be useful in some scenarios. For example, if all uplinks to a particular ISP fail, it is desirable to prevent hosts from using source addresses from that ISP address space.

Rule 4 (Avoid home addresses) does not apply here because we are not considering Mobile IP.

Rule 5 (Prefer outgoing interface) is not useful in this scenario, because both source addresses are assigned to the same interface.

Rule 5.5 (Prefer addresses in a prefix advertised by the next-hop) is not useful in the scenario when both R1 and R2 will advertise both source prefixes. However potentially this rule may allow a host to select the correct source prefix by selecting a next-hop. The most obvious way would be to make R1 to advertise itself as a default router and send PIO for 2001:db8:0:a010::/64, while R2 is advertising itself as a default router and sending PIO for 2001:db8:0:b010::/64. We'll discuss later how Rule 5.5 can be used to influence a source address selection in single-router topologies (e.g. when H41 is sending traffic using R3 as a default gateway).

Rule 6 (Prefer matching label) refers to the Label value determined for each source and destination prefix as a result of applying the policy table to the prefix. With the default policy table defined in Section 2.1 of [RFC6724], Label(2001:db8:0:a010::31) = 5, Label(2001:db8:0:b010::31) = 5, and Label(2001:db8:0:1234::101) = 5. So with the default policy, Rule 6 does not break the tie. However, the algorithms in [RFC6724] are defined in such as way that non-default address selection policy tables can be used. [RFC7078] defines a way to distribute a non-default address selection policy table to hosts using DHCPv6. So even though the application of rule 6 to this scenario using the default policy table is not useful, rule 6 may still be a useful tool.

Rule 7 (Prefer temporary addresses) has to do with the technique described in [RFC4941] to periodically randomize the interface portion of an IPv6 address that has been generated using stateless address autoconfiguration. In general, if H31 were using this technique, it would use it for both source addresses, for example creating temporary addresses 2001:db8:0:a010:2839:9938:ab58:830f and 2001:db8:0:b010:4838:f483:8384:3208, in addition to 2001:db8:0:a010::31 and 2001:db8:0:b010::31. So this rule would

prefer the two temporary addresses, but it would not break the tie between the two source prefixes from ISP-A and ISP-B.

Rule 8 (Use longest matching prefix) dictates that between two candidate source addresses the one which has longest common prefix length with the destination address. For example, if H31 were selecting the source address for sending packets to H101, this rule would not be a tie breaker as for both candidate source addresses 2001:db8:0:a101::31 and 2001:db8:0:b101::31 the common prefix length with the destination is 48. However if H31 were selecting the source address for sending packets H41 address 2001:db8:0:a020::41, then this rule would result in using 2001:db8:0:a101::31 as a source (2001:db8:0:a101::31 and 2001:db8:0:a020::41 share the common prefix 2001:db8:0:a000::/58, while for '2001:db8:0:b101::31 and 2001:db8:0:a020::41 the common prefix is 2001:db8:0:a000::/51). Therefore rule 8 might be useful for selecting the correct source address in some but not all scenarios (for example if ISP-B services belong to 2001:db8:0:b000::/59 then H31 would always use 2001:db8:0:b010::31 to access those destinations).

So we can see that of the 8 source selection address rules from [RFC6724], five actually apply to our basic site multihoming scenario. The rules that are relevant to this scenario are summarized below.

- Rule 3: Avoid deprecated addresses.
- Rule 5.5: Prefer addresses in a prefix advertised by the next-hop.
- Rule 6: Prefer matching label.
- Rule 8: Prefer longest matching prefix.

The two methods that we discuss for controlling the source address selection through the four relevant rules above are SLAAC Router Advertisement messages and DHCPv6.

We also consider a possible role for ICMPv6 for getting traffic-driven feedback from the network. With the source address selection algorithm discussed above, the goal is to choose the correct source address on the first try, before any traffic is sent. However, another strategy is to choose a source address, send the packet, get feedback from the network about whether or not the source address is correct, and try another source address if it is not.

We consider four scenarios where a host needs to select the correct source address. The first is when both uplinks are working. The second is when one uplink has failed. The third one is a situation

when one failed uplink has recovered. The last one is failure of both (all) uplinks.

4.2. Selecting Source Address When Both Uplinks Are Working

Again we return to the topology in Figure 3. Suppose that the site administrator wants to implement a policy by which all hosts need to use ISP-A to reach H01 at D=2001:db8:0:1234::101. So for example, H31 needs to select S=2001:db8:0:a010::31.

4.2.1. Distributing Address Selection Policy Table with DHCPv6

This policy can be implemented by using DHCPv6 to distribute an address selection policy table that assigns the same label to destination address that match 2001:db8:0:1234::/64 as it does to source addresses that match 2001:db8:0:a000::/52. The following two entries accomplish this.

Prefix	Precedence	Label
2001:db8:0:1234::/64	50	33
2001:db8:0:a000::/52	50	33

Figure 9: Policy table entries to implement a routing policy

This requires that the hosts implement [RFC6724], the basic source and destination address framework, along with [RFC7078], the DHCPv6 extension for distributing a non-default policy table. Note that it does NOT require that the hosts use DHCPv6 for address assignment. The hosts could still use stateless address autoconfiguration for address configuration, while using DHCPv6 only for policy table distribution (see [RFC3736]). However this method has a number of disadvantages:

- DHCPv6 support is not a mandatory requirement for IPv6 hosts, so this method might not work for all devices.

- Network administrators are required to explicitly configure the desired network access policies on DHCPv6 servers. While it might be feasible in the scenarion of a single multihomed network, such approach might have some scalability issues, especially if the centralized DHCPv6 solution is deployed to serve a large number of multiomed sites.

4.2.2. Controlling Source Address Selection With Router Advertisements

Neighbor Discovery currently has two mechanisms to communicate prefix information to hosts. The base specification for Neighbor Discovery (see [RFC4861]) defines the Prefix Information Option (PIO) in the

Router Advertisement (RA) message. When a host receives a PIO with the A-flag set, it can use the prefix in the PIO as source prefix from which it assigns itself an IP address using stateless address autoconfiguration (SLAAC) procedures described in [RFC4862]. In the example of Figure 3, if the site network is using SLAAC, we would expect both R1 and R2 to send RA messages with PIOs for both source prefixes 2001:db8:0:a010::/64 and 2001:db8:0:b010::/64 with the A-flag set. H31 would then use the SLAAC procedure to configure itself with the 2001:db8:0:a010::31 and 2001:db8:0:b010::31.

Whereas a host learns about source prefixes from PIO messages, hosts can learn about a destination prefix from a Router Advertisement containing Route Information Option (RIO), as specified in [RFC4191]. The destination prefixes in RIOs are intended to allow a host to choose the router that it uses as its first hop to reach a particular destination prefix.

As currently standardized, neither PIO nor RIO options contained in Neighbor Discovery Router Advertisements can communicate the information needed to implement the desired routing policy. PIO's communicate source prefixes, and RIO communicate destination prefixes. However, there is currently no standardized way to directly associate a particular destination prefix with a particular source prefix.

[I-D.pfister-6man-sadr-ra] proposes a Source Address Dependent Route Information option for Neighbor Discovery Router Advertisements which would associate a source prefix and with a destination prefix. The details of [I-D.pfister-6man-sadr-ra] might need tweaking to address this use case. However, in order to be able to use Neighbor Discovery Router Advertisements to implement this routing policy, an extension that allows a R1 and R2 to explicitly communicate to H31 an association between S=2001:db8:0:a000::/52 D=2001:db8:0:1234::/64 would be needed.

However, Rule 5.5 of the source address selection algorithm (discussed in Section 4.1 above), together with default router preference (specified in [RFC4191]) and RIO can be used to influence a source address selection on a host as described below. Let's look at source address selection on the host H41. It receives RAs from R3 with PIOs for 2001:db8:0:a020::/64 and 2001:db8:0:b020::/64. At that point all traffic would use the same next-hop (R3 link-local address) so Rule 5.5 does not apply. Now let's assume that R3 supports SADR and has two scoped forwarding tables, one scoped to S=2001:db8:0:a000::/52 and another scoped to S=2001:db8:0:b000::/52. If R3 generates two different link-local addresses for its interface facing H41 (one for each scoped forwarding table, LLA_A and LLA_B) and starts sending two different RAs: one is sent from LLA_A and

includes PIO for 2001:db8:0:a020::/64, another us sent from LLA_B and includes PIO for 2001:db8:0:b020::/64. Now it is possible to influence H41 source address selection for destinations which follow the default route by setting default router preference in RAs. If it is desired that H41 reaches H101 (or any destinations in the Internet) via ISP-A, then RAs sent from LLA_A should have default router preference set to 01 (high priority), while RAs sent from LLA_B should have preference set to 11 (low). Then LLA_A would be chosen as a next-hop for H101 and therefore (as per rule 5.5) 2001:db8:0:a020::41 would be selected as the source address. If, at the same time, it is desired that H61 is accessible via ISP-B then R3 should include a RIO for 2001:db8:0:6666::/64 to its RA sent from LLA_B. H41 would chose LLA_B as a next-hop for all traffic to H61 and then as per Rule 5.5, 2001:db8:0:b020::41 would be selected as a source address.

If in the above mentioned scenario it is desirable that all Internet traffic leaves the network via ISP-A and the link to ISP-B is used for accessing ISP-B services only (not as ISP-A link backup), then RAs sent by R3 from LLA_B should have Router Lifetime set to 0 and should include RIOs for ISP-B address space. It would instruct H41 to use LLA_A for all Internet traffic but use LLA_B as a next-hop while sending traffic to ISP-B addresses.

The description of the mechanism above assumes SADR support by the first-hop routers as well as SERs. However, a first-hop router can still provide a less flexible version of this mechanism even without implementing SADR. This could be done by providing configuration knobs on the first-hop router that allow it to generate different link-local addresses and to send individual RAs for each prefix.

The mechanism described above relies on Rule 5.5 of the default source address selection algorithm defined in [RFC6724]. [RFC8028] recommends that a host SHOULD select default routers for each prefix in which it is assigned an address. It also recommends that hosts SHOULD implement Rule 5.5. of [RFC6724]. Hosts following the recommendations specified in [RFC8028] therefore should be able to benefit from the solution described in this document. No standards need to be updated in regards to host behavior.

4.2.3. Controlling Source Address Selection With ICMPv6

We now discuss how one might use ICMPv6 to implement the routing policy to send traffic destined for H101 out the uplink to ISP-A, even when uplinks to both ISPs are working. If H31 started sending traffic to H101 with S=2001:db8:0:b010::31 and D=2001:db8:0:1234::101, it would be routed through SER-b1 and out the uplink to ISP-B. SERb1 could recognize that this is traffic is not

following the desired routing policy and react by sending an ICMPv6 message back to H31.

In this example, we could arrange things so that SERb1 drops the packet with S=2001:db8:0:b010::31 and D=2001:db8:0:1234::101, and then sends to H31 an ICMPv6 Destination Unreachable message with Code 5 (Source address failed ingress/egress policy). When H31 receives this packet, it would then be expected to try another source address to reach the destination. In this example, H31 would then send a packet with S=2001:db8:0:a010::31 and D=2001:db8:0:1234::101, which will reach SERa and be forwarded out the uplink to ISP-A.

However, we would also want it to be the case that SERb1 does not enforce this routing policy when the uplink from SERa to ISP-A has failed. This could be accomplished by having SERa originate a source-prefix-scoped route for (S=2001:db8:0:a000::/52, D=2001:db8:0:1234::/64) and have SERb1 monitor the presence of that route. If that route is not present (because SERa has stopped originating it), then SERb1 will not enforce the routing policy, and it will forward packets with S=2001:db8:0:b010::31 and D=2001:db8:0:1234::101 out its uplink to ISP-B.

We can also use this source-prefix-scoped route originated by SERa to communicate the desired routing policy to SERb1. We can define an EXCLUSIVE flag to be advertised together with the IGP route for (S=2001:db8:0:a000::/52, D=2001:db8:0:1234::/64). This would allow SERa to communicate to SERb that SERb should reject traffic for D=2001:db8:0:1234::/64 and respond with an ICMPv6 Destination Unreachable Code 5 message, as long as the route for (S=2001:db8:0:a000::/52, D=2001:db8:0:1234::/64) is present.

Finally, if we are willing to extend ICMPv6 to support this solution, then we could create a mechanism for SERb1 to tell the host what source address it should be using to successfully forward packets that meet the policy. In its current form, when SERb1 sends an ICMPv6 Destination Unreachable Code 5 message, it is basically saying, "This source address is wrong. Try another source address." In the absence of a clear indication which address to try next, the host will iterate over all addresses assigned to the interface (e.g. various privacy addresses) which would lead to significant delays and degraded user experience. It would be better is if the ICMPv6 message could say, "This source address is wrong. Instead use a source address in S=2001:db8:0:a000::/52.".

However using ICMPv6 for signalling source address information back to hosts introduces new challenges. Most routers currently have software or hardware limits on generating ICMP messages. An site administrator deploying a solution that relies on the SERs generating

ICMP messages could try to improve the performance of SERs for generating ICMP messages. However, in a large network, it is still likely that ICMP message generation limits will be reached. As a result hosts would not receive ICMPv6 back which in turn leads to traffic blackholing and poor user experience. To improve the scalability of ICMPv6-based signalling hosts SHOULD cache the preferred source address (or prefix) for the given destination (which in turn might cause issues in case of the corresponding ISP uplinks failure - see Section 4.3). In addition, the same source prefix SHOULD be used for other destinations in the same /64 as the original destination address. The source prefix SHOULD have a specific lifetime. Expiration of the lifetime SHOULD trigger the source address selection algorithm again.

Using ICMPv6 Code 5 message for influencing source address selection allows an attacker to exhaust the list of candidate source addresses on the host by sending spoofed ICMPv6 Code 5 for all prefixes known on the network (therefore preventing a victim from establishing a communication with the destination host). To protect from such attack hosts SHOULD verify that the original packet header included into ICMPv6 error message was actually sent by the host.

As currently standardized in [RFC4443], the ICMPv6 Destination Unreachable Message with Code 5 would allow for the iterative approach to retransmitting packets using different source addresses. As currently defined, the ICMPv6 message does not provide a mechanism to communication information about which source prefix should be used for a retransmitted packet. The current document does not define such a mechanism. However, we note that this might be a useful extension to define in a different document.

4.2.4. Summary of Methods For Controlling Source Address Selection To Implement Routing Policy

So to summarize this section, we have looked at three methods for implementing a simple routing policy where all traffic for a given destination on the Internet needs to use a particular ISP, even when the uplinks to both ISPs are working.

The default source address selection policy cannot distinguish between the source addresses needed to enforce this policy, so a non-default policy table using associating source and destination prefixes using Label values would need to be installed on each host. A mechanism exists for DHCPv6 to distribute a non-default policy table but such solution would heavily rely on DHCPv6 support by host operating system. Moreover there is no mechanism to translate desired routing/traffic engineering policies into policy tables on

DHCPv6 servers. Therefore using DHCPv6 for controlling address selection policy table is not recommended and SHOULD NOT be used.

At the same time Router Advertisements provide a reliable mechanism to influence source address selection process via PIO, RIO and default router preferences. As all those options have been standardized by IETF and are supported by various operating systems, no changes are required on hosts. First-hop routers in the enterprise network need to be able of sending different RAs for different SLAAC prefixes (either based on scoped forwarding tables or based on pre-configured policies).

SERs can enforce the routing policy by sending ICMPv6 Destination Unreachable messages with Code 5 (Source address failed ingress/ egress policy) for traffic that is being sent with the wrong source address. The policy distribution can be automated by defining an EXCLUSIVE flag for the source-prefix-scoped route which can be set on the SER that originates the route. As ICMPv6 message generation can be rate-limited on routers, it SHOULD NOT be used as the only mechanism to influence source address selection on hosts. While hosts SHOULD select the correct source address for a given destination the network SHOULD signal any source address issues back to hosts using ICMPv6 error messages.

4.3. Selecting Source Address When One Uplink Has Failed

Now we discuss if DHCPv6, Neighbor Discovery Router Advertisements, and ICMPv6 can help a host choose the right source address when an uplink to one of the ISPs has failed. Again we look at the scenario in Figure 3. This time we look at traffic from H31 destined for external host H501 at D=2001:db8:0:5678::501. We initially assume that the uplink from SERa to ISP-A is working and that the uplink from SERb1 to ISP-B is working.

We assume there is no particular routing policy desired, so H31 is free to send packets with S=2001:db8:0:a010::31 or S=2001:db8:0:b010::31 and have them delivered to H501. For this example, we assume that H31 has chosen S=2001:db8:0:b010::31 so that the packets exit via SERb to ISP-B. Now we see what happens when the link from SERb1 to ISP-B fails. How should H31 learn that it needs to start sending the packet to H501 with S=2001:db8:0:a010::31 in order to start using the uplink to ISP-A? We need to do this in a way that doesn’t prevent H31 from still sending packets with S=2001:db8:0:b010::31 in order to reach H61 at D=2001:db8:0:6666::61.

4.3.1. Controlling Source Address Selection With DHCPv6

For this example we assume that the site network in Figure 3 has a centralized DHCP server and all routers act as DHCP relay agents. We assume that both of the addresses assigned to H31 were assigned via DHCP.

We could try to have the DHCP server monitor the state of the uplink from SERb1 to ISP-B in some manner and then tell H31 that it can no longer use S=2001:db8:0:b010::31 by settings its valid lifetime to zero. The DHCP server could initiate this process by sending a Reconfigure Message to H31 as described in Section 19 of [RFC3315]. Or the DHCP server can assign addresses with short lifetimes in order to force clients to renew them often.

This approach would prevent H31 from using S=2001:db8:0:b010::31 to reach the a host on the Internet. However, it would also prevent H31 from using S=2001:db8:0:b010::31 to reach H61 at D=2001:db8:0:6666::61, which is not desirable.

Another potential approach is to have the DHCP server monitor the uplink from SERb1 to ISP-B and control the choice of source address on H31 by updating its address selection policy table via the mechanism in [RFC7078]. The DHCP server could initiate this process by sending a Reconfigure Message to H31. Note that [RFC3315] requires that Reconfigure Message use DHCP authentication. DHCP authentication could be avoided by using short address lifetimes to force clients to send Renew messages to the server often. If the host is not obtaining its IP addresses from the DHCP server, then it would need to use the Information Refresh Time option defined in [RFC4242].

If the following policy table can be installed on H31 after the failure of the uplink from SERb1, then the desired routing behavior should be achieved based on source and destination prefix being matched with label values.

Prefix	Precedence	Label
::/0	50	44
2001:db8:0:a000::/52	50	44
2001:db8:0:6666::/64	50	55
2001:db8:0:b000::/52	50	55

Figure 10: Policy Table Needed On Failure Of Uplink From SERb1

The described solution has a number of significant drawbacks, some of them already discussed in Section 4.2.1.

- DHCPv6 support is not required for an IPv6 host and there are operating systems which do not support DHCPv6. Besides that, it does not appear that [RFC7078] has been widely implemented on host operating systems.

- [RFC7078] does not clearly specify this kind of a dynamic use case where address selection policy needs to be updated quickly in response to the failure of a link. In a large network it would present scalability issues as many hosts need to be reconfigured in very short period of time.

- Updating DHCPv6 server configuration each time an ISP uplink changes its state introduces some scalability issues, especially for mid/large distributed scale enterprise networks. In addition to that, the policy table needs to be manually configured by administrators which makes that solution prone to human error.

- No mechanism exists for making DHCPv6 servers aware of network topology/routing changes in the network. In general DHCPv6 servers monitoring network-related events sounds like a bad idea as completely new functionality beyond the scope of DHCPv6 role is required.

4.3.2. Controlling Source Address Selection With Router Advertisements

The same mechanism as discussed in Section 4.2.2 can be used to control the source address selection in the case of an uplink failure. If a particular prefix should not be used as a source for any destinations, then the router needs to send RA with Preferred Lifetime field for that prefix set to 0.

Let’s consider a scenario when all uplinks are operational and H41 receives two different RAs from R3: one from LLA_A with PIO for 2001:db8:0:a020::/64, default router preference set to 11 (low) and another one from LLA_B with PIO for 2001:db8:0:a020::/64, default router preference set to 01 (high) and RIO for 2001:db8:0:6666::/64. As a result H41 is using 2001:db8:0:b020::41 as a source address for all Internet traffic and those packets are sent by SERs to ISP-B. If SERb1 uplink to ISP-B failed, the desired behavior is that H41 stops using 2001:db8:0:b020::41 as a source address for all destinations but H61. To achieve that R3 should react to SERb1 uplink failure (which could be detected as the scoped route (S=2001:db8:0:b000::/52, D=::/0) disappearance) by withdrawing itself as a default router. R3 sends a new RA from LLA_B with Router Lifetime value set to 0 (which means that it should not be used as default router). That RA still contains PIO for 2001:db8:0:b020::/64 (for SLAAC purposes) and RIO for 2001:db8:0:6666::/64 so H41 can reach H61 using LLA_B as a next-hop and 2001:db8:0:b020::41 as a source address. For all traffic

following the default route, LLA_A will be used as a next-hop and 2001:db8:0:a020::41 as a source address.

If all uplinks to ISP-B have failed and therefore source addresses from ISP-B address space should not be used at all, the forwarding table scoped S=2001:db8:0:b000::/52 contains no entries. Hosts can be instructed to stop using source addresses from that block by sending RAs containing PIO with Preferred Lifetime set to 0.

4.3.3. Controlling Source Address Selection With ICMPv6

Now we look at how ICMPv6 messages can provide information back to H31. We assume again that at the time of the failure H31 is sending packets to H501 using (S=2001:db8:0:b010::31, D=2001:db8:0:5678::501). When the uplink from SERb1 to ISP-B fails, SERb1 would stop originating its source-prefix-scoped route for the default destination (S=2001:db8:0:b000::/52, D=::/0) as well as its unscoped default destination route. With these routes no longer in the IGP, traffic with (S=2001:db8:0:b010::31, D=2001:db8:0:5678::501) would end up at SERa based on the unscoped default destination route being originated by SERa. Since that traffic has the wrong source address to be forwarded to ISP-A, SERa would drop it and send a Destination Unreachable message with Code 5 (Source address failed ingress/egress policy) back to H31. H31 would then know to use another source address for that destination and would try with (S=2001:db8:0:a010::31, D=2001:db8:0:5678::501). This would be forwarded to SERa based on the source-prefix-scoped default destination route still being originated by SERa, and SERa would forward it to ISP-A. As discussed above, if we are willing to extend ICMPv6, SERa can even tell H31 what source address it should use to reach that destination. The expected host behaviour has been discussed in Section 4.2.3. Potential issue with using ICMPv6 for signalling source address issues back to hosts is that uplink to an ISP-B failure immediately invalidates source addresses from 2001:db8:0:b000::/52 for all hosts which triggers a large number of ICMPv6 being sent back to hosts - the same scalability/rate limiting issues discussed in Section 4.2.3 would apply.

4.3.4. Summary Of Methods For Controlling Source Address Selection On The Failure Of An Uplink

It appears that DHCPv6 is not particularly well suited to quickly changing the source address used by a host in the event of the failure of an uplink, which eliminates DHCPv6 from the list of potential solutions. On the other hand Router Advertisements provides a reliable mechanism to dynamically provide hosts with a list of valid prefixes to use as source addresses as well as prevent particular prefixes to be used. While no additional new features are

required to be implemented on hosts, routers need to be able to send RAs based on the state of scoped forwarding tables entries and to react to network topology changes by sending RAs with particular parameters set.

The use of ICMPv6 Destination Unreachable messages generated by the SER (or any SADR-capable) routers seem like they have the potential to provide a support mechanism together with RAs to signal source address selection errors back to hosts, however scalability issues may arise in large networks in case of sudden topology change. Therefore it is highly desirable that hosts are able to select the correct source address in case of uplinks failure with ICMPv6 being an additional mechanism to signal unexpected failures back to hosts.

The current behavior of different host operating system when receiving ICMPv6 Destination Unreachable message with code 5 (Source address failed ingress/egress policy) is not clear to the authors. Information from implementers, users, and testing would be quite helpful in evaluating this approach.

4.4. Selecting Source Address Upon Failed Uplink Recovery

The next logical step is to look at the scenario when a failed uplink on SERb1 to ISP-B is coming back up, so hosts can start using source addresses belonging to 2001:db8:0:b000::/52 again.

4.4.1. Controlling Source Address Selection With DHCPv6

The mechanism to use DHCPv6 to instruct the hosts (H31 in our example) to start using prefixes from ISP-B space (e.g. S=2001:db8:0:b010::31 for H31) to reach hosts on the Internet is quite similar to one discussed in Section 4.3.1 and shares the same drawbacks.

4.4.2. Controlling Source Address Selection With Router Advertisements

Let’s look at the scenario discussed in Section 4.3.2. If the uplink(s) failure caused the complete withdrawal of prefixes from 2001:db8:0:b000::/52 address space by setting Preferred Lifetime value to 0, then the recovery of the link should just trigger new RA being sent with non-zero Preferred Lifetime. In another scenario discussed in Section 4.3.2, the SERb1 uplink to ISP-B failure leads to disappearance of the (S=2001:db8:0:b000::/52, D=::/0) entry from the forwarding table scoped to S=2001:db8:0:b000::/52 and, in turn, caused R3 to send RAs from LLA_B with Router Lifetime set to 0. The recovery of the SERb1 uplink to ISP-B leads to (S=2001:db8:0:b000::/52, D=::/0) scoped forwarding entry re-appearance and instructs R3 that it should advertise itself as a

default router for ISP-B address space domain (send RAs from LLA_B with non-zero Router Lifetime).

4.4.3. Controlling Source Address Selection With ICMP

It looks like ICMPv6 provides a rather limited functionality to signal back to hosts that particular source addresses have become valid again. Unless the changes in the uplink state a particular (S,D) pair, hosts can keep using the same source address even after an ISP uplink has come back up. For example, after the uplink from SERb1 to ISP-B had failed, H31 received ICMPv6 Code 5 message (as described in Section 4.3.3) and allegedly started using (S=2001:db8:0:a010::31, D=2001:db8:0:5678::501) to reach H501. Now when the SERb1 uplink comes back up, the packets with that (S,D) pair are still routed to SERa1 and sent to the Internet. Therefore H31 is not informed that it should stop using 2001:db8:0:a010::31 and start using 2001:db8:0:b010::31 again. Unless SERa has a policy configured to drop packets (S=2001:db8:0:a010::31, D=2001:db8:0:5678::501) and send ICMPv6 back if SERb1 uplink to ISP-B is up, H31 will be unaware of the network topology change and keep using S=2001:db8:0:a010::31 for Internet destinations, including H51.

One of the possible option may be using a scoped route with EXCLUSIVE flag as described in Section 4.2.3. SERa1 uplink recovery would cause (S=2001:db8:0:a000::/52, D=2001:db8:0:1234::/64) route to reappear in the routing table. In the absence of that route packets to H101 which were sent to ISP-B (as ISP-A uplink was down) with source addresses from 2001:db8:0:b000::/52. When the route re-appears SERb1 would reject those packets and sends ICMPv6 back as discussed in Section 4.2.3. Practically it might lead to scalability issues which have been already discussed in Section 4.2.3 and Section 4.4.3.

4.4.4. Summary Of Methods For Controlling Source Address Selection Upon Failed Uplink Recovery

Once again DHCPv6 does not look like reasonable choice to manipulate source address selection process on a host in the case of network topology changes. Using Router Advertisement provides the flexible mechanism to dynamically react to network topology changes (if routers are able to use routing changes as a trigger for sending out RAs with specific parameters). ICMPv6 could be considered as a supporting mechanism to signal incorrect source address back to hosts but should not be considered as the only mechanism to control the address selection in multihomed environments.

4.5. Selecting Source Address When All Uplinks Failed

One particular tricky case is a scenario when all uplinks have failed. In that case there is no valid source address to be used for any external destinations while it might be desirable to have intra-site connectivity.

4.5.1. Controlling Source Address Selection With DHCPv6

From DHCPv6 perspective uplinks failure should be treated as two independent failures and processed as described in Section 4.3.1. At this stage it is quite obvious that it would result in quite complicated policy table which needs to be explicitly configured by administrators and therefore seems to be impractical.

4.5.2. Controlling Source Address Selection With Router Advertisements

As discussed in Section 4.3.2 an uplink failure causes the scoped default entry to disappear from the scoped forwarding table and triggers RAs with zero Router Lifetime. Complete disappearance of all scoped entries for a given source prefix would cause the prefix being withdrawn from hosts by setting Preferred Lifetime value to zero in PIO. If all uplinks (SERa, SERb1 and SERb2) failed, hosts either lost their default routers and/or have no global IPv6 addresses to use as a source. (Note that 'uplink failure' might mean 'IPv6 connectivity failure with IPv4 still being reachable', in which case hosts might fall back to IPv4 if there is IPv4 connectivity to destinations). As a results intra-site connectivity is broken. One of the possible way to solve it is to use ULAs.

All hosts have ULA addresses assigned in addition to GUAs and used for intra-site communication even if there is no GUA assigned to a host. To avoid accidental leaking of packets with ULA sources SADR-capable routers SHOULD have a scoped forwarding table for ULA source for internal routes but MUST NOT have an entry for D=::/0 in that table. In the absence of (S=ULA_Prefix; D=::/0) first-hop routers will send dedicated RAs from a unique link-local source LLA_ULA with PIO from ULA address space, RIO for the ULA prefix and Router Lifetime set to zero. The behaviour is consistent with the situation when SERb1 lost the uplink to ISP-B (so there is no Internet connectivity from 2001:db8:0:b000::/52 sources) but those sources can be used to reach some specific destinations. In the case of ULA there is no Internet connectivity from ULA sources but they can be used to reach another ULA destinations. Note that ULA usage could be particularly useful if all ISPs assign prefixes via DHCP-PD. In the absence of ULAs uplinks failure hosts would lost all their GUAs upon prefix lifetime expiration which again makes intra-site communication impossible.

It should be noted that the Rule 5.5 (prefer a prefix advertised by the selected next-hop) takes precedence over the Rule 6 (prefer matching label, which ensures that GUA source addresses are preferred over ULAs for GUA destinations). Therefore if ULAs are used, the network adminstrator needs to ensure that while the site has an Internet connectivity, hosts do not select a roter which advertises ULA prefixes as their default router.

4.5.3. Controlling Source Address Selection With ICMPv6

In case of all uplinks failure all SERs will drop outgoing IPv6 traffic and respond with ICMPv6 error message. In the large network when many hosts are trying to reach Internet destinations it means that SERs need to generate an ICMPv6 error to every packet they receive from hosts which presents the same scalability issues discussed in Section 4.3.3

4.5.4. Summary Of Methods For Controlling Source Address Selection When All Uplinks Failed

Again, combining SADR with Router Advertisements seems to be the most flexible and scalable way to control the source address selection on hosts.

4.6. Summary Of Methods For Controlling Source Address Selection

To summarize the scenarios and options discussed above:

While DHCPv6 allows administrators to manipulate source address selection policy tables, this method has a number of significant disadvantages which eliminates DHCPv6 from a list of potential solutions:

1. It required hosts to support DHCPv6 and its extension (RFC7078);

2. DHCPv6 server needs to monitor network state and detect routing changes.

3. The use of policy tables requires manual configuration and might be extremely complicated, especially in the case of distributed network when large number of remote sites are being served by centralized DHCPv6 servers.

4. Network topology/routing policy changes could trigger simultaneous re-configuration of large number of hosts which present serious scalability issues.

The use of Router Advertisements to influence the source address selection on hosts seem to be the most reliable, flexible and scalable solution. It has the following benefits:

1. no new (non-standard) functionality needs to be implemented on hosts (except for [RFC4191] support);

2. no changes in RA format;

3. routers can react to routing table changes by sending RAs which would minimize the failover time in the case of network topology changes;

4. information required for source address selection is broadcast to all affected hosts in case of topology change event which improves the scalability of the solution (comparing to DHCPv6 reconfiguration or ICMPv6 error messages).

To fully benefit from the RA-based solution, first-hop routers need to implement SADR and be able to send dedicated RAs per scoped forwarding table as discussed above, reacting to network changes with sending new RAs. It should be noted that the proposed solution would work even if first-hop routers are not SADR-capable but still able to send individual RAs for each ISP prefix and react to topology changes as discussed above (e.g. via configuration knobs).

The RA-based solution relies heavily on hosts correctly implementing default address selection algorith as defined in [RFC6724]. While the basic (and most common) multihoming scenario (two or more Internet uplinks, no ’wall gardens’) would work for any host supporting the minimal implementation of [RFC6724], more complex use cases (such as "wall garden" and other scenarios when some ISP resources can only be reached from that ISP address space) require that hosts support Rule 5.5 of the default address selection algorithm. There is some evidence that not all host OSes have that rule implemented currently. However it should be noted that [RFC8028] states that Rule 5.5 SHOULD be implemented.

ICMPv6 Code 5 error message SHOULD be used to complement RA-based solution to signal incorrect source address selection back to hosts, but it SHOULD NOT be considered as the stand-alone solution. To prevent scenarios when hosts in multihomed envinronments incorrectly identify onlink/offlink destinations, hosts should treat ICMPv6 Redirects as discussed in [RFC8028].

4.7. Other Configuration Parameters

4.7.1. DNS Configuration

In mutihomed envinronment each ISP might provide their own list of DNS servers. E.g. in the topology show on Figure 3, ISP-A might provide recursive DNS server H51 2001:db8:0:5555::51, while ISP-B might provide H61 2001:db8:0:6666::61 as a recursive DNS server. [RFC6106] defines IPv6 Router Advertisement options to allow IPv6 routers to advertise a list of DNS recursive server addresses and a DNS Search List to IPv6 hosts. Using RDNSS together with 'scoped' RAs as described above would allow a first-hop router (R3 in the Figure 3) to send DNS server addresses and search lists provided by each ISP (or the corporate DNS servers addresses if the enterprise is running its own DNS servers).

As discussed in Section 4.5.2, failure of all ISP uplinks would cause deprecaction of all addresses assigned to a host from the address space if all ISPs. If any intra-site IPv6 connectivity is still desirable (most likely to be the case for any mid/large scare network), then ULAs should be used as discussed in Section 4.5.2. In such a scenario, the enterprise network should run its own recursive DNS server(s) and provide its ULA addresses to hosts via RDNSS in RAs send for ULA-scoped forwarding table as described in Section 4.5.2.

There are some scenarios when the final outcome of the name resolution might be different depending on:

- which DNS server is used;
- which source address the client uses to send a DNS query to the server (DNS split horizon).

There is no way currently to instruct a host to use a particular DNS server out of the configured servers list for resolving a particular name. Therefore it does not seem feasible to solve the problem of DNS server selection on the host (it should be noted that this particular issue is protocol-agnostic and happens for IPv4 as well). In such a scenario it is recommended that the enterprise run its own local recursive DNS server.

To influence host source address selection for packets sent to a particular DNS server the following requirements must be met:

- the host supports RIO as defined in [RFC4191];
- the routers send RIO for routes to DNS server addresses.

For example, if it is desirable that host H31 reaches the ISP-A DNS server H51 2001:db8:0:5555::51 using its source address 2001:db8:0:a010::31, then both R1 and R2 should send the RIO containing the route to 2001:db8:0:5555::51 (or covering route) in their 'scoped' RAs, containing LLA_A as the default router address and the PO for SLAAC prefix 2001:db8:0:a010::/64. In that case the host H31 (if it supports the Rule 5.5) would select LLA_A as a next-hop and then chose 2001:db8:0:a010::31 as the source address for packets to the DNS server.

It should be noted that [RFC6106] explicitly prohibits using DNS information if the RA router Lifetime expired: "An RDNSS address or a DNSSL domain name MUST be used only as long as both the RA router Lifetime (advertised by a Router Advertisement message) and the corresponding option Lifetime have not expired.". Therefore hosts might ignore RDNSS information provided in ULA-scoped RAs as those RAs would have router lifetime set to 0. However the updated version of RFC6106 ([I-D.ietf-6man-rdnss-rfc6106bis]) has that requirement removed.

5. Other Solutions

5.1. Shim6

The Shim6 working group specified the Shim6 protocol [RFC5533] which allows a host at a multihomed site to communicate with an external host and exchange information about possible source and destination address pairs that they can use to communicate. It also specified the REAP protocol [RFC5534] to detect failures in the path between working address pairs and find new working address pairs. A fundamental requirement for Shim6 is that both internal and external hosts need to support Shim6. That is, both the host internal to the multihomed site and the host external to the multihomed site need to support Shim6 in order for there to be any benefit for the internal host to run Shim6. The Shim6 protocol specification was published in 2009, but it has not been implemented on widely used operating systems.

We do not consider Shim6 to be a viable solution. It suffers from the fact that it requires widespread deployment of Shim6 on hosts all over the Internet before the host at a PA multihomed site sees significant benefit. However, there appears to be no motivation for the vast majority of hosts on the Internet (which are not at PA multihomed sites) to deploy Shim6. This may help explain why Shim6 has not been widely implemented.

5.2. IPv6-to-IPv6 Network Prefix Translation

IPv6-to-IPv6 Network Prefix Translation (NPTv6) [RFC6296] is not the focus of this document. This document describes a solution where a host in a multihomed site determines which ISP a packet will be sent to based on the source address it applies to the packet. This solution has many moving parts. It requires some routers in the enterprise site to support some form of Source Address Dependent Routing (SADR). It requires a host to be able to learn when the uplink to an ISP fails so that it can stop using the source address corresponding to that ISP. Ongoing work to create mechanisms to accomplish this are discussed in this document, but they are still a work in progress.

This document attempts to create a PA multihoming solution that is as easy as possible for an enterprise to deploy. However, the success of this solution will depend greatly on whether or not the mechanisms for hosts to select source addresses based on the state of ISP uplinks gets implemented across a wide range of operating systems as the default mode of operation. Until that occurs, NPTv6 should still be considered a viable option to enable PA multihoming for enterprises.

6. IANA Considerations

This memo asks the IANA for no new parameters.

7. Security Considerations

7.1. Privacy Considerations

8. Acknowledgements

The original outline was suggested by Ole Troan.

9. References

9.1. Normative References

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <https://www.rfc-editor.org/info/rfc1122>.

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <https://www.rfc-editor.org/info/rfc1123>.

[RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, DOI 10.17487/RFC1918, February 1996, <https://www.rfc-editor.org/info/rfc1918>.

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

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <https://www.rfc-editor.org/info/rfc2460>.

[RFC2827] Ferguson, P. and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", BCP 38, RFC 2827, DOI 10.17487/RFC2827, May 2000, <https://www.rfc-editor.org/info/rfc2827>.

[RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, DOI 10.17487/RFC3315, July 2003, <https://www.rfc-editor.org/info/rfc3315>.

[RFC3582] Abley, J., Black, B., and V. Gill, "Goals for IPv6 Site-Multihoming Architectures", RFC 3582, DOI 10.17487/RFC3582, August 2003, <https://www.rfc-editor.org/info/rfc3582>.

[RFC4116] Abley, J., Lindqvist, K., Davies, E., Black, B., and V. Gill, "IPv4 Multihoming Practices and Limitations", RFC 4116, DOI 10.17487/RFC4116, July 2005, <https://www.rfc-editor.org/info/rfc4116>.

[RFC4191] Draves, R. and D. Thaler, "Default Router Preferences and More-Specific Routes", RFC 4191, DOI 10.17487/RFC4191, November 2005, <https://www.rfc-editor.org/info/rfc4191>.

[RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, DOI 10.17487/RFC4193, October 2005, <https://www.rfc-editor.org/info/rfc4193>.

[RFC4218] Nordmark, E. and T. Li, "Threats Relating to IPv6 Multihoming Solutions", RFC 4218, DOI 10.17487/RFC4218, October 2005, <https://www.rfc-editor.org/info/rfc4218>.

[RFC4219] Lear, E., "Things Multihoming in IPv6 (MULTI6) Developers Should Think About", RFC 4219, DOI 10.17487/RFC4219, October 2005, <https://www.rfc-editor.org/info/rfc4219>.

[RFC4242] Venaas, S., Chown, T., and B. Volz, "Information Refresh Time Option for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 4242, DOI 10.17487/RFC4242, November 2005, <https://www.rfc-editor.org/info/rfc4242>.

[RFC6106] Jeong, J., Park, S., Beloeil, L., and S. Madanapalli, "IPv6 Router Advertisement Options for DNS Configuration", RFC 6106, DOI 10.17487/RFC6106, November 2010, <https://www.rfc-editor.org/info/rfc6106>.

[RFC6296] Wasserman, M. and F. Baker, "IPv6-to-IPv6 Network Prefix Translation", RFC 6296, DOI 10.17487/RFC6296, June 2011, <https://www.rfc-editor.org/info/rfc6296>.

[RFC7157] Troan, O., Ed., Miles, D., Matsushima, S., Okimoto, T., and D. Wing, "IPv6 Multihoming without Network Address Translation", RFC 7157, DOI 10.17487/RFC7157, March 2014, <https://www.rfc-editor.org/info/rfc7157>.

9.2. Informative References

[I-D.baker-ipv6-isis-dst-src-routing]
Baker, F. and D. Lamparter, "IPv6 Source/Destination Routing using IS-IS", draft-baker-ipv6-isis-dst-src-routing-07 (work in progress), July 2017.

[I-D.baker-rtgwg-src-dst-routing-use-cases]
Baker, F., Xu, M., Yang, S., and J. Wu, "Requirements and Use Cases for Source/Destination Routing", draft-baker-rtgwg-src-dst-routing-use-cases-02 (work in progress), April 2016.

[I-D.boutier-babel-source-specific]
Boutier, M. and J. Chroboczek, "Source-Specific Routing in Babel", draft-boutier-babel-source-specific-03 (work in progress), July 2017.

[I-D.huitema-shim6-ingress-filtering]
Huitema, C., "Ingress filtering compatibility for IPv6 multihomed sites", draft-huitema-shim6-ingress-filtering-00 (work in progress), September 2005.

[I-D.ietf-6man-rdnss-rfc6106bis]
Jeong, J., Park, S., Beloeil, L., and S. Madanapalli, "IPv6 Router Advertisement Options for DNS Configuration", draft-ietf-6man-rdnss-rfc6106bis-16 (work in progress), February 2017.

[I-D.ietf-mif-mpvd-arch]
Anipko, D., "Multiple Provisioning Domain Architecture", draft-ietf-mif-mpvd-arch-11 (work in progress), March 2015.

[I-D.ietf-mptcp-experience]
Bonaventure, O., Paasch, C., and G. Detal, "Use Cases and Operational Experience with Multipath TCP", draft-ietf-mptcp-experience-07 (work in progress), October 2016.

[I-D.ietf-rtgwg-dst-src-routing]
Lamparter, D. and A. Smirnov, "Destination/Source Routing", draft-ietf-rtgwg-dst-src-routing-05 (work in progress), July 2017.

[I-D.pfister-6man-sadr-ra]
Pfister, P., "Source Address Dependent Route Information Option for Router Advertisements", draft-pfister-6man-sadr-ra-01 (work in progress), June 2015.

[I-D.xu-src-dst-bgp]
Xu, M., Yang, S., and J. Wu, "Source/Destination Routing Using BGP-4", draft-xu-src-dst-bgp-00 (work in progress), March 2016.

[PATRICIA]
Morrison, D., "Practical Algorithm to Retrieve Information Coded in Alphanumeric", Journal of the ACM 15(4) pp514-534, October 1968.

[RFC3704] Baker, F. and P. Savola, "Ingress Filtering for Multihomed Networks", BCP 84, RFC 3704, DOI 10.17487/RFC3704, March 2004, <https://www.rfc-editor.org/info/rfc3704>.

[RFC3736] Droms, R., "Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6", RFC 3736, DOI 10.17487/RFC3736, April 2004, <https://www.rfc-editor.org/info/rfc3736>.

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <https://www.rfc-editor.org/info/rfc4443>.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, DOI 10.17487/RFC4861, September 2007, <https://www.rfc-editor.org/info/rfc4861>.

[RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, DOI 10.17487/RFC4862, September 2007, <https://www.rfc-editor.org/info/rfc4862>.

[RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, DOI 10.17487/RFC4941, September 2007, <https://www.rfc-editor.org/info/rfc4941>.

[RFC5533] Nordmark, E. and M. Bagnulo, "Shim6: Level 3 Multihoming Shim Protocol for IPv6", RFC 5533, DOI 10.17487/RFC5533, June 2009, <https://www.rfc-editor.org/info/rfc5533>.

[RFC5534] Arkko, J. and I. van Beijnum, "Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming", RFC 5534, DOI 10.17487/RFC5534, June 2009, <https://www.rfc-editor.org/info/rfc5534>.

[RFC6555] Wing, D. and A. Yourtchenko, "Happy Eyeballs: Success with Dual-Stack Hosts", RFC 6555, DOI 10.17487/RFC6555, April 2012, <https://www.rfc-editor.org/info/rfc6555>.

[RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, DOI 10.17487/RFC6724, September 2012, <https://www.rfc-editor.org/info/rfc6724>.

[RFC7078] Matsumoto, A., Fujisaki, T., and T. Chown, "Distributing Address Selection Policy Using DHCPv6", RFC 7078, DOI 10.17487/RFC7078, January 2014, <https://www.rfc-editor.org/info/rfc7078>.

[RFC7788] Stenberg, M., Barth, S., and P. Pfister, "Home Networking Control Protocol", RFC 7788, DOI 10.17487/RFC7788, April 2016, <https://www.rfc-editor.org/info/rfc7788>.

[RFC8028] Baker, F. and B. Carpenter, "First-Hop Router Selection by Hosts in a Multi-Prefix Network", RFC 8028, DOI 10.17487/RFC8028, November 2016, <https://www.rfc-editor.org/info/rfc8028>.

Appendix A. Change Log

Initial Version: July 2016

Authors' Addresses

Fred Baker
Santa Barbara, California 93117
USA

Email: FredBaker.IETF@gmail.com

Chris Bowers
Juniper Networks
Sunnyvale, California 94089
USA

Email: cbowers@juniper.net

Jen Linkova
Google
Mountain View, California 94043
USA

Email: furry@google.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 7, 2018

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

X. Liu
Jabil

December 4, 2017

YANG Logical Network Elements
draft-ietf-rtgwg-lne-model-05

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 https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on June 7, 2018.

Copyright Notice

Table of Contents

1. Introduction 2
1.1. Terminology 3
2. Overview 3
3. Logical Network Elements 5
3.1. LNE Instantiation and Resource Assignment 6
3.2. LNE Management - LNE View 7
3.3. LNE Management - Host Network Device View 7
4. Security Considerations 8
5. IANA Considerations 9
6. Logical Network Element Model 9
7. References 13
7.1. Normative References 13
7.2. Informative References 14
Appendix A. Acknowledgments 14
Appendix B. Examples 15
B.1. Example: Host Device Managed LNE 15
B.1.1. Configuration Data 19
B.1.2. State Data 23
B.2. Example: Self Managed LNE 33
B.2.1. Configuration Data 36
B.2.2. State Data 39
Authors' Addresses 48

1. Introduction

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

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.

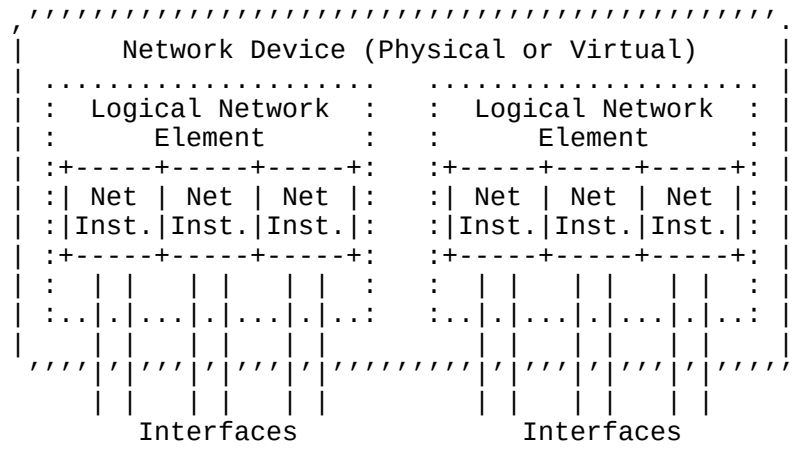


Figure 1: Module Element Relationships

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@2017-12-04.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 2017-12-04 {
     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;
         default "true";
         description
           "True if the host can access LNE information
            using the root mount point.  This value
            my not be modifiable in all implementations.";
       }
       leaf description {
         type string;
         description
           "Description of the logical network element.";
       }
       container "root" {
         description
           "Container for mount point.";
         yangmnt:mount-point "root" {
           description
             "Root for models supported per logical
              network element.  This mount point may or may not
              be inline based on the server implementation.  It
              SHALL always contain a YANG library and interfaces
              instance.
```

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

    // augment statements

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

         Note that a standard error will be returned if the
         identified leafref isn't present.  If an interfaces cannot
         be assigned for any other reason, the operation SHALL fail
         with an error-tag of 'operation-failed' and an error-app-tag
         of 'lne-assignment-failed'.  A meaningful error-info that
         indicates the source of the assignment failure SHOULD also
         be provided.";
      leaf bind-lne-name {
        type leafref {
          path "/logical-network-elements/logical-network-element/name";
        }
        description
          "Logical network element ID to which interface is bound.";
      }
    }

    // notification statements

    notification bind-lne-name-failed {
      description
        "Indicates an error in the association of an interface to an
         LNE. Only generated after success is initially returned when
         bind-lne-name is set.";
      leaf name {
        type leafref {
          path "/if:interfaces/if:interface/if:name";
        }
        mandatory true;
        description
          "Contains the interface name associated with the
```

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

7. References

7.1. Normative References

[I-D.ietf-netmod-schema-mount]
Bjorklund, M. and L. Lhotka, "YANG Schema Mount", draft-ietf-netmod-schema-mount-08 (work in progress), October 2017.

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

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[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>.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <https://www.rfc-editor.org/info/rfc7223>.

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-02 (work in progress), October 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., Bogdanovic, D., and X. Liu, "YANG Network Instances", draft-ietf-rtgwg-ni-model-04 (work in progress), September 2017.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <https://www.rfc-editor.org/info/rfc7317>.

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <https://www.rfc-editor.org/info/rfc7895>.

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

[RFC8022] Lhotka, L. and A. Lindem, "A YANG Data Model for Routing Management", RFC 8022, DOI 10.17487/RFC8022, November 2016, <https://www.rfc-editor.org/info/rfc8022>.

Appendix A. Acknowledgments

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

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

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

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

```
            "arbitrary-names",
            "pre-provisioning"
          ],
          "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-interfaces",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-ip",
          "revision": "2014-06-16",
          "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ip",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-logical-network-element",
          "revision": "2017-03-13",
          "feature": [
            "bind-lne-name"
          ],
          "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-logical-network-element",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-ospf",
          "revision": "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-yang-schema-mount:schema-mounts": {
      "mount-point": [
        {
          "module": "ietf-logical-network-element",
          "label": "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"
            }
          ]
        }
      ]
    }
  }
```

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

```
    "ietf-yang-library:modules-state": {
      "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
      "module": [
        {
          "name": "iana-if-type",
          "revision": "2014-05-08",
          "namespace":
          "urn:ietf:params:xml:ns:yang:iana-if-type",
          "conformance-type": "import"
        },
        {
          "name": "ietf-inet-types",
          "revision": "2013-07-15",
          "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-inet-types",
          "conformance-type": "import"
        },
        {
          "name": "ietf-interfaces",
          "revision": "2014-05-08",
          "feature": [
            "arbitrary-names",
            "pre-provisioning"
          ],
          "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-interfaces",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-ip",
          "revision": "2014-06-16",
          "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ip",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-logical-network-element",
          "revision": "2017-03-13",
          "feature": [
            "bind-lne-name"
          ],
          "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-logical-network-element",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-system",
          "revision": "2014-08-06",
```

```
          "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-system",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-yang-library",
          "revision": "2016-06-21",
          "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-yang-library",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-yang-schema-mount",
          "revision": "2017-05-16",
          "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-yang-types",
          "revision": "2013-07-15",
          "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-yang-types",
          "conformance-type": "import"
        }
      ]
    },

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

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: July 20, 2018

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

X. Liu
Jabil

January 16, 2018

YANG Network Instances
draft-ietf-rtgwg-ni-model-06

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 July 20, 2018.

Table of Contents

1. Introduction 2
1.1. Terminology 3
1.2. Status of Work and Open Issues 3
2. Overview 4
3. Network Instances 5
3.1. NI Types and Mount Points 6
3.1.1. Well Known Mount Points 7
3.1.2. NI Type Example 8
3.2. NIs and Interfaces 9
3.3. Network Instance Management 11
3.4. Network Instance Instantiation 13
4. Security Considerations 14
5. IANA Considerations 14
6. Network Instance Model 15
7. References 21
7.1. Normative References 21
7.2. Informative References 22
Appendix A. Acknowledgments 23
Appendix B. Example NI usage 23
B.1. Configuration Data 23
B.2. State Data 26
Authors' Addresses 32

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.

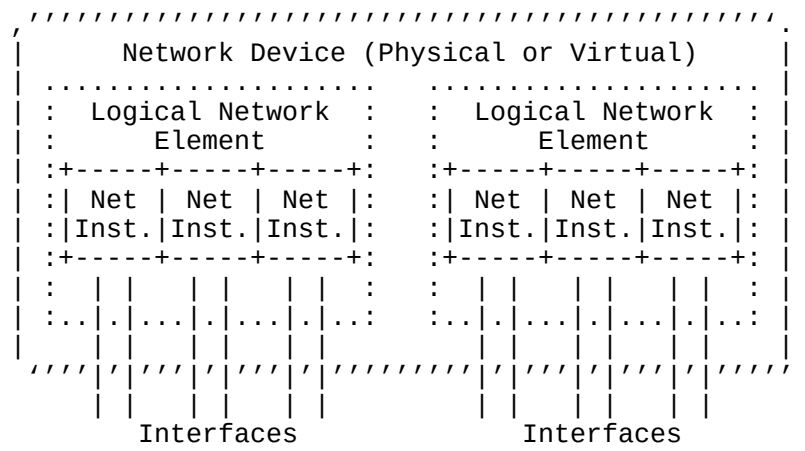


Figure 1: Module Element Relationships

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. An implementation MAY choose to restrict parent referenced information to information related to a specific instance, e.g., only allowing references to interfaces that have a "bind-network-instance-name" which is identical to the instance’s "name".

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

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

```
"ietf-yang-schema-mount:schema-mounts": {
  "mount-point": [
    {
      "module": "ietf-network-instance",
      "label": "vrf-root",
      "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.

To allow a client to understand the previously mentioned instance restrictions on parent referenced information, an implementation MAY

represent such restrictions in the "parent-reference" leaf-list. For example:

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

3.4. Network Instance Instantiation

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

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

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

4. Security Considerations

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@2017-12-04.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 2017-12-04 {
      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;
          mandatory true;
          description
            "device scoped identifier for the network
             instance.";
        }
        leaf enabled {
          type boolean;
          default "true";
          description
            "Flag indicating whether or not the network
             instance is enabled.";
        }
        leaf description {
          type string;
          description
            "Description of the network instance
             and its intended purpose.";
        }
        choice ni-type {
          description
            "This node serves as an anchor point for different types
             of network instances.  Each 'case' is expected to
             differ in terms of the information needed in the
             parent/core to support the NI, and may differ in their
             mounted schema definition. When the mounted schema is
             not expected to be the same for a specific type of NI
             a mount point should be defined.";
        }
        choice root-type {
          mandatory true;
          description
            "Well known mount points.";
          container vrf-root {
            description
              "Container for mount point.";
            yangmnt:mount-point "vrf-root" {
              description
                "Root for L3VPN type models. This will typically
                 not be an inline type mount point.";
            }
          }
          container vsi-root {
            description
              "Container for mount point.";
```

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

      // augment statements

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

           Note that a standard error will be returned if the
           identified leafref isn't present.  If an interfaces cannot
           be assigned for any other reason, the operation SHALL fail
           with an error-tag of 'operation-failed' and an
           error-app-tag of 'ni-assignment-failed'.  A meaningful
           error-info that indicates the source of the assignment
           failure SHOULD also be provided.";
        leaf bind-ni-name {
          type leafref {
            path "/network-instances/network-instance/name";
          }
          description
            "Network Instance to which an interface is bound.";
        }
      }
      augment "/if:interfaces/if:interface/ip:ipv4" {
        description
          "Add a node for the identification of the network
           instance associated with the information configured
           on an IPv4 interface.
```

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

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

      // notification statements

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

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

```
          At least one container with a bind-ni-name leaf MUST be
          included in this notification.";
        leaf name {
          type leafref {
            path "/if:interfaces/if:interface/if:name";
          }
          mandatory true;
          description
            "Contains the interface name associated with the
            failure.";
        }
        container interface {
          description
            "Generic interface type.";
          leaf bind-ni-name {
            type leafref {
              path "/if:interfaces/if:interface/ni:bind-ni-name";
            }
            description
              "Contains the bind-ni-name associated with the
               failure.";
          }
        }
        container ipv4 {
          description
            "IPv4 interface type.";
          leaf bind-ni-name {
            type leafref {
              path "/if:interfaces/if:interface"
                + "/ip:ipv4/ni:bind-ni-name";
            }
            description
              "Contains the bind-ni-name associated with the
               failure.";
          }
        }
        container ipv6 {
          description
            "IPv6 interface type.";
          leaf bind-ni-name {
            type leafref {
              path "/if:interfaces/if:interface"
                + "/ip:ipv6/ni:bind-ni-name";
            }
            description
              "Contains the bind-ni-name associated with the
               failure.";
          }
```

```
      }
      leaf error-info {
        type string;
        description
          "Optionally, indicates the source of the assignment
           failure.";
      }
    }
  }
  <CODE ENDS>
```

7. References

7.1. Normative References

[I-D.ietf-netmod-schema-mount]
Bjorklund, M. and L. Lhotka, "YANG Schema Mount", draft-ietf-netmod-schema-mount-08 (work in progress), October 2017.

[I-D.ietf-netmod-yang-tree-diagrams]
Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-04 (work in progress), December 2017.

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

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[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>.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <https://www.rfc-editor.org/info/rfc7223>.

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

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-07 (work in progress), October 2017.

[I-D.ietf-bess-l3vpn-yang]
Jain, D., Patel, K., Brissette, P., Li, Z., Zhuang, S., Liu, X., Haas, J., Esale, S., and B. Wen, "Yang Data Model for BGP/MPLS L3 VPNs", draft-ietf-bess-l3vpn-yang-02 (work in progress), October 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-09 (work in progress), October 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., Bogdanovic, D., and X. Liu, "YANG Logical Network Elements", draft-ietf-rtgwg-lne-model-05 (work in progress), December 2017.

[RFC4026] Andersson, L. and T. Madsen, "Provider Provisioned Virtual Private Network (VPN) Terminology", RFC 4026, DOI 10.17487/RFC4026, March 2005, <https://www.rfc-editor.org/info/rfc4026>.

[RFC4364] Rosen, E. and Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4364, DOI 10.17487/RFC4364, February 2006, <https://www.rfc-editor.org/info/rfc4364>.

[RFC4664] Andersson, L., Ed. and E. Rosen, Ed., "Framework for Layer 2 Virtual Private Networks (L2VPNs)", RFC 4664, DOI 10.17487/RFC4664, September 2006, <https://www.rfc-editor.org/info/rfc4664>.

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

[RFC8022] Lhotka, L. and A. Lindem, "A YANG Data Model for Routing Management", RFC 8022, DOI 10.17487/RFC8022, November 2016, <https://www.rfc-editor.org/info/rfc8022>.

Appendix A. Acknowledgments

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

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

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

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

```
          "urn:ietf:params:xml:ns:yang:ietf-interfaces",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-ip",
          "revision": "2014-06-16",
          "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ip",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-network-instance",
          "revision": "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-yang-schema-mount:schema-mounts": {
      "mount-point": [
        {
          "module": "ietf-network-instance",
          "label": "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"
```

```
            }
          ]
        }
      ]
    }
  }
```

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

Routing Area Working Group
Internet-Draft
Intended status: Informational
Expires: June 9, 2018

S. Litkowski
Orange Business Service
B. Decraene
Orange
M. Horneffer
Deutsche Telekom

December 6, 2017

Link State protocols SPF trigger and delay algorithm impact on IGP micro-loops
draft-ietf-rtgwg-spf-uloop-pb-statement-05

Abstract

A micro-loop is a packet forwarding loop that may occur transiently among two or more routers in a hop-by-hop packet forwarding paradigm.

In this document, we are trying to analyze the impact of using different Link State IGP implementations in a single network in regards of micro-loops. The analysis is focused on the SPF triggers and SPF delay algorithm.

Requirements Language

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

Status of This Memo

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

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

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

This Internet-Draft will expire on June 9, 2018.

Table of Contents

1. Introduction 2
2. Problem statement 3
3. SPF trigger strategies 5
4. SPF delay strategies 5
4.1. Two steps SPF delay 5
4.2. Exponential backoff 6
5. Mixing strategies 7
6. Proposed work items 11
7. Security Considerations 13
8. Acknowledgements 13
9. IANA Considerations 13
10. References 13
10.1. Normative References 13
10.2. Informative References 13
Authors' Addresses 14

1. Introduction

Link State IGP protocols are based on a topology database on which an SPF (Shortest Path First) algorithm like Dijkstra is implemented to find the optimal routing paths.

Specifications like IS-IS ([RFC1195]) propose some optimizations of the route computation (See Appendix C.1) but not all the implementations are following those not mandatory optimizations.

We will call "SPF trigger", the events that would lead to a new SPF computation based on the topology.

Link State IGP protocols, like OSPF ([RFC2328]) and IS-IS ([RFC1195]), are using multiple timers to control the router behavior

in case of churn: SPF delay, PRC delay, LSP generation delay, LSP flooding delay, LSP retransmission interval...

Some of those timers are standardized in protocol specification, some are not especially the SPF computation related timers.

For non standardized timers, implementations are free to implement it in any way. For some standardized timer, we can also see that rather than using static configurable values for such timer, implementations may offer dynamically adjusted timers to help controlling the churn.

We will call "SPF delay", the timer that exists in most implementations that specifies the required delay before running SPF computation after a SPF trigger is received.

A micro-loop is a packet forwarding loop that may occur transiently among two or more routers in a hop-by-hop packet forwarding paradigm. We can observe that these micro-loops are formed when two routers do not update their Forwarding Information Base (FIB) for a certain prefix at the same time. The micro-loop phenomenon is described in [I-D.ietf-rtgwg-microloop-analysis].

Some micro-loop mitigation techniques have been defined by IETF (e.g. [RFC6976], [I-D.ietf-rtgwg-uloop-delay]) but are not implemented due to complexity or are not providing a complete mitigation.

In multi-vendor networks, using different implementations of a link state protocol may favor micro-loops creation during the convergence process due to discrepancies of timers. Service Providers are already aware to use similar timers for all the network as a best practice, but sometimes it is not possible due to limitations of implementations.

This document will present why it sounds important for service providers to have consistent implementations of Link State protocols across vendors. We are particularly analyzing the impact of using different Link State IGP implementations in a single network in regards of micro-loops. The analysis is focused on the SPF triggers and the SPF delay algorithm.

This document is only stating the problem, and defining some work items but its not intended to provide a solution.

2. Problem statement

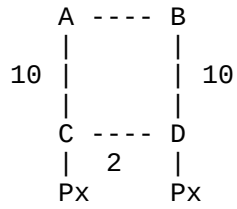


Figure 1 - Network topology suffering from micro-loops

In Figure 1, A uses primarily the AC link to reach C. When the AC link fails, the IGP convergence occurs. If A converges before B, A will forward the traffic to C through B, but as B as not converged yet, B will loop back traffic to A, leading to a micro-loop.

The micro-loop appears due to the asynchronous convergence of nodes in a network when an event occurs.

Multiple factors (and combination of these factors) may increase the probability for a micro-loop to appear:

- the delay of failure notification: the more B is advised of the failure later than A, the more a micro-loop may have a chance to appear.

- the SPF delay: most of the implementations supports a delay for the SPF computation to try to catch as many events as possible. If A uses an SPF delay timer of x msec and B uses an SPF delay timer of y msec and x < y, B would start converging after A leading to a potential micro-loop.

- the SPF computation time: mostly a matter of CPU power and optimizations like incremental SPF. If A computes its SPF faster than B, there is a chance for a micro-loop to appear. CPUs are today faster enough to consider SPF computation time as negligeable (order of msec in a large network).

- the RIB and FIB prefix insertion speed or ordering: highly implementation dependant.

This document will focus on analysis SPF delay (and associated triggers).

3. SPF trigger strategies

Depending of the change advertised in LSP/LSA, the topology may be affected or not. An implementation may avoid running the SPF computation (and may only run IP reachability computation instead) if the advertised change is not affecting topology.

Different strategies exists to trigger the SPF computation:

1. An implementation may always run a full SPF whatever the change to process.

2. An implementation may run a full SPF only when required: e.g. if a link fails, a local node will run an SPF for its local LSP update. If the LSP from the neighbor (describing the same failure) is received after SPF has started, the local node can decide that a new full SPF is not required as the topology has not change.

3. If the topology does not change, an implementation may only recompute the IP reachability.

As pointed in Section 1, SPF optimizations are not mandatory in specifications, leading to multiple strategies to be implemented.

4. SPF delay strategies

Implementations of link state routing protocols use different strategies to delay the SPF computation. We usually see the following:

1. Two steps delay.

2. Exponential backoff delay.

Those behavior will be explained in the next sections.

4.1. Two steps SPF delay

The SPF delay is managed by four parameters:

- Rapid delay: amount of time to wait before running SPF.

- Rapid runs: amount of consecutive SPF runs that can use the rapid delay. When the amount is exceeded the delay moves to the slow delay value .

- Slow delay: amount of time to wait before running SPF.

o Wait time: amount of time to wait without events before going back to the rapid delay.

Example: Rapid delay = 50msec, Rapid runs = 3, Slow delay = 1sec, Wait time = 2sec

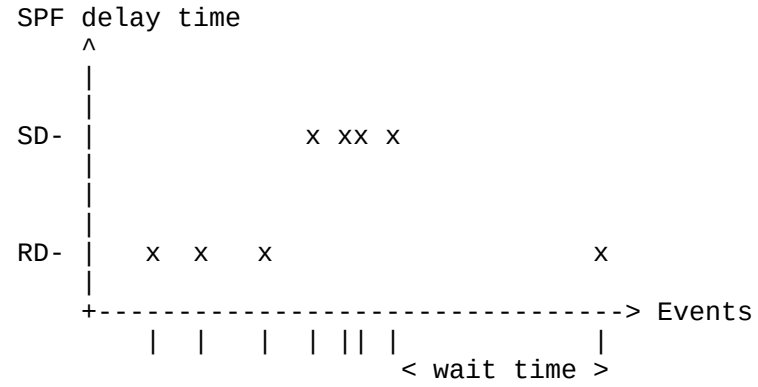


Figure 2 - Two steps delay algorithm

4.2. Exponential backoff

The algorithm has two modes: the fast mode and the backoff mode. In the fast mode, the SPF delay is usually delayed by a very small amount of time (fast reaction). When an SPF computation has run in the fast mode, the algorithm automatically moves to the backoff mode (a single SPF run is authorized in the fast mode). In the backoff mode, the SPF delay is increasing exponentially at each run. When the network becomes stable, the algorithm moves back to the fast mode. The SPF delay is managed by four parameters:

o First delay: amount of time to wait before running SPF. This delay is used only when SPF is in fast mode.

o Incremental delay: amount of time to wait before running SPF. This delay is used only when SPF is in backoff mode and increments exponentially at each SPF run.

o Maximum delay: maximum amount of time to wait before running SPF.

o Wait time: amount of time to wait without events before going back to the fast mode.

Example: First delay = 50msec, Incremental delay = 50msec, Maximum delay = 1sec, Wait time = 2sec

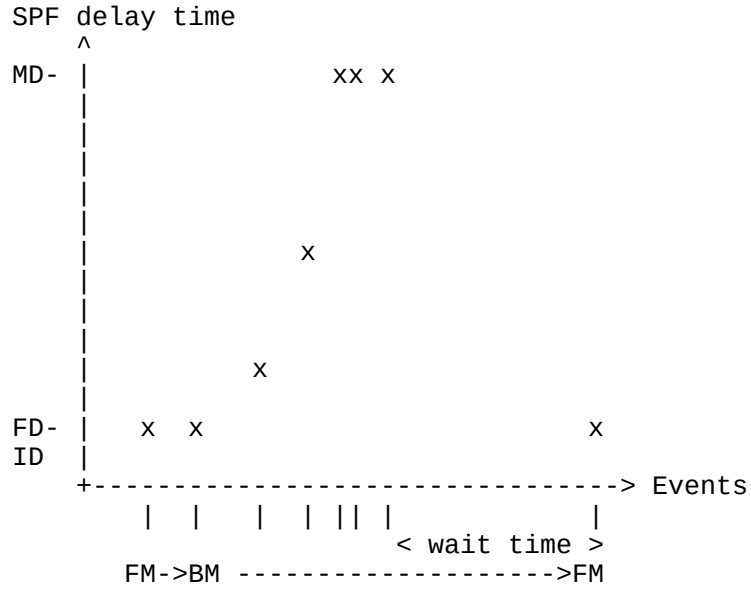


Figure 3 - Exponential delay algorithm

5. Mixing strategies

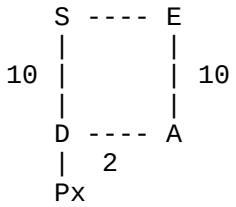


Figure 4

In Figure 4, we consider a flow of packet from S to D. We consider that S is using optimized SPF triggering (Full SPF is triggered only when necessary), and two steps SPF delay (rapid=150ms,rapid-runs=3, slow=1s). As implementation of S is optimized, Partial Reachability Computation (PRC) is available. We consider the same timers as SPF for delaying PRC. We consider that E is using a SPF trigger strategy that always compute Full SPF and exponential backoff strategy for SPF delay (start=150ms, inc=150ms, max=1s)

We also consider the following sequence of events (note : the time scale does not intend to represent a real router time scale where jitters are introduced to all timers) :

- t0=0 ms: a prefix is declared down in the network. We consider this event to happen at time=0.
- 200ms: the prefix is declared as up.
- 400ms: a prefix is declared down in the network.
- 1000ms: S-D link fails.

Time	Network Event	Router S events	Router E events
t0=0	Prefix DOWN		
10ms		Schedule PRC (in 150ms)	Schedule SPF (in 150ms)
160ms		PRC starts	SPF starts
161ms		PRC ends	
162ms		RIB/FIB starts	
163ms			SPF ends
164ms			RIB/FIB starts
175ms		RIB/FIB ends	
178ms			RIB/FIB ends
200ms	Prefix UP		
212ms		Schedule PRC (in 150ms)	
214ms			Schedule SPF (in 150ms)
370ms		PRC starts	
372ms		PRC ends	
373ms			SPF starts
373ms		RIB/FIB starts	
375ms			SPF ends
376ms			RIB/FIB starts
383ms		RIB/FIB ends	
385ms			RIB/FIB ends
400ms	Prefix DOWN		
410ms		Schedule PRC (in 300ms)	Schedule SPF (in 300ms)

710ms		PRC starts	SPF starts	
711ms		PRC ends		
712ms		RIB/FIB starts		
713ms			SPF ends	
714ms			RIB/FIB starts	
716ms		RIB/FIB ends	RIB/FIB ends	
1000ms	S-D link DOWN			
1010ms		Schedule SPF (in 150ms)	Schedule SPF (in 600ms)	
1160ms		SPF starts		
1161ms		SPF ends		
1162ms	Micro-loop may start from here	RIB/FIB starts		
1175ms		RIB/FIB ends		
1612ms			SPF starts	
1615ms			SPF ends	
1616ms			RIB/FIB starts	
1626ms	Micro-loop ends		RIB/FIB ends	

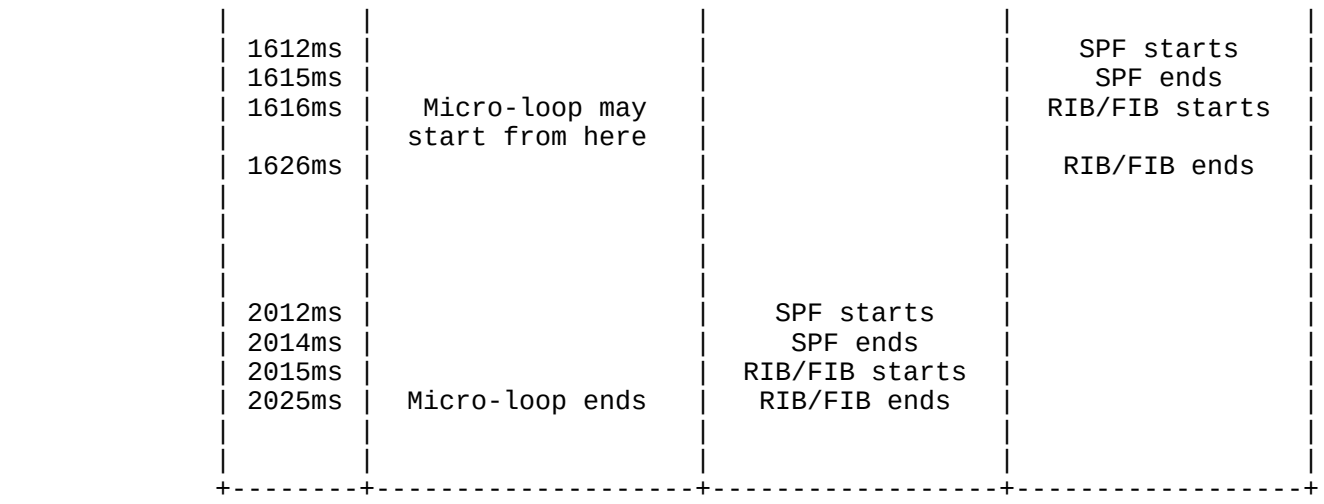
Route computation event time scale

In the table above, we can see that due to discrepancies in the SPF management, after multiple events (of a different type), the values of the SPF delay are completely misaligned between nodes leading to long micro-loops creation.

The same issue can also appear with only single type of events as displayed below:

Time	Network Event	Router S events	Router E events
t0=0	Link DOWN		
10ms		Schedule SPF (in 150ms)	Schedule SPF (in 150ms)

160ms		SPF starts	SPF starts	
161ms		SPF ends		
162ms		RIB/FIB starts		
163ms			SPF ends	
164ms			RIB/FIB starts	
175ms		RIB/FIB ends		
178ms			RIB/FIB ends	
200ms	Link DOWN			
212ms		Schedule SPF (in 150ms)		
214ms			Schedule SPF (in 150ms)	
370ms		SPF starts		
372ms		SPF ends		
373ms			SPF starts	
373ms		RIB/FIB starts		
375ms			SPF ends	
376ms			RIB/FIB starts	
383ms		RIB/FIB ends		
385ms			RIB/FIB ends	
400ms	Link DOWN			
410ms		Schedule SPF (in 150ms)	Schedule SPF (in 300ms)	
560ms		SPF starts		
561ms		SPF ends		
562ms	Micro-loop may start from here	RIB/FIB starts		
568ms		RIB/FIB ends		
710ms			SPF starts	
713ms			SPF ends	
714ms			RIB/FIB starts	
716ms	Micro-loop ends		RIB/FIB ends	
1000ms	Link DOWN			
1010ms		Schedule SPF (in 1s)	Schedule SPF (in 600ms)	



Route computation event time scale

6. Proposed work items

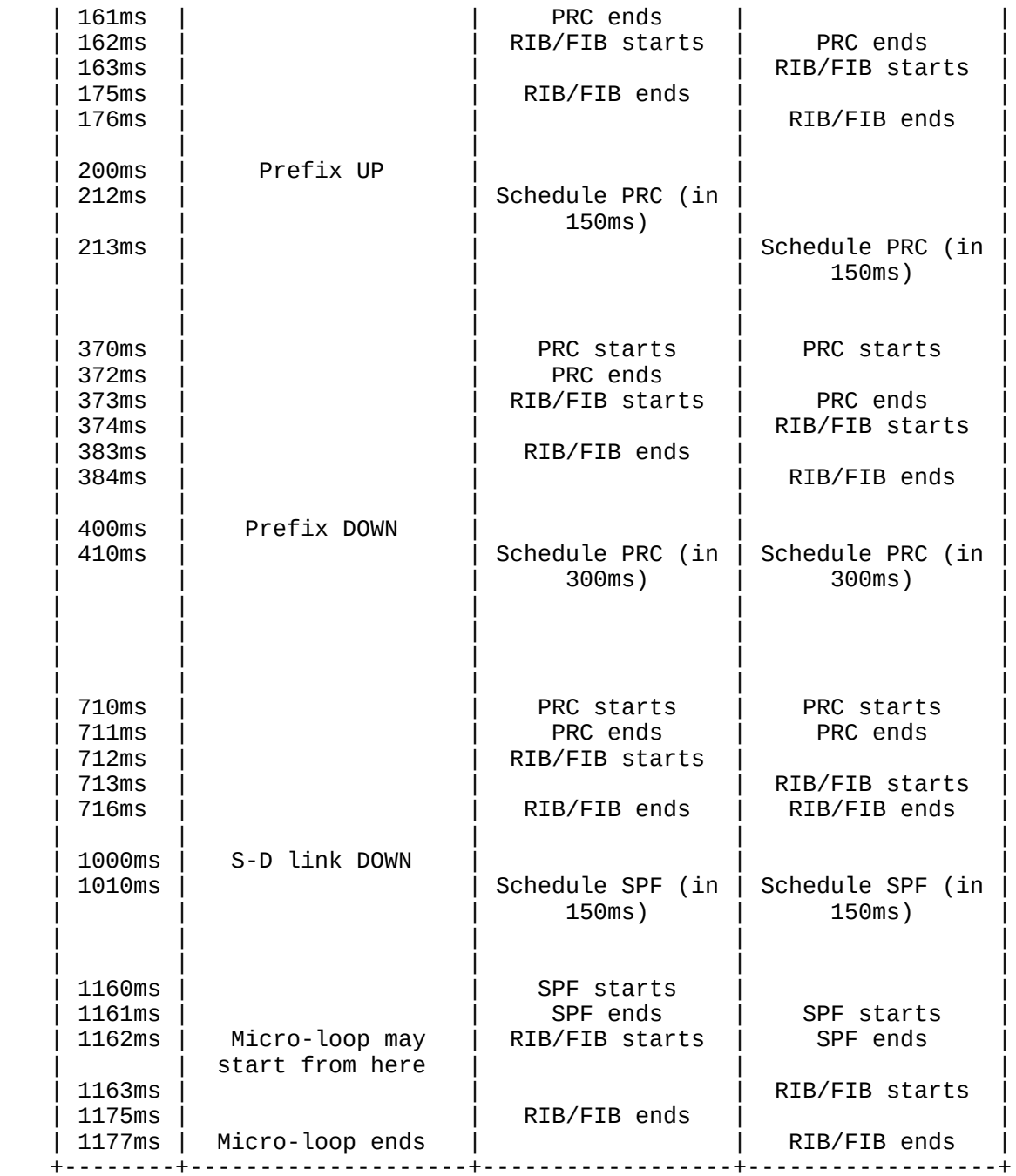
In order to enhance the current Link State IGP behavior, authors would encourage working on standardization of some behaviours.

Authors are proposing the following work items :

- Standardize SPF trigger strategy.
- Standardize computation timer scope: single timer for all computation operations, separated timers ...
- Standardize "slowdown" timer algorithm including its association to a particular timer: authors of this document does not presume that the same algorithm must be used for all timers.

Using the same event sequence as in figure 2, we may expect fewer and/or shorter micro-loops using standardized implementations.

Time	Network Event	Router S events	Router E events
t0=0	Prefix DOWN		
10ms		Schedule PRC (in 150ms)	Schedule SPF (in 150ms)
160ms		PRC starts	PRC starts



Route computation event time scale

As displayed above, there could be some other parameters like router computation power, flooding timers that may also influence micro-loops. In Figure 4, we consider E to be a bit slower than S, leading to micro-loop creation. Despite of this, we expect that by aligning implementations at least on SPF trigger and SPF delay, service provider may reduce the number and the duration of micro-loops.

7. Security Considerations

This document does not introduce any security consideration.

8. Acknowledgements

Authors would like to thank Mike Shand for his useful comments.

9. IANA Considerations

This document has no action for IANA.

10. References

10.1. Normative References

[RFC1195] Callon, R., "Use of OSI IS-IS for routing in TCP/IP and dual environments", RFC 1195, DOI 10.17487/RFC1195, December 1990, <https://www.rfc-editor.org/info/rfc1195>.

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

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

10.2. Informative References

[I-D.ietf-rtgwg-microloop-analysis] Zinin, A., "Analysis and Minimization of Microloops in Link-state Routing Protocols", draft-ietf-rtgwg-microloop-analysis-01 (work in progress), October 2005.

[I-D.ietf-rtgwg-uloop-delay] Litkowski, S., Decraene, B., Filsfils, C., and P. Francois, "Micro-loop prevention by introducing a local convergence delay", draft-ietf-rtgwg-uloop-delay-09 (work in progress), November 2017.

[RFC6976] Shand, M., Bryant, S., Previdi, S., Filsfils, C., Francois, P., and O. Bonaventure, "Framework for Loop-Free Convergence Using the Ordered Forwarding Information Base (oFIB) Approach", RFC 6976, DOI 10.17487/RFC6976, July 2013, <https://www.rfc-editor.org/info/rfc6976>.

Authors' Addresses

Stephane Litkowski
Orange Business Service

Email: stephane.litkowski@orange.com

Bruno Decraene
Orange

Email: bruno.decraene@orange.com

Martin Horneffer
Deutsche Telekom

Email: martin.horneffer@telekom.de

Network Working Group
Internet-Draft
Expires: March 16, 2018

F. Devetak
S. Kapoor
Illinois Institute of Technology

September 12, 2017

Dynamic MultiPath Routing
draft-kapoor-rtgwg-dynamic-multipath-routing-01

Abstract

In this draft we consider dynamic multipath routing and introduce two methods that use additive increase and multiplicative decrease for flow control, similar to TCP. Our first method allows for congestion control and re-routing flows as users join in or leave the network. As the number of applications and services supported by the Internet grows, bandwidth requirements increase dramatically so it is imperative to design methods to ensure not only that network throughput is maximized but also to ensure a level of fairness in network resource allocation. Our second method provides fairness over multiple streams of traffic. We drive the multiplicative decrease part of the algorithm with link queue occupancy data provided by an enhanced routing protocol.

1. Introduction

Internet packet traffic keeps growing as the number of applications and services it supports as well as their bandwidth requirements explode. It has then become necessary to find ways to ensure that network throughput is maximized. In this draft we propose dynamic multi-path routing to improve network throughput.

Multipath routing is important, not only for throughput but also for reliability and security. In multipath routing, improvements in performance are achieved by utilizing more than one feasible path [M75]. This approach to routing makes for more effective network resource utilization. Various research on multipath routing have addressed network redundancy, congestion, and QoS issues [CRS99] [ST92]. Prior work on multipath routing includes work on bounding delays as well as delay variance [DSAK11] [SKDA13].

The prior work is primarily from the viewpoint of static network design but, in practice, congestion control is necessary to prevent some user flows from being choked due to link bottlenecks. Single path routing implementations of TCP achieve that by rate control on specified paths. TCP is able to handle elastic traffic from applications and establishes a degree of fairness by reducing the rate of transmission rapidly upon detecting congestion. Regular TCP has been shown to provide Pareto-optimal allocation of resources [PU12]. However, unlike the single path approach of TCP, we consider multipath routing with associated issues of path selection and congestion. We may note that multipath TCP (MPTCP) has been studied extensively [RG10] [GWH11] [RH10] [AP13] with a number of IETF proposals [M05] [M06] [M07] [M08]. Prior work on multipath TCP is defined over a specific set of paths and the choice of paths or the routing is independent of congestion control; determining the right number of paths thus becomes a problem. The variation of throughput with the number of paths has been illustrated in [RG10] [GWH11]

Along with consideration of congestion, we also need to ensure a level of fairness in network resource allocation. Factoring fairness into the protocol is important in order to prevent some user’s flows from suffering due to bottlenecks in some links. Based on mathematical optimization formulations, we consider route

determination methods that ensure fairness where all users can achieve at least a minimum percentage of their demand. We introduce an algorithm that uses additive increase and multiplicative decrease for flow control and we have experiments to illustrate its stability and convergence. The algorithm may be considered as a generalization of TCP.

We have performed an extensive set of simulations using the NS-3 simulation environment. In our implementation we drive the multiplicative decrease part of the algorithm using queue occupancy data at each outgoing network link, with that data provided by an enhanced routing protocol. For a more in-depth evaluation of the algorithm’s performance we simulated not only the fairness algorithm but also a version of the same without the fairness component. We also performed and compared simulations using standard TCP and TCP with ECMP enabled.

2. Joint Routing and Congestion Control: Preliminaries

The Joint Routing and Congestion Control utilizes the Link State of the network. The algorithm utilizes a price variable that models congestion at each link and a variable that models the fairness coefficient. The fairness coefficient is used to establish the same percentage of traffic is being routed for multiple source-sink pairs.

2.1. The Price function

Each edge of the network has a price function associated with it, referred to as P(e). The price function measures the congestion on the link. The price function lies in the interval [0,1] and is 0 if the edge occupancy is low and 1 if the edge occupancy is high.

In theory the edge occupancy is given by f(e)/C(e) where f(e) is the amount of traffic on the link and C(e) is the capacity of the link. In practice, the edge occupancy is measured by the congestion in the queue serving the link.

The price function increases as the congestion grows. This function’s values will be referred to by a price variable on link e which is denoted by PBQ(e)

2.2. The Fairness function

The price function is complemented by an "increase" function, i.e. a variable that regulates the amount of traffic changes based on the fraction of traffic of the commodity that has been routed. This function, th values represented by the variable PBF(s-t), is used to model the fairness. This variable is related to the source-

destination pair, denoted by s-t, whose requirements are being satisfied.

The variable PBF(s-t) starts with an initial value and goes down to zero as the requirement of s-t is being increasingly met. The increase in PBF is dictated by a fairness co-efficient, Gamma.

The formula for PBF(s-t) is 1- y(s-t)/[Gamma *d(s-t)] where Gamma is the fairness co-efficient, d(s-t) is the demand and y(s-t) is the amount of requirement that is being met.

3. Joint Routing and Congestion Control: Algorithm

We present the details of the algorithms below

3.1. Preliminaries

Let T be the time interval used to increment or modify routing.

We use two coefficients for each path Pi

1. Additive increase coefficient: A positive value ai by which we increment the flow on a path at each iteration xi(t) = xi(t-T) + ai

2. Multiplicative decrease coefficient: The coefficient bi that we apply to decrement flows: xi(t) = (1-bi)xi(t-T) where x, t, T are the same as above.

We utilize multiple methods for calculating bi:

METHOD 1: bi may be computed as follows:

1. bi =0 if no edge on the path Pi is congested.

2. bi =0.5 if one edge on the path Pi is congested.

3. bi= 1.0 if more than one edge on the path Pi is congested

METHOD 2: bi may be computed as follows:

1. bi = 1-1/2^c where c is the number of congested edges.

3.2. The Basic Multi-path Algorithm

We propose two routing mechanisms. The first, presented here,is a basic mechanism that is primarily based on multiplicative decrease and additive increase

In the first routing method, for each commodity c, let P(c) be the set of paths being used. If any of the paths Pi is congested, the flow on the path is reduced using the multiplier (1-bi). Next, the shortest path is found to push additional flow requirements if they exist. An additive flow of value ai, which is chosen to be a constant independent of i, is pushed on that path.

```
 At the end of each time interval T
        FOR each  commodity c:
        Calculate commodity flow
                FOR each flow path, i, of commodity c
                        Calculate bi
                        IF {demand is met and bi = 0}
                                No change in path flow
                        ELSE
                                Apply coefficient bi to
                                        decrease path flow
                        ENDIF
                ENDFOR
                IF {demand not met}
                        Find shortest path for commodity c
                        IF{shortest Path is new}
                                Add new path to list
                        ENDIF
                        Increment shortest path flow by a
                ENDIF
        ENDFOR
```

If the number of paths used is excessive then no new paths need be generated.

3.3. The Multi-path Algorithm with Fairness

In order to ensure that different source-sink pairs are treated fairly, the coefficient bi for path Pi is chosen as PBQ - PBF with two components: a congestion component PBQ and a fairness component PBF .

1. PBQ is calculated as bi before;

2. PBF is calculated using the formula PBF(s-t) = 1-Total_current_FLOW(S-t)/(Gamma* Demand(s-t))

where Gamma is the fairness parameter and DEMAND(s-t) is the demand.

In the second routing method, again, for each commodity c, let P(c) be the set of paths being used. If any of the paths is congested, the flow on the path Pi is reduced using the multiplier (1-bi). The key difference is in the computation of bi. bi is not uniform across various user requests but is dependent on the fraction of flow of that commodity that is already being serviced by the network. Having reduced congestion, if it exists, a shortest path is found to push additional flow requirements if they exist. Again an additive flow of value ai, which in our current implementation is chosen to be a constant independent of i, is pushed on that path.

```
At the end of each time interval T
FOR {each  commodity}
        Calculate commodity c flow
        FOR {each path i}
                Calculate PBQ and PBF
                IF {demand is met}
                        IF {NO Congestion}
                                 No change in path flow
                        ELSE
                                bi = max (0, PBQ-PBF)
                                flow on path i,
                                xi(t) = (1-bi)*xi(t-T))
                        ENDIF
                ELSE
                        IF {NO Congestion}
                                bi = -PBF
                        ELSE
                                bi = max (0, PBQ-PBF)
                        ENDIF
                        xi(t) = a + (1-bi)*xi(t-T))
                ENDIF
        ENDFOR
        Recalculate Commodity flow
        IF {flow change in any path AND demand not met}
                Find shortest path
                IF {shortest path is new}
                         Add new path to list
                ENDIF
                Increment shortest path flow by a
        ENDIF
ENDFOR
```

If the number of paths become excessive then they can be curtailed. At that stage no additional flow is pushed until congestion is relieved.

4. Experiments

We implemented [1] the two algorithms using the NS-3 simulation environment. We modeled the network topology on the network of a large service provider, with link capacities proportional to capacities in the actual physical network. In our implementation we used, for routing, a combination of link-state routing protocol and source routing. For the link-state part we augmented the NS-3 implementation of the OSPF routing protocol by adding link queue occupancy to the data exchanged by nodes in Link State Advertisement (LSA) messages, a minimal increase in LSA data. That allows for more sophisticated monitoring of network status: if the queue occupancy for one of more links of a path exceeds a given threshold we conclude that the path is experiencing congestion and that the multiplicative decrease has to be applied to adjust the allocation of flow to the paths. The additive increase is applied at each iteration, if demand is not met, to augment the sending rate. Details of congestion measurement are as follows: In a node-to-node connection (data link) both the source (originating) node and the sink (destination) node have a PointToPointNetDevice. The PointToPointNetDevice at the originating node is associated with a queue function of type DropTailQueue. The queue function stores the number of packets in the queue (waiting to be sent to the destination node). A queueOccupancy parameter is calculated: numPacketsInQueue/ queueMaxSize and compared to a queueThreshold parameter. If queueOccupancy> queueThreshold the path that uses the data link is flagged as congested. queueThreshold is an input parameter declared at the start of the simulation. At the same time path delay is also calculated. The source node uses OSPF to find the shortest path to the destination and, based on available network data, builds a source-routing vector that is inserted in the packet and used by intermediate nodes to route the packet to the destination. To implement the source-routing function we augmented the NS-3 Nix-Vector protocol that builds the source-routing vector from the list of nodes to be traversed, list that is obtained from OSPF. The main process is iterative as we refresh LSA at a fixed interval: for our simulations we experimented with updating LSAs every 50 ms and 500 ms.

Our results for the throughput improvement are presented below and compared with throughput results for the standard TCP and TCP using ECMP. We show the average throughput over multiple runs.

No of Commodities	TCP	TCPw ECMP	DMP	DMP/Fairness
5	4008	3330	7949	4139
10	3275	2966	6017	5612
15	2849	2715	5373	5090
20	2912	2582	4633	3703

5. Conclusion

In conclusion we found that our algorithm with fairness provides throughput improvement over both regular TCP and TCP with ECMP. In addition, its ability to discover additional path dynamically eliminates the need to set a preselected set of paths, allowing the spreading of the traffic load amongst a wider but still reasonable set of paths. Further results may be found at www.cs.iit.edu/~kapoor/papers/reducerate.pdf .

6. References

6.1. References

[AP13] Amer, P. and F. Pang, "Non-renegable selective acknowledgments (nr-sacks) for mptcp.", Proceedings of the 2013 27th International Conference on Advanced Information Networking and Applications Workshops , 2013.

[CRS99] Cidon, I., Rom, R., and Y. Shavitt, "Analysis of multi-path routing", IEEE/ACM Trans on Networking pages 885-896, 1999.

[DSAK11] Devetak, F., Shin, J., Anjali, T., and S. Kapoor, "Minimizing path delay in multipath networks", IEEE ICC, 2011.

[GWH11] Greenhalgh, A., Wischik, D., Handley, M., and C. Raiciu, "Design, implementation and evaluation of congestion control for multipath tcp.", 8th USENIX conference on Networked systems design and implementation , 2011.

[M05] Internet Engineering Task Force, "Coupled congestion control for multipath transport protocols", RFC 6356 , 2011.

[M06] Internet Engineering Task Force, "Multipath tcp (mptcp) application interface considerations", RFC 6897 , 2013.

[M07] Internet Engineering Task Force, "Tcp extensions for multipath operation with multiple addresses", RFC 6824 , 2013.

[M08] Internet Engineering Task Force, "Architectural guidelines for multipath tcp development", RFC 6182 , 2011.

[M75] Maxemchuk, N., "Dispersity Routing", IEEE ICC, 1975.

[PU12] Popovic, M., Upadhyay, U., Le Boudec, J., Khalili, R., and N. Gast, "Mptcp is not pareto-optimal: performance issues and a possible solution", Proceedings of the 8th international conference on Emerging networking experiments and technologies , 2012.

[RG10] Barre, S., Greenhalgh, A., Wischik, D., Handley, M., Raiciu, C., and C. Pluntke, "Data center networking with multipath tcp.", 9th ACM SIGCOMM , 2010.

[RH10] Raiciu, C., Handley, M., Barre, S., and O. Bonaventure, "Experimenting with multipath tcp.", Proceedings of the ACM SIGCOMM 2010 conference , 2010.

[SKDA13] Devetak, F., Anjali, T., Shin, J., and S. Kapoor, "Concurrent multipath routing over bounded paths: Minimizing delay variance", Globecom 2013 , 2013.

[ST92] Suzuki, H. and F. Tobagi, "Fast bandwidth reservation with multiline and multipath routing in atm networks", Proceedings of IEEE Infocom pages 2233-2240, 1992.

6.2. URIs

[1] http:www.cs.iit.edu/˜kapoor/papers/reducerate.pdf

Authors' Addresses

Fabrizio Devetak
Illinois Institute of Technology
10W 31 Street
Stuart Building
Chicago, IL 60565
US

Sanjiv Kapoor
Illinois Institute of Technology
10W 31 Street
Stuart Building
Chicago, IL 60565
US

Phone: +1 312 567 5329
Email: kapoor@iit.edu
URI: http:www.cs.iit.edu/~kapoor

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

K. Patel
Arrcus, Inc.
A. Lindem
Cisco Systems
S. Zandi
Linkedin
G. Van de Velde
Nokia

January 12, 2018

Shortest Path Routing Extensions for BGP Protocol
draft-keyupate-idr-bgp-spf-04.txt

Abstract

Many Massively Scaled Data Centers (MSDCs) have converged on simplified layer 3 routing. Furthermore, requirements for operational simplicity have lead many of these MSDCs to converge on BGP as their single routing protocol for both their fabric routing and their Data Center Interconnect (DCI) routing. This document describes a solution which leverages BGP Link-State distribution and the Shortest Path First algorithm similar to Internal Gateway Protocols (IGPs) such as OSPF.

Status of This Memo

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

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

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

This Internet-Draft will expire on July 16, 2018.

Copyright Notice

Table of Contents

1. Introduction 3
1.1. BGP Shortest Path First (SPF) Motivation 4
1.2. Requirements Language 5
2. BGP Peering Models 5
2.1. BGP Single-Hop Peering on Network Node Connections . . . 5
2.2. BGP Peering Between Directly Connected Network Nodes . . 5
2.3. BGP Peering in Route-Reflector or Controller Topology . . 6
3. BGP-LS Shortest Path Routing (SPF) SAFI 6
4. Extensions to BGP-LS 6
4.1. Node NLRI Usage and Modifications 6
4.2. Link NLRI Usage 7
4.3. Prefix NLRI Usage 7
4.4. BGP-LS Attribute Sequence-Number TLV 8
5. Decision Process with SPF Algorithm 9
5.1. Phase-1 BGP NLRI Selection 9
5.2. Dual Stack Support 10
5.3. NEXT_HOP Manipulation 10
5.4. NLRI Advertisement and Convergence 10
5.5. Error Handling 11
6. IANA Considerations 11
7. Security Considerations 12
7.1. Acknowledgements 12
7.2. Contributorss 12
8. References 12

8.1. Normative References 12
8.2. Information References 13
Authors' Addresses 14

1. Introduction

Many Massively Scaled Data Centers (MSDCs) have converged on simplified layer 3 routing. Furthermore, requirements for operational simplicity have lead many of these MSDCs to converge on BGP [RFC4271] as their single routing protocol for both their fabric routing and their Data Center Interconnect (DCI) routing. Requirements and procedures for using BGP are described in [RFC7938]. This document describes an alternative solution which leverages BGP-LS [RFC7752] and the Shortest Path First algorithm similar to Internal Gateway Protocols (IGPs) such as OSPF [RFC2328].

[RFC4271] defines the Decision Process that is used to select routes for subsequent advertisement by applying the policies in the local Policy Information Base (PIB) to the routes stored in its Adj-RIBs-In. The output of the Decision Process is the set of routes that are announced by a BGP speaker to its peers. These selected routes are stored by a BGP speaker in the speaker's Adj-RIBs-Out according to policy.

[RFC7752] describes a mechanism by which link-state and TE information can be collected from networks and shared with external components using BGP. This is achieved by defining NLRI carried within BGP-LS AFI and BGP-LS SAFIs. The BGP-LS extensions defined in [RFC7752] makes use of the Decision Process defined in [RFC4271].

This document augments [RFC7752] by replacing its use of the existing Decision Process. The BGP-LS-SPF and BGP-LS-SPF-VPN AFI/SAFI are introduced to insure backward compatibility. The Phase 1 and 2 decision functions of the Decision Process are replaced with the Shortest Path Algorithm (SPF) also known as the Dijkstra Algorithm. The Phase 3 decision function is also simplified since it is no longer dependent on the previous phases. This solution avails the benefits of both BGP and SPF-based IGPs. These include TCP based flow-control, no periodic link-state refresh, and completely incremental NLRI advertisement. These advantages can reduce the overhead in MSDCs where there is a high degree of Equal Cost Multi-Path (ECMPs) and the topology is very stable. Additionally, using a SPF-based computation can support fast convergence and the computation of Loop-Free Alternatives (LFAs) [RFC5286] in the event of link failures. Furthermore, a BGP based solution lends itself to multiple peering models including those incorporating route-reflectors [RFC4456] or controllers.

Support for Multiple Topology Routing (MTR) as described in [RFC4915] is an area for further study dependent on deployment requirements.

1.1. BGP Shortest Path First (SPF) Motivation

Given that [RFC7938] already describes how BGP could be used as the sole routing protocol in an MSDC, one might question the motivation for defining an alternate BGP deployment model when a mature solution exists. For both alternatives, BGP offers the operational benefits of a single routing protocol. However, BGP SPF offers some unique advantages above and beyond standard BGP distance-vector routing.

A primary advantage is that all BGP speakers in the BGP SPF routing domain will have a complete view of the topology. This will allow support of ECMP, IP fast-reroute (e.g., Loop-Free Alternatives), Shared Risk Link Groups (SRLGs), and other routing enhancements without advertisement of addition BGP paths or other extensions. In short, the advantages of an IGP such as OSPF [RFC2328] are availed in BGP.

With the simplified BGP decision process as defined in Section 5.1, NLRI changes can be disseminated throughout the BGP routing domain much more rapidly (equivalent to IGPs with the proper implementation).

Another primary advantage is a potential reduction in NLRI advertisement. With standard BGP distance-vector routing, a single link failure may impact 100s or 1000s prefixes and result in the withdrawal or re-advertisement of the attendant NLRI. With BGP SPF, only the BGP speakers corresponding to the link NLRI need withdraw the corresponding BGP-LS Link NLRI. This advantage will contribute to both faster convergence and better scaling.

With controller and route-reflector peering models, BGP SPF advertisement and distributed computation require a minimal number of sessions and copies of the NLRI since only the latest verion of the NLRI from the originator is required. Given that verification of the adjacencies is done outside of BGP (see Section 2), each BGP speaker will only need as many sessions and copies of the NLRI as required for redundancy (e.g., one for SPF computation and another for backup). Functions such as Optimized Route Reflection (ORR) are supported without extension by virture of the primary advantages. Additionally, a controller could inject topology that is learned outside the BGP routing domain.

Given that controllers are already consuming BGP-LS NLRI [RFC7752], reusing for the BGP-LS SPF leverages the existing controller implementations.

Another potential advantage of BGP SPF is that both IPv6 and IPv4 can be supported in the same address family using the same topology. Although not described in this version of the document, multi-topology extensions can be used to support separate IPv4, IPv6, unicast, and multicast topologies while sharing the same NLRI.

Finally, the BGP SPF topology can be used as an underlay for other BGP address families (using the existing model) and realize all the above advantages. A simplified peering model using IPv6 link-local addresses as next-hops can be deployed similar to [RFC5549].

1.2. Requirements Language

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

2. BGP Peering Models

Depending on the requirements, scaling, and capabilities of the BGP speakers, various peering models are supported. The only requirement is that all BGP speakers in the BGP SPF routing domain receive link-state NLRI on a timely basis, run an SPF calculation, and update their data plane appropriately. The content of the Link NLRI is described in Section 4.2.

2.1. BGP Single-Hop Peering on Network Node Connections

The simplest peering model is the one described in section 5.2.1 of [RFC7938]. In this model, EBGP single-hop sessions are established over direct point-to-point links interconnecting the network nodes. For the purposes of BGP SPF, Link NLRI is only advertised if a single-hop BGP session has been established and the Link-State/SPF adddress family capability has been exchanged [RFC4790] on the corresponding session. If the session goes down, the NLRI will be withdrawn.

2.2. BGP Peering Between Directly Connected Network Nodes

In this model, BGP speakers peer with all directly connected network nodes but the sessions may be multi-hop and the direct connection discovery and liveliness detection for those connections are independent of the BGP protocol. How this is accomplished is outside the scope of this document. Consequently, there will be a single session even if there are multiple direct connections between BGP speakers. For the purposes of BGP SPF, Link NLRI is advertised as long as a BGP session has been established, the Link-State/SPF

address family capability has been exchanged [RFC4790] and the corresponding link is up and considered operational.

2.3. BGP Peering in Route-Reflector or Controller Topology

In this model, BGP speakers peer solely with one or more Route Reflectors [RFC4456] or controllers. As in the previous model, direct connection discovery and liveliness detection for those connections are done outside the BGP protocol. For the purposes of BGP SPF, Link NLRI is advertised as long as the corresponding link is up and considered operational.

3. BGP-LS Shortest Path Routing (SPF) SAFI

In order to replace the Phase 1 and 2 decision functions of the existing Decision Process with an SPF-based Decision Process and streamline the Phase 3 decision functions in a backward compatible manner, this draft introduces a couple AFI/SAFIs for BGP LS SPF operation. The BGP-LS-SPF (AF 16388 / SAFI TBD1) and BGP-LS-SPF-VPN (AFI 16388 / SAFI TBD2) [RFC4790] are allocated by IANA as specified in the Section 6.

4. Extensions to BGP-LS

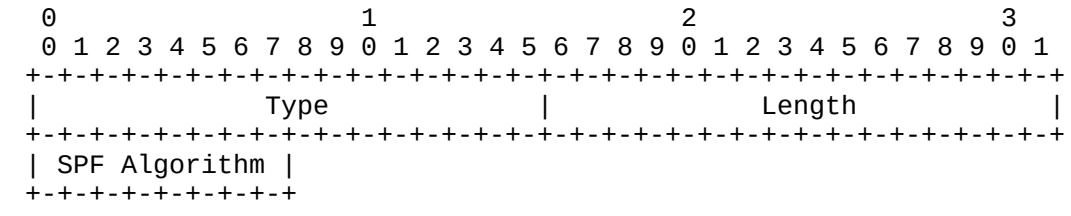
[RFC7752] describes a mechanism by which link-state and TE information can be collected from networks and shared with external components using BGP protocol. It contains two parts: definition of a new BGP NLRI that describes links, nodes, and prefixes comprising IGP link-state information and definition of a new BGP path attribute (BGP-LS attribute) that carries link, node, and prefix properties and attributes, such as the link and prefix metric or auxiliary Router-IDs of nodes, etc.

The BGP protocol will be used in the Protocol-ID field specified in table 1 of [I-D.ietf-idr-bgpls-segment-routing-epe]. The local and remote node descriptors for all NLRI will be the BGP Router-ID (TLV 516) and either the AS Number (TLV 512) [RFC7752] or the BGP Confederation Member (TLV 517) [I-D.ietf-idr-bgpls-segment-routing-epe]. However, if the BGP Router-ID is known to be unique within the BGP Routing domain, it can be used as the sole descriptor.

4.1. Node NLRI Usage and Modifications

The SPF capability is a new Node Attribute TLV that will be added to those defined in table 7 of [RFC7752]. The new attribute TLV will only be applicable when BGP is specified in the Node NLRI Protocol ID

field. The TBD TLV type will be defined by IANA. The new Node Attribute TLV will contain a single octet SPF algorithm field:



The SPF Algorithm may take the following values:

1 - Normal SPF
2 - Strict SPF

When computing the SPF for a given BGP routing domain, only BGP nodes advertising the SPF capability attribute will be included the Shortest Path Tree (SPT).

4.2. Link NLRI Usage

The criteria for advertisement of Link NLRI are discussed in Section 2.

Link NLRI is advertised with local and remote node descriptors as described above and unique link identifiers dependent on the addressing. For IPv4 links, the links local IPv4 (TLV 259) and remote IPv4 (TLV 260) addresses will be used. For IPv6 links, the local IPv6 (TLV 261) and remote IPv6 (TLV 262) addresses will be used. For unnumbered links, the link local/remote identifiers (TLV 258) will be used. For links supporting having both IPv4 and IPv6 addresses, both sets of descriptors may be included in the same Link NLRI. The link identifiers are described in table 5 of [RFC7752].

The link IGP metric attribute TLV (TLV 1095) as well as any others required for non-SPF purposes SHOULD be advertised. Algorithms such as setting the metric inversely to the link speed as done in the OSPF MIB [RFC4750] may be supported. However, this is beyond the scope of this document.

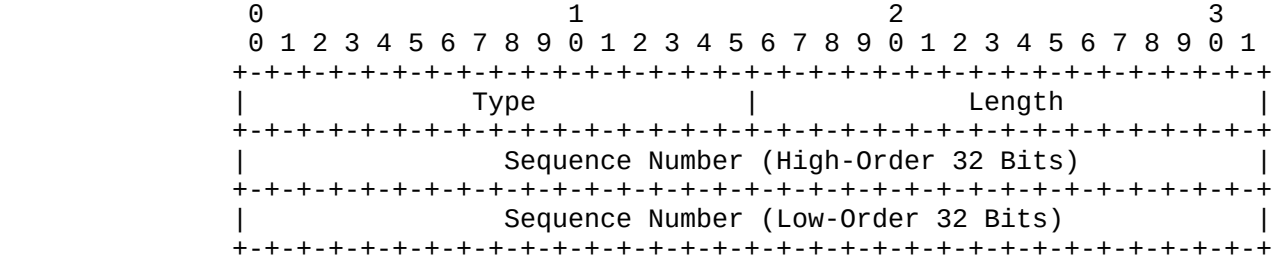
4.3. Prefix NLRI Usage

Prefix NLRI is advertised with a local descriptor as described above and the prefix and length used as the descriptors (TLV 265) as described in [RFC7752]. The prefix metric attribute TLV (TLV 1155) as well as any others required for non-SPF purposes SHOULD be

advertised. For loopback prefixes, the metric should be 0. For non-loopback, the setting of the metric is beyond the scope of this document.

4.4. BGP-LS Attribute Sequence-Number TLV

A new BGP-LS Attribute TLV to BGP-LS NLRI types is defined to assure the most recent version of a given NLRI is used in the SPF computation. The TBD TLV type will be defined by IANA. The new BGP-LS Attribute TLV will contain an 8 octet sequence number. The usage of the Sequence Number TLV is described in Section 5.1.



Sequence Number

The 64-bit strictly increasing sequence number is incremented for every version of BGP-LS NLRI originated. BGP speakers implementing this specification MUST use available mechanisms to preserve the sequence number's strictly increasing property for the deployed life of the BGP speaker (including cold restarts). One mechanism for accomplishing this would be to use the high-order 32 bits of the sequence number as a wrap/boot count that is incremented anytime the BGP Router router loses its sequence number state or the low-order 32 bits wrap.

When incrementing the sequence number for each self-originated NLRI, the sequence number should be treated as an unsigned 64-bit value. If the lower-order 32-bit value wraps, the higher-order 32-bit value should be incremented and saved in non-volatile storage. If by some chance the BGP Speaker is deployed long enough that there is a possibility that the 64-bit sequence number may wrap or a BGP Speaker completely loses its sequence number state (e.g, the BGP speaker hardware is replaced), the phase 1 decision function (see Section 5.1) rules should insure convergance, albeit, not immediately.

5. Decision Process with SPF Algorithm

The Decision Process described in [RFC4271] takes place in three distinct phases. The Phase 1 decision function of the Decision Process is responsible for calculating the degree of preference for each route received from a Speaker's peer. The Phase 2 decision function is invoked on completion of the Phase 1 decision function and is responsible for choosing the best route out of all those available for each distinct destination, and for installing each chosen route into the Loc-RIB. The combination of the Phase 1 and 2 decision functions is also known as a Path vector algorithm.

When BGP-LS-SPF NLRI is received, all that is required is to determine whether it is the best-path by examining the Node-ID and sequence number as described in Section 5.1. If the best-path NLRI had changed, it will be advertised to other BGP-LS-SPF peers. If the attributes have changed (other than the sequence number), a BGP SPF calculation will be scheduled. However, a changed best-path can be advertised to other peer immediately and propagation of changes can approach IGP convergence times.

The SPF based Decision process starts with selecting only those Node NLRI whose SPF capability TLV matches with the local BGP speaker's SPF capability TLV value. Since Link-State NLRI always contains the local descriptor [RFC7752], it will only be originated by a single BGP speaker in the BGP routing domain. These selected Node NLRI and their Link/Prefix NLRI are used to build a directed graph during the SPF computation. The best paths for BGP prefixes are installed as a result of the SPF process.

The Phase 3 decision function of the Decision Process [RFC4271] is also simplified since under normal SPF operation, a BGP speaker would advertise the NLRI selected for the SPF to all BGP peers with the BGP-LS/BGP-SPF AFI/SAFI. Application of policy would not be prevented but would normally not be necessary.

5.1. Phase-1 BGP NLRI Selection

The rules for NLRI selection are greatly simplified from [RFC4271].

1. If the NLRI is received from the BGP speaker originating the NLRI (as determined by the comparing BGP Router ID in the NLRI Node identifiers with the BGP speaker Router ID), then it is preferred over the same NLRI from non-originators.

2. If the Sequence-Number TLV is present in the BGP-LS Attribute, then the NLIR with the most recent, i.e., highest sequence number is selected. BGP-LS NLRI with a Sequence-Number TLV will be

considered more recent than NLRI without a BGP-LS or a BGP-LS Attribute that doesn't include the Sequence-Number TLV.

3. The final tie-breaker is the NLRI from the BGP Speaker with the numerically largest BGP Router ID.

The modified Decision Process with SPF algorithm uses the metric from Link and Prefix NLRI Attribute TLVs [RFC7752]. As a result, any attributes that would influence the Decision process defined in [RFC4271] like ORIGIN, MULTI_EXIT_DISC, and LOCAL_PREF attributes are ignored by the SPF algorithm. Furthermore, the NEXT_HOP attribute value is preserved and validated but otherwise ignored during the SPF or best-path.

5.2. Dual Stack Support

The SPF based decision process operates on Node, Link, and Prefix NLRIs that support both IPv4 and IPv6 addresses. Whether to run a single SPF instance or multiple SPF instances for separate AFs is a matter of a local implementation. Normally, IPv4 next-hops are calculated for IPv4 prefixes and IPv6 next-hops are calculated for IPv6 prefixes. However, an interesting use-case is deployment of [RFC5549] where IPv6 link-local next-hops are calculated for both IPv4 and IPv6 prefixes. As stated in Section 1, support for Multiple Topology Routing (MTR) is an area for future study.

5.3. NEXT_HOP Manipulation

A BGP speaker that supports SPF extensions MAY interact with peers that don't support SPF extensions. If the BGP Link-State address family is advertised to a peer not supporting the SPF extensions described herein, then the BGP speaker MUST conform to the NEXT_HOP rules mentioned in [RFC4271] when announcing the Link-State address family routes to those peers.

All BGP peers that support SPF extensions would locally compute the NEXT_HOP values as result of the SPF process. As a result, the NEXT_HOP attribute is always ignored on receipt. However BGP speakers should set the NEXT_HOP address according to the NEXT_HOP attribute rules mentioned in [RFC4271].

5.4. NLRI Advertisement and Convergence

A local failure will prevent a link from being used in the SPF calculation due to the IGP bi-directional connectivity requirment. Consequently, local link failues should always be given priority over updates (e.g., withdrawing all routes learned on a session) in order to ensure the highest priority progation and optimal convergence.

Delaying the withdrawal of non-local routes is an area for further study as more IGP-like mechanisms would be required to prevent usage of stale NLRI.

5.5. Error Handling

When a BGP speaker receives a BGP Update containing a malformed SPF Capability TLV in the Node NLRI BGP-LS Attribute [RFC7752], it MUST ignore the received TLV and the Node NLRI and not pass it to other BGP peers as specified in [RFC7606]. When discarding a Node NLRI with malformed TLV, a BGP speaker SHOULD log an error for further analysis.

6. IANA Considerations

This document defines a couple AFI/SAFIs for BGP LS SPF operation and requests IANA to assign the BGP-LS-SPF AFI 16388 / SAFI TBD1 and the BGP-LS-SPF-VPN AFI 16388 / SAFI TBD2 as described in [RFC4750].

This document also defines two attribute TLV for BGP LS NLRI. We request IANA to assign TLVs for the SPF capability and the Sequence Number from the "BGP-LS Node Descriptor, Link Descriptor, Prefix Descriptor, and Attribute TLVs" Registry. Additionally, IANA is requested to create a new registry for "BGP-LS SPF Capability Algorithms" for the value of the algorithm both in the BGP-LS Node Attribute TLV and the BGP SPF Capability. The initial assignments are:

Value(s)	Assignment Policy
0	Reserved (not to be assigned)
1	SPF
2	Strict SPF
3-254	Unassigned (IETF Review)
255	Reserved (not to be assigned)

BGP-LS SPF Capability Algorithms

7. Security Considerations

This extension to BGP does not change the underlying security issues inherent in the existing [RFC4724] and [RFC4271].

7.1. Acknowledgements

The authors would like to thank .... for the review and comments.

7.2. Contributorss

In addition to the authors listed on the front page, the following co-authors have contributed to the document.

Derek Yeung
Arrcus, Inc.
derek@arrcus.com

Abhay Roy
Cisco Systems
akr@cisco.com

Venu Venugopal
Cisco Systems
venuv@cisco.com

8. References

8.1. Normative References

[I-D.ietf-idr-bgpls-segment-routing-epe] Previdi, S., Filsfils, C., Patel, K., Ray, S., and J. Dong, "BGP-LS extensions for Segment Routing BGP Egress Peer Engineering", draft-ietf-idr-bgpls-segment-routing-epe-14 (work in progress), December 2017.

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

[RFC4271] Rekhter, Y., Ed., Li, T., Ed., and S. Hares, Ed., "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, DOI 10.17487/RFC4271, January 2006, <https://www.rfc-editor.org/info/rfc4271>.

[RFC7606] Chen, E., Ed., Scudder, J., Ed., Mohapatra, P., and K. Patel, "Revised Error Handling for BGP UPDATE Messages", RFC 7606, DOI 10.17487/RFC7606, August 2015, <https://www.rfc-editor.org/info/rfc7606>.

[RFC7752] Gredler, H., Ed., Medved, J., Previdi, S., Farrel, A., and S. Ray, "North-Bound Distribution of Link-State and Traffic Engineering (TE) Information Using BGP", RFC 7752, DOI 10.17487/RFC7752, March 2016, <https://www.rfc-editor.org/info/rfc7752>.

[RFC7938] Lapukhov, P., Premji, A., and J. Mitchell, Ed., "Use of BGP for Routing in Large-Scale Data Centers", RFC 7938, DOI 10.17487/RFC7938, August 2016, <https://www.rfc-editor.org/info/rfc7938>.

8.2. Information References

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

[RFC4456] Bates, T., Chen, E., and R. Chandra, "BGP Route Reflection: An Alternative to Full Mesh Internal BGP (IBGP)", RFC 4456, DOI 10.17487/RFC4456, April 2006, <https://www.rfc-editor.org/info/rfc4456>.

[RFC4724] Sangli, S., Chen, E., Fernando, R., Scudder, J., and Y. Rekhter, "Graceful Restart Mechanism for BGP", RFC 4724, DOI 10.17487/RFC4724, January 2007, <https://www.rfc-editor.org/info/rfc4724>.

[RFC4750] Joyal, D., Ed., Galecki, P., Ed., Giacalone, S., Ed., Coltun, R., and F. Baker, "OSPF Version 2 Management Information Base", RFC 4750, DOI 10.17487/RFC4750, December 2006, <https://www.rfc-editor.org/info/rfc4750>.

[RFC4790] Newman, C., Duerst, M., and A. Gulbrandsen, "Internet Application Protocol Collation Registry", RFC 4790, DOI 10.17487/RFC4790, March 2007, <https://www.rfc-editor.org/info/rfc4790>.

[RFC4915] Psenak, P., Mirtorabi, S., Roy, A., Nguyen, L., and P. Pillay-Esnault, "Multi-Topology (MT) Routing in OSPF", RFC 4915, DOI 10.17487/RFC4915, June 2007, <https://www.rfc-editor.org/info/rfc4915>.

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

[RFC5549] Le Faucheur, F. and E. Rosen, "Advertising IPv4 Network Layer Reachability Information with an IPv6 Next Hop", RFC 5549, DOI 10.17487/RFC5549, May 2009, <https://www.rfc-editor.org/info/rfc5549>.

Authors' Addresses

Keyur Patel
Arrcus, Inc.

Email: keyur@arrcus.com

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

Email: acee@cisco.com

Shawn Zandi
Linkedin
222 2nd Street
San Francisco, CA 94105
USA

Email: szandi@linkedin.com

Gunter Van de Velde
Nokia
Antwerp
Belgium

Email: gunter.van_de_velde@nokia.com

Network Working Group   Z. Li
Internet-Draft   J. Zhang
Intended status: Informational   Huawei Technologies
Expires: May 4, 2017   October 31, 2016

An Architecture of Network Artificial Intelligence(NAI)
draft-li-rtgwg-network-ai-arch-00

Abstract

Artificial intelligence is an important technical trend in the industry. With the development of network, it is necessary to introduce artificial intelligence technology to achieve self-adjustment, self- optimization, self-recovery of the network through collection of huge data of network state and machine learning. This draft defines the architecture of Network Artificial Intelligence (NAI), including the key components and the key protocol extension requirements.

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 inRFC 2119 [RFC2119]

Status of This Memo

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

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

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

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

Copyright Notice

Table of Contents

1. Introduction 2
2. Terminology 3
3. Architecture 3
3.1. Reference Model 3
3.2. Requirement of Protocol Extensions 4
4. IANA Considerations 5
5. Security Considerations 5
6. Normative References 5
Authors' Addresses 5

1. Introduction

Artificial Intelligence is an important technical trend in the industry. The two key aspects of Artificial Intelligence are perception and cognition. Artificial Intelligence has evolved from an early non-learning expert system to a learning-capable machine learning era. In recent years, the rapid development of the deep learning branch based on the neural network and the maturity of the big data technology and software distributed architecture make the Artificial Intelligence in many fields (such as transportation, medical treatment, education, etc.) have been applied. With the development of network, it is necessary to introduce artificial intelligence technology to achieve self-adjustment, self-optimization, self-recovery of the network through collection of huge data of network state and machine learning. The areas of machine learning which are easier to be used in the network field may include: troubleshooting of network problems, network traffic prediction, traffic optimization adjustment, security defense, security auditing, etc., to implement network perception and cognition.

This draft defines the architecture of Network Artificial Intelligence (NAI), including the key components and the key protocol extension requirements.

2. Terminology

AI: Artificial Intelligence

NAI: Network Artificial Intelligence

3. Architecture

3.1. Reference Model

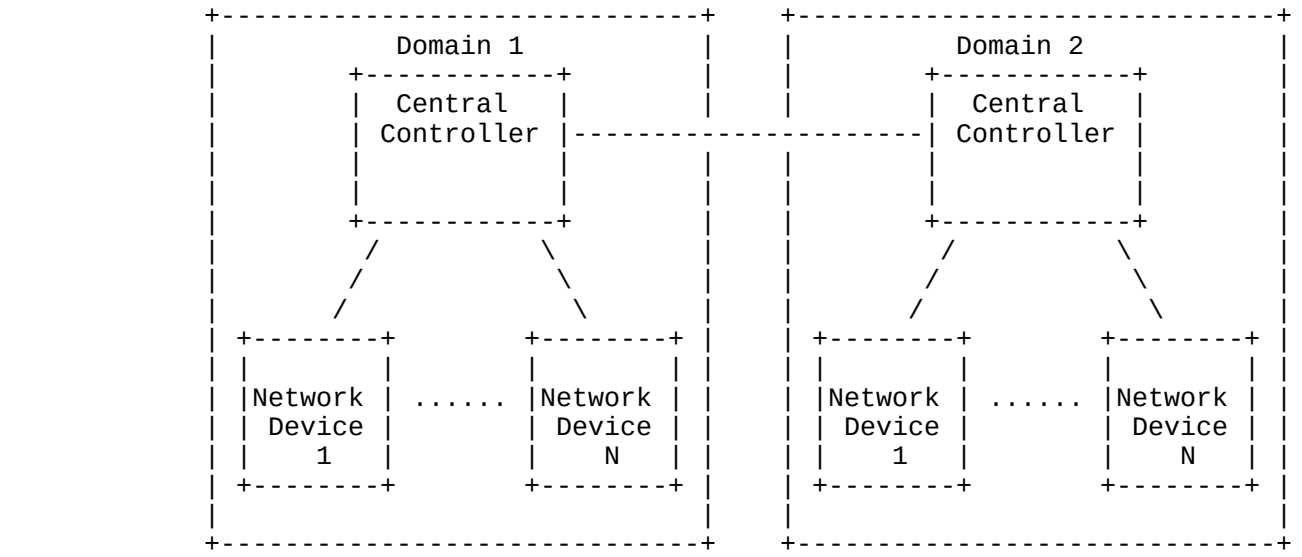


Figure 1: An Architecture of Network Artificial Intelligence(NAI)

The architecture of Network artificial intelligence includes following key component:

1. Central Controller: Centralized controller is the core component of Network Artificial Intelligence which can be called as 'Network Brain'. It man collect huge data of network states, store the data based on the big data platform, and carry on the machine learning, to achieve network perception and cognition, including network self-optimization, self- adjustment, self-recovery, intelligent fault location and a series of network artificial intelligence goals.

2. Network Device: IP network operation and maintenance are always a big challenge since the network can only provide limited state information. The network states includes but are not limited to topology, traffic engineering, operation and maintenance information, network failure information and related information to locate the

network failure.. In order to provide these information, the network must be able to support more OAM mechanisms to acquire more state information and report to the controller. Then the controller can get the complete state information of the network which is the base of Network Artificial Intelligence(NAI).

3. Southbound Protocol and Models of Controller: As network devices provide huge network state information, it proposes a number of new requirements for protocols and models between controllers and network devices. The traditional southbound protocol such as Netconf and SNMP can not meet the performance requirements. It is necessary to introduce some new high-performance protocols to collect network state data. At the same time, the models of network data should be completed. Moreover with the introduction of new OAM mechanisms of network devices, new models of network data should be introduced.

4. Northbound Model of Controller: The goal of the Network Artificial Intelligence is to reduce the technical requirements on the network administrators and release them from the heavy network management, control, maintenance work. The abstract northbound model of the controller for different network services should be simple and easy to be understood.

3.2. Requirement of Protocol Extensions

REQ 01: The new southbound protocol of the controller should be introduced to meet the performance requirements of collecting huge data of network states.

REQ 02: The models of network elements should be completed to collect the network states based on the new southbound protocol of the controller.

REQ 03: New OAM mechanisms should be introduced for the network devices in order to acquire more types of network state data.

REQ 04: New models of network elements should be introduced as the new OAM mechanisms are introduced.

REQ 05: The operation models of network elements should be completed based on the new southbound protocol to carry on the corresponding network operation as the result of Network Artificial Intelligence.

REQ 06: The abstract network-based service models should be provided by the controller as the northbound models to satisfy the requirements of different services.

4. IANA Considerations

This document makes no request of IANA.

5. Security Considerations

TBD.

6. 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>.

Authors' Addresses

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

Email: lizhenbin@huawei.com

Jinhui Zhang
Huawei Technologies
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095
China

Email: jason.zhangjinhui@huawei.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 14, 2017

Z. Li
K. Lu
Huawei Technologies
March 13, 2017

Inter-SDN (SDNi) in Seamless MPLS for Mobile Backhaul Network
draft-li-rtgwg-sdni-seamless-mpls-mbh-00

Abstract

This document introduces the inter-SDN framework of Seamless MPLS to integrate the mobile backhaul network with the core network.

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].

Status of This Memo

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

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

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

This Internet-Draft will expire on September 14, 2017.

Copyright Notice

Table of Contents

1. Introduction 2
2. Terminology 3
3. Requirements 3
4. Framework 3
4.1. Inter-SDN Architecture 4
4.2. Procedures 5
4.2.1. Traffic Optimization in Each Domain 5
4.2.2. End-to-End Traffic Optimization 7
4.2.3. End-to-End Service Provision 8
4.2.4. Auto Discovery 9
5. IANA Considerations 10
6. Security Considerations 10
7. Informative References 10
Authors' Addresses 11

1. Introduction

Seamless MPLS [I-D.ietf-mpls-seamless-mpls] describes an architecture which can be used to extend MPLS networks to integrate access and core/aggregation networks into a single MPLS domain. It provides a highly flexible and a scalable architecture and the possibility to integrate 100.000 of nodes. One of the key elements of Seamless MPLS is the separation of the service and transport plane: it can reduce the service specific configurations in network transport node.

The main purpose of Seamless MPLS is to deal with the integration of access networks and core/aggregation networks. The typical access devices taken into account are DSLAM(Digital Subscriber Link Access Multiplexer), etc. Now the mobile backhaul service has been deployed widely, the requirement of the integration of mobile backhaul networks and core networks has been proposed. At the same time, SDN is being developed to facilitate network operation and management. In the seamless MPLS for mobile backhaul, since there are multiple domains including the core network and multiple mobile backhaul networks, when SDN is introduced, for each domain there maybe one controller. In order to implement the end-to-end network service provision, there should be orchestration among multiple controllers. Thus the inter-SDN requirements are proposed in the Seamless MPLS architecture for mobile backhaul networks.

This document describes new requirements and framework for inter-SDN in the Seamless MPLS architecture for mobile backhaul network

2. Terminology

This document uses the following terminology:

- ABR: Area Border Router
- ASBR: AS Border Router
- PE: Provider Edge
- PCE: Path Calculation Element
- PCC: Path Calculation Client

3. Requirements

The framework of Seamless MPLS for mobile backhaul is introduced in [I-D.li-mpls-seamless-mpls-mbh]. There are following requirements for SDN in Seamless MPLS for mobile backhaul network:

1. Network Traffic Optimization in each Domain

For mobile service, there are different SLA requirement. For each domain, in order to satisfy the SLA requirement for the traffic in each domain, the path optimization is necessary. This means in each domain the MPLS traffic engineering based on SDN can be introduced to optimize the traffic path.

2. End-to-End Traffic Optimization

For seamless MPLS, the end-to-end label BGP LSP should be set up to bear the mobile service. In order to satisfy the SLA requirement for the traffic, it is necessary to implement the end-to-end traffic optimization. This means the SDN can be introduced to optimize the end-to-end BGP LSP.

3. End-to-End Service Provision

In the seamless MPLS, there may be complex provision work including MPLS LDP/TE, label BGP, L2VPN/L3VPN, etc. The SDN can be introduced to facilitate the service provision.

4. Framework

4.1. Inter-SDN Architecture

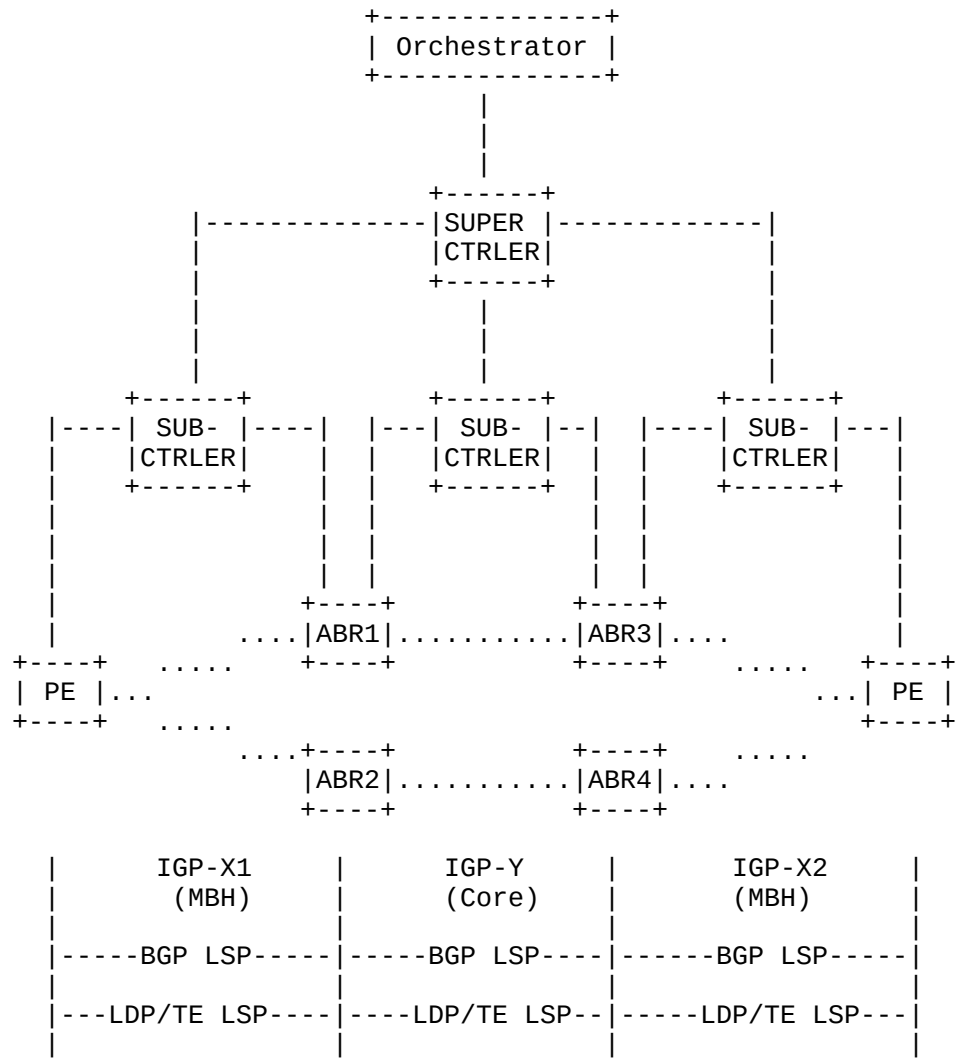


Figure 1 Inter-SDN Architecture

In the inter-SDN architecture, there are multiple components. The functionalities of components are introduced as follows:

1. Orchestrator

-- Provides E2E cross-controller orchestration, and downloads path optimization policy to manage traffic on demand. Orchestrator connects to the Super Controller by Netconf.

2. Super Controller

-- Network model management: Service/Network model abstraction. Report the network model to Orchestrator by Netconf interface.

-- Inter-domain topology management: domain edge topology management

-- Inter-domain LSP path management: Inter-domain BGP LSP path management. Calculate and establish BGP LSP path across different domain based on inter-domain topology, and inform sub-controller to establish LSP inside the domain. Super-controller communicate with sub-controller by Netconf Interface, getting inter-routing/link information, and transmitting LSP modification information.

-- Network configuration: Configuration according to the network model (Netconf).

3. Sub-Controller:

-- Network model management: network model abstraction. Report the network model to Super Controller by Netconf interface.

-- Intra-domain topology management: Collect Inner-domain topology by IGP-TE/BGP LS. Identify domain edge topology and report it to Super Controller by Netconf interface.

-- Intra-domain path management: Intra-domain service path management. Calculate/establish LSP path Inner-domain based on Inter-domain topology collected by BGP/BGP LS.

-- Network configuration: Configuration according to the network model (Netconf).

4.2. Procedures

4.2.1. Traffic Optimization in Each Domain

1. Topology Information Collection

The topology information needs to be collected for the traffic optimization. Both IGP and BGP LS [I-D.ietf-idr-ls-distribution]can satisfy the requirements. If there are multiple IGP areas in each mobile backhaul network and the end-to-end MPLS TE tunnels needs to set up, when IGP is adopted for topology collection, the Sub-

Controller needs to set up multiple IGP adjacencies to collect topology information of multiple IGP areas.

2. Traffic Optimization

When Sub-Controller is used for the path optimization in each domain, It can be acted as the PCE server. There are two different scenarios for the traffic optimization in each domain:

o Scenario 1: There is only MPLS TE tunnel optimization.

The MPLS TE tunnel optimization requirement can be sent from the Orchestrator to the Super Controller to the Sub-Controller. The protocol can be Netconf. The Yang model needs to be defined based on the [I-D.ji-i2rs-usecases-ccne-service].

There are two cases for Sub-Controller to optimize MPLS TE path:

Case 1: PCE will initiate the new path calculation. Then it will send the new path from the PCE to the PCC through PCEP to re-optimize the path for the existing tunnel in the distributed devices.

Case 2: PCE will initiate the new path calculation. And it will initiate setup of the new MPLS TE tunnel from the PCE to the PCC. The new tunnel will be created in the ingress LSR without configuration.

o Scenario 2: Label BGP LSP optimization triggers dynamic MPLS TE.

Label BGP LSP optimization requirement can be sent from the orchestrator to the Super Controller to the Sub-Controller. The protocol can be Netconf. The Yang model needs to be defined based on the [I-D.ji-i2rs-usecases-ccne-service].

The label BGP LSP optimization requirement will trigger the MPLS TE tunnel optimization in Sub-Controller. There are two cases:

Case 1: label BGP LSP reuses the existing MPLS TE tunnel. PCE will initiate the new path calculation. Then it will send the new path from the PCE to the PCC through PCEP to re-optimize the path for the existing tunnel in the distributed devices.

Case 2: There is no existing MPLS TE tunnel for the optimized label BGP route. PCE will initiate the new path calculation. And it will initiate setup of the new MPLS TE tunnel from the PCE to the PCC. The new tunnel will be created in the ingress LSR without configuration.

4.2.2. End-to-End Traffic Optimization

1. Label BGP Route Collection

BGP should run between the Sub-Controller and the distributed devices. And BGP should also run betweens the Super Controller and the Sub-Controllers. BGP Add-Paths [I-D.ietf-idr-add-paths] is enabled for all BGP peers. Then the Super Controller and the Sub-Controller can collect all the label BGP routes.

2. Topology Information Collection

IGP can be run between the Sub-Controller and the distributed devices to collect network topology.

There are two options for the Super Controller to collect topology information from the Sub-Controller.

Option 1: Collect all topology information from the Sub-Controller by the Super Controller: If so, for the reason of performance and scalability, it needs to run BGP-LS for the Super Controller to collect the topology information from the Sub-Controller.

Option 2: Collect abstract topology information from the Sub-Controller by the Super Controller: In this method, the Sub-Controller needs to abstract the network topology which means the whole network topology will not leak to the Super Controller. This is for the better scalability and to comply with the hierarchy principle. If the abstract topology information is reported, both BGP-LS and Netconf can be adopted owing to limited topology information.

3. Traffic Optimization for Label BGP LSP

-- The orchestrator can determine what label BGP LSP should be optimized and the constraints and the policy.

-- When Super Controller receives the optimization requirement, it can calculate the optimal path for the label BGP LSP based on the global network topology information. The result is that it may change to another BGP nexthop for the specific label BGP LSP. Or it need not change the nexthop for the specific label BGP LSP, but need to optimize the TE tunnel which bear the label BGP LSP. Then the Super Controller will download the different network optimization requirement to the Sub-Controller.

-- If only MPLS TE tunnel needs to optimize, please refer to the above process of traffic optimization in each domain. If the dynamic

TE result is to do LSP optimization for the tunnel (This means the label BGP LSP still uses the existing MPLS TE tunnel. It is just to do LSP make-before-break for the tunnel), there is nothing about the change of the label BGP route. If the result is to set up new MPLS TE tunnel (This means the BGP route will use the new MPLS TE tunnel), it will use the Netconf or BGP extensions to make the corresponding label BGP route in the network devices will use the new MPLS TE tunnel.

-- If the label BGP route needs to optimize and it may use the alternative nexthop, it will trigger the dynamic MPLS TE optimization firstly. Please refer to the above process of traffic optimization in each domain. Then the Sub-Controller will use the Netconf or BGP extensions to make the corresponding label BGP route to change the nexthop and correspondingly reuse the existing MPLS TE tunnel or use the new tunnel to the new nextop.

4.2.3. End-to-End Service Provision

1. MPLS TE Tunnel in Each Domain

For dynamic traffic engineering, the MPLS TE tunnel is not necessary to configure. As the PCE initiated LSP is adopted, the configuration for MPLS TE tunnel in the network devices can be reduced. And through the above process we can see that the MPLS TE tunnel can be triggered by label BGP LSP, the static MPLS TE tunnel will be removed gradually.

2. Label BGP Route

According to the principle of Seamless MPLS, the connectivity between any pair of nodes should exist to facilitate the service provision. So the BGP peer should always set up between the Super Controller and the Sub-Controller and set up between the Sub-Controller and the PE/ABR. This should be static configuration and need not to change frequently.

The challenge for the label BGP route provision is the limited capability of access nodes. In this case, BGP PUSH mode can not be adopted since the advertised routes may exceed the capability limitation of the access nodes. Then BGP PULL mode based on the BGP ORF extensions should be introduced between the Sub-Controller and the access node. It need depend on the L3VPN/L2VPN service provision which will be introduced in the next section.

3. L3VPN/L2VPN Service Provision

1) The Orchestrator will provide the simplified user-oriented VPN provision method based on the abstract topology information collected from the Super Controller. Then the VPN provision requirement will be downloaded to Super Controller through Netconf. The Yang model should be defined according to the user-oriented VPN.

2) When Super Controller receives the VPN provision requirement, it will convert the user-based VPN model to device-based VPN model. Then the following configuration can be calculated by the Super Controller:

-- the VPN configuration in PEs in different domains.

-- the BGP configuration for VPN in different domains.

-- the VPN interface configuration between PE and CE in different domains.

The configuration can be distributed through the Netconf from Super Controller to the Sub-Controller to the distributed devices.

3) When the device receives the BGP/VPN configuration, it can determine the remote BGP peers for the specific VPN service. If BGP PULL mode is adopted for the access node, it can trigger BGP ORF to get the corresponding label BGP route from the Sub-Controllers for the access node.

4.2.4. Auto Discovery

As the increasing of controller and network nodes, IGP and BGP extensions can be introduced for auto discovery. IGP-based PCE auto-discovery method ([RFC5088] and [RFC5089]) can be introduced between the PCE and the PCC. But the functionality of the Sub-Controller is not confined to PCE, these auto-discovery needs to be extended for the purpose of central control. [I-D.li-rtgwg-igp-cc-reqs] requirement defines the possible extensions requirements for auto-discovery, including:

o Advertisement of the role info of different components.

o Advertisement of the capability info of different components.

When BGP peer needs to setup between the Super Controller and the Sub-Controller, the BGP extension which is similar as the IGP extensions should be introduced for the auto-discovery.

5. IANA Considerations

This document makes no request of IANA.

6. Security Considerations

TBD.

7. Informative References

[I-D.ietf-idr-add-paths]
Walton, D., Retana, A., Chen, E., and J. Scudder, "Advertisement of Multiple Paths in BGP", draft-ietf-idr-add-paths-15 (work in progress), May 2016.

[I-D.ietf-idr-ls-distribution]
Gredler, H., Medved, J., Previdi, S., Farrel, A., and S. Ray, "North-Bound Distribution of Link-State and TE Information using BGP", draft-ietf-idr-ls-distribution-13 (work in progress), October 2015.

[I-D.ietf-mpls-seamless-mpls]
Leymann, N., Decraene, B., Filsfils, C., Konstantynowicz, M., and D. Steinberg, "Seamless MPLS Architecture", draft-ietf-mpls-seamless-mpls-07 (work in progress), June 2014.

[I-D.ji-i2rs-usecases-ccne-service]
Ji, X., Zhuang, S., Huang, T., and S. Hares, "I2RS Use Cases for Control of Forwarding Path by Central Control Network Element (CCNE)", draft-ji-i2rs-usecases-ccne-service-02 (work in progress), July 2014.

[I-D.li-mpls-seamless-mpls-mbh]
Li, Z., Li, L., Morillo, M., Yang, T., and L. Contreras, "Seamless MPLS for Mobile Backhaul Network", draft-li-mpls-seamless-mpls-mbh-00 (work in progress), July 2014.

[I-D.li-rtgwg-igp-cc-reqs]
Li, Z. and H. Chen, "Requirements of Interior Gateway Protocol (IGP) for Central Control", draft-li-rtgwg-igp-cc-reqs-00 (work in progress), July 2014.

[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>.

[RFC5088] Le Roux, JL., Ed., Vasseur, JP., Ed., Ikejiri, Y., and R. Zhang, "OSPF Protocol Extensions for Path Computation Element (PCE) Discovery", RFC 5088, DOI 10.17487/RFC5088, January 2008, <http://www.rfc-editor.org/info/rfc5088>.

[RFC5089] Le Roux, JL., Ed., Vasseur, JP., Ed., Ikejiri, Y., and R. Zhang, "IS-IS Protocol Extensions for Path Computation Element (PCE) Discovery", RFC 5089, DOI 10.17487/RFC5089, January 2008, <http://www.rfc-editor.org/info/rfc5089>.

Authors' Addresses

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

Email: lizhenbin@huawei.com

Kai Lu
Huawei Technologies
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095
China

Email: lukai1@huawei.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 14, 2017

R. Shakir
A. Shaikh
P. Borman
M. Hines
C. Lebsack
C. Morrow
Google

March 13, 2017

gRPC Network Management Interface (gNMI)
draft-openconfig-rtgwg-gnmi-spec-00

Abstract

This document describes the gRPC Network Management Interface (gNMI), a network management protocol based on the gRPC RPC framework. gNMI supports retrieval and manipuation of state from network elements where the data is represented by a tree structure, and addressable by paths. The gNMI service defines operations for configuration management, operational state retrieval, and bulk data collection via streaming telemtry.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 14, 2017.

Copyright Notice

Table of Contents

1. Introduction 3
2. Common Message Types and Encodings 4
2.1. Reusable Notification Message Format 4
2.2. Common Data Types 5
2.2.1. Timestamps 5
2.2.2. Paths 5
2.2.3. Node Values 6
2.3. Encoding Data in an Update Message 6
2.3.1. JSON and JSON_IETF 6
2.3.2. Bytes 9
2.3.3. Protobuf 9
2.3.4. ASCII 9
2.4. Use of Data Schema Paths 9
2.4.1. Path Prefixes 9
2.4.2. Path Aliases 10
2.4.3. Interpretation of Paths Used in RPCs 11
2.5. Error handling 12
2.6. Schema Definition Models 13
2.6.1. The ModelData message 13
3. Service Definition 14
3.1. Session Security, Authentication and RPC Authorization . 14
3.2. Capability Discovery 15
3.2.1. The CapabilityRequest message 15
3.2.2. The CapabilityResponse message 15
3.3. Retrieving Snapshots of State Information 16
3.3.1. The GetRequest Message 16
3.3.2. The GetResponse message 17
3.3.3. Considerations for using Get 18
3.4. Modifying State 18
3.4.1. The SetRequest Message 19
3.4.2. The SetResponse Message 20
3.4.3. Transactions 20
3.4.4. Modes of Update: Replace versus Update 21
3.4.5. Modifying Paths Identified by Attributes 21
3.4.6. Deleting Configuration 22
3.4.7. Error Handling 23
3.5. Subscribing to Telemetry Updates 24
3.5.1. Managing Subscriptions 26
3.5.2. Sending Telemetry Updates 32

4.1. URIs 35
Appendix A. Appendix: Current Protobuf Message and Service Specification 36
Appendix B. Appendix: Current Outstanding Issues/Future Features 36
Authors' Addresses 36

1. Introduction

This document defines a gRPC [1]-based protocol for the modification and retrieval of configuration from a network element, as well as the control and generation of telemetry streams from a network element to a data collection system. The intention is that a single gRPC service definition can cover both configuration and telemetry - allowing a single implementation on the network element, as well as a single NMS element to interact with the device via telemetry and configuration RPCs.

All messages within the gRPC service definition are defined as protocol buffers [2] (specifically proto3). gRPC service definitions are expected to be described using the relevant features of the protobuf IDL. A reference protobuf definition is maintained in [REFERENCE-PROTO] [3]. The current, authoritative version of this specification is available at [GNMI-SPEC] [4].

The service defined within this document is assumed to carry payloads that contain data instances of OpenConfig [5] YANG schemas, but can be used for any data with the following characteristics:

1. structure can be represented by a tree structure where nodes can be uniquely identified by a path consisting of node names, or node names coupled with attributes;

2. values can be serialised into a scalar object.

Currently, values may be serialised to a scalar object through encoding as a JSON string, a byte-array, or a serialised protobuf object - although the definition of new serialisations is possible.

Throughout this specification the following terminology is used:

- _Telemetry_ - refers to streaming data relating to underlying characteristics of the device - either operational state or configuration.

- _Configuration_ - elements within the data schema which are read/write and can be manipulated by the client.

- _Target_ - the device within the protocol which acts as the owner of the data that is being manipulated or reported on. Typically this will be a network device.

- _Client_ - the device or system using the protocol described in this document to query/modify data on the target, or act as a collector for streamed data. Typically this will be a network management system.

2. Common Message Types and Encodings

2.1. Reusable Notification Message Format

When a target wishes to communicate data relating to the state of its internal database to an interested client, it does so via means of a common "Notification" message. Notification messages are reused in other higher-layer messages for various purposes. The exact use of the Notification message is described on a per-RPC basis.

The fields of the Notification message are as follows:

- "timestamp" - The time at which the data was collected by the device from the underlying source, or the time that the target generated the Notification message (in the case that the data does not reflect an underlying data source). This value is always represented according to the definition in Section 2.2.1.

- "prefix" - a prefix which is applied to all path fields (encoded as per Section 2.2.2) included in the "Notification" message. The paths expressed within the message are formed by the concatenation of "prefix + path". The "prefix" always precedes the "path" elements. Further semantics of prefixes are described in Section 2.4.1.

- "alias"- a string providing an alias for the prefix specified within the notification message. The encoding of an alias, and the procedure for their creation is described in Section 2.4.2"."

- "update" - a list of update messages that indicate changes in the underlying data of the target. Both modification and creation of data is expressed through the update message.

  * An "Update" message has two subfields:

    + "path" - a path encoded as per Section 2.2.2.

    + "value" - a value encoded as per Section 2.2.3.

    * The set of paths that are specified within the list of updates MUST be unique. In this context, the path is defined to be the fully resolved path (including the prefix). In the case that there is a duplicate path specified within an update, only the final update should be processed by the receiving entity.

- "delete" - a list of paths (encoded as per Section 2.2.2) that indicate the deletion of data nodes on the target.

The creator of a Notification message MUST include the "timestamp" field. All other fields are optional.

2.2. Common Data Types

2.2.1. Timestamps

Timestamp values MUST be represented as the number of nanoseconds since the Unix epoch (January 1st 1970 00:00:00 UTC). The value MUST be encoded as a signed 64-bit integer ("int64").

2.2.2. Paths

Paths are represented according to gNMI Path Conventions [6], a simplified form of XPATH. Rather than utilising a single string to represent the path - with the "/" character separating each element of the path, the path is represented by an ordered list of strings, starting at the root node, and ending at the most specific path element.

A path is represented by the "Path" message with the following fields:

- "element" -- a set of path elements, encoded as strings (see examples below).

- "origin" - field which MAY be used to disambiguate the path if necessary. For example, the origin may be used to indicate which organization defined the schema to which the path belongs.

Each "Path" element should correspond to a node in the data tree. For example, the path "/a/b/c/d" is encoded as:

```
path: <
  element: "a"
  element: "b"
  element: "c"
  element: "d"
>
```

Where attributes are to be specified, these are encoded alongside the node name within the path element, for example a node specified by "/a/e[key=k1]/f/g" would have the path encoded as:

```
path: <
  element: "a"
  element: "e[key=k1]"
  element: "f"
  element: "g"
>
```

The root node ("/") is indicated by encoding a single path element which is an empty string, as per the following example:

```
path: <
  element: ""
>
```

Paths (defined to be the concatenation of the "Prefix" and "Path" within the message) specified within a message MUST be absolute - no messages with relative paths should be generated.

2.2.3. Node Values

The value of a data node is encoded as a two-field message:

- "bytes" - an arbitrary series of bytes which indicates the value of the node referred to within the message context.
- "type" - a field indicating the type of data contained in the bytes field. Currently defined types are:

2.3. Encoding Data in an Update Message

2.3.1. JSON and JSON_IETF

The "JSON" type indicates that the value included within the "bytes" field of the node value message is encoded as a JSON string. This format utilises the specification in RFC7159 [7]. Additional types (e.g., "JSON_IETF") are utilised to indicate specific additional characteristics of the encoding of the JSON data (particularly where they relate to serialisation of YANG-modeled data).

For any JSON encoding:

- In the case that the data item at the specified path is a leaf node (i.e., has no children) the value of that leaf is encoded

directly - i.e., the "bare" value is specified (i.e., a JSON object is not required, and a bare JSON value is included).

- Where the data item referred to has child nodes, the value field contains a serialised JSON entity (object or array) corresponding to the referenced item.

Using the following example data tree:

```
            root +
                 |
                 +-- a +
                       |
                       +-- b[name=b1] +
                                      |
                                      +-- c +
                                            |
                                            +-- d (string)
                                            +-- e (uint32)
```

The following serialisations would be used (note that the examples below follow the conventions for textproto, and Golang-style backticks are used for string literals that would otherwise require escaping):

For "/a/b[name=b1]/c/d":

```
                      update: <
                        path: <
                          element: "a"
                          element: "b[name=b1]"
                          element: "c"
                          element: "d"
                        >
                        value: <
                          value: "AStringValue"
                          type: JSON
                        >
                      >
```

For "/a/b[name=b1]/c/e":

```
                update: <
                  path: <
                    element: "a"
                    element: "b[name=b1]"
                    element: "c"
                    element: "e"
                  >
                  value: <
                    Value: 10042    // decoded byte array
                    type: JSON
                  >
                >
```

For "/a/b[name=b1]/c":

```
            update: <
              path: <
                element: "a"
                element: "b[name=b1]"
                element: "c"
              >
              value: <
                value: { "d": "AStringValue", "e": 10042 }
                type: JSON
              >
            >
```

For "/a" :

```
             update: <
               path: <
                 element: "a"
               >
               value: <
                 value: ‘{ "b": [
                                  {
                                    "name": "b1",
                                    "c": {
                                    "d": "AStringValue",
                                      "e": 10042
                                    }
                                   }
                                ]
                           }‘
                 type: JSON_IETF
               >
             >
```

Note that all JSON values MUST be valid JSON. That is to say, whilst a value or object may be included in the message, the relevant quoting according to the JSON specification in RFC7159 [8] must be used. This results in quoted string values, and unquoted number values.

"JSON_IETF" encoded data MUST conform with the rules for JSON serialisation described in RFC7951 [9]. Data specified with a type of JSON MUST be valid JSON, but no additional constraints are placed upon it. An implementation MUST NOT serialise data with mixed "JSON" and "JSON_IETF" encodings.

Both the client and target MUST support the JSON encoding as a minimum.

2.3.2. Bytes

The "BYTES" type indicates that the contents of the "bytes" field of the message contains a byte sequence whose semantics is opaque to the protocol.

2.3.3. Protobuf

The "PROTOBUF" type indicates that the contents of the "bytes" field of the message contains a serialised protobuf message. Note that in the case that the sender utilises this type, the receiver must understand the schema (and hence the type of protobuf message that is serialised) in order to decode the value. Such agreement is not guaranteed by the protocol and hence must be established out-of-band.

2.3.4. ASCII

The "ASCII" type indicates that the contents of the "bytes" field of the message contains system-formatted ASCII encoded text. For configuration data, for example, this may consist of semi-structured CLI configuration data formatted according to the target platform. The gNMI protocol does not define the format of the text - this must be established out-of-band.

2.4. Use of Data Schema Paths

2.4.1. Path Prefixes

In a number of messages, a prefix can be specified to reduce the lengths of path fields within the message. In this case, a "prefix" field is specified within a message - comprising of a valid path encoded according to Section Section 2.2.2 In the case that a prefix is specified, the absolute path is comprised of the concatenation of

the list of path elements representing the prefix and the list of path elements in the "path" field.

For example, again considering the data tree shown in Section Section 2.3.1 if a "Notification" message updating values, a prefix could be used to refer to the "/a/b[name=b1]/c/d" and "/a/b[name=b1]/c/e"data nodes:

```
     notification: <
         timestamp: (timestamp)      // timestamp as int64
         prefix: <
          element: "a"
           element: "b[name=b1]"
           element: "c"
         >
       update: <
         path: <
           element: "d"
         >
         value: <
           value: "AStringValue"
           type: JSON
         >
       >
       update: <
         path: <
           element: "e"
         >
         value: <
           value: 10042      // converted to int representation
           type: JSON
         >
       >
     >
```

2.4.2. Path Aliases

In some cases, a client or target MAY desire to utilises aliases for a particular path - such that subsequent messages can be compressed by utilising the alias, rather than using a complete representation of the path. Doing so reduces total message length, by ensuring that redundant information can be removed.

Support for path aliases MAY be provided by a target. In a case where a target does not support aliases, the maximum message length SHOULD be considered, especially in terms of bandwidth utilisation, and the efficiency of message generation.

A path alias is encoded as a string. In order to avoid valid data paths clashing with aliases (e.g., "a" in the above example), an alias name MUST be prefixed with a "#" character.

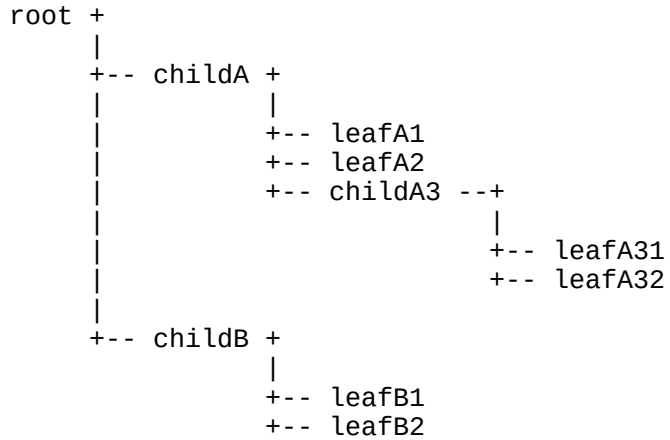
The means by which an alias is created is defined on a per-RPC basis. In order to delete an alias, the alias name is sent with the path corresponding to the alias empty.

Aliases MUST be specified as a fully expanded path, and hence MUST NOT reference other aliases within their definition, such that a single alias lookup is sufficient to resolve the absolute path.

2.4.3. Interpretation of Paths Used in RPCs

When a client specifies a path within an RPC message which indicates a read, or retrieval of data, the path MUST be interpreted such that it refers to the node directly corresponding with the path *and* all its children. The path refers to the direct node and all descendent branches which originate from the node, recursively down to each leaf element. If specific nodes are expected to be excluded then an RPC MAY provide means to filter nodes, such as regular-expression based filtering, lists of excluded paths, or metadata-based filtering (based on annotations of the data schema being manipulated, should such annotations be available and understood by both client and target).

For example, consider the following data tree:



A path referring to "root" (which is represented by a Path consisting of a single element specifying an empty string) should result in the nodes "childA" and "childB" and all of their children ("leafA1,

leafA2, leafB1, leafB2, childA3, leafA31" and "leafA32") being considered by the relevant operation.

In the case that the RPC is modifying the state of data (i.e., a write operation), such recursion is not required - rather the modification operation should be considered to be targeted at the node within the schema that is specified by the path, and the value should be deserialized such that it modifies the content of any child nodes if required to do so.

2.5. Error handling

Where the client or target wishes to indicate an error, an "Error" message is generated. Errors MUST be represented by a canonical gRPC error code (Java [10], Golang [11], C++ [12]) . The entity generating the error MUST specify a free-text string which indicates the context of the error, allowing the receiving entity to generate log entries that allow a human operator to understand the exact error that occurred, and its context. Each RPC defines the meaning of the relevant canonical error codes within the context of the operation it performs.

The canonical error code that is chosen MUST consider the expected behavior of the client on receipt of the message. For example, error codes which indicate that a client may subsequently retry SHOULD only be used where retrying the RPC is expected to result in a different outcome.

A re-usable "Error" message MUST be used when sending errors in response to an RPC operation. This message has the following fields:

- "code" - an unsigned 32-bit integer value corresponding to the canonical gRPC error code.
- "message"- a human-readable string describing the error condition in more detail. This string is not expected to be machine-parsable, but rather provide contextual information which may be passed to upstream systems.
- "data"- an arbitrary sequence of bytes (encoded as "[proto.Any](https://github.com/google/protobuf/blob/master/src/google/protobuf/any.proto)") which provides further contextual information relating to the error.

2.6. Schema Definition Models

The data tree supported by the target is expected to be defined by a set of schemas. The definition and format of these models is out of scope of this specification (YANG-modeled data is one example). In the case that such schema definitions are used, the client should be able to determine the models that are supported by the target, so that it can generate valid modifications to the data tree, and interpret the data returned by "Get" and "Subscribe" RPC calls.

Additionally, the client may wish to restrict the set of models that are utilised by the target so that it can validate the data returned to it against a specific set of data models. This is particularly relevant where the target may otherwise add new values to restricted value data elements (e.g., those representing an enumerated type), or augment new data elements into the data tree.

In order to allow the client to restrict the set of data models to be used when interacting with the target, the client MAY discover the set of models that are supported by the target using the "Capabilities" RPC described in Section 3.2. For subsequent "Get" and "Subscribe" RPCs, the client MAY specify the models to be used by the target. The set of models to use is expressed as a "ModelData" message, as specified in Section 2.6.1.

If the client specifies a set of models in a "Get" or "Subscribe" RPC, the target MUST NOT utilize data tree elements that are defined in schema modules outside the specified set. In addition, where there are data tree elements that have restricted value sets (e.g., enumerated types), and the set is extended by a module which is outside of the set, such values MUST NOT be used in data instances that are sent to the client. Where there are other elements of the schema that depend on the existence of such enumerated values, the target MUST NOT include such values in data instances sent to the client.

2.6.1. The ModelData message

The "ModelData" message describes a specific model that is supported by the target and used by the client. The fields of the "ModelData" message identify a data model registered in a model catalog, as described in [MODEL_CATALOG_DOC] [13] (the schema of the catalog itself - expressed in YANG - is described in [MODEL_CATALOG_YANG [14]]). Each model specified by a "ModelData" message may refer to a specific schema module, a bundle of modules, or an augmentation or deviation, as described by the catalog entry.

Each "ModelData" message contains the following fields:

- "name" - name of the model expressed as a string.
- "organization" - the organization publishing the model, expressed as a string.
- "version" - the supported (or requested) version of the model, expressed as a string which represents the semantic version of the catalog entry.

The combination of "name", "organization", and "version" uniquely identifies an entry in the model catalog.

3. Service Definition

A single gRPC service is defined - future revisions of this specification MAY result in additional services being introduced, and hence an implementation MUST NOT make assumptions that limit to a single service definition.

The service consists of the following RPCs:

- "Capabilities" - defined in Section 3.2 and used by the client and target as an initial handshake to exchange capability information
- "Get" - defined in Section 3.3, used to retrieve snapshots of the data on the target by the client.
- "Set" - defined in Section 3.4 and used by the client to modify the state of the target.
- "Subscribe" - defined in Section 3.5 and used to control subscriptions to data on the target by the client.

3.1. Session Security, Authentication and RPC Authorization

The session between the client and server MUST be encrypted using TLS - and a target or client MUST NOT fall back to unencrypted channels.

New connections are mutually authenticated -- each entity validates the X.509 certificate of the remote entity to ensure that the remote entity is both known, and authorized to connect to the local system.

If the target is expected to authenticate an RPC operation, the client MUST supply a username and password in the metadata of the RPC message (e.g., "SubscribeRequest", "GetRequest" or "SetRequest"). If the client supplies username/password credentials, the target MUST authenticate the RPC per its local authentication functionality.

Authorization is also performed per-RPC by the server, through validating client-provided metadata. The client MAY include the appropriate AAA metadata, which MUST contain a username, and MAY include a password in the context of each RPC call it generates. If the client includes both username and password, the target MUST authenticate and authorize the request. If the client only supplies the username, the target MUST authorize the RPC request.

A more detailed discussion of the requirements for authentication and encryption used for gNMI is in [GNMI-AUTH] [15].

3.2. Capability Discovery

A client MAY discover the capabilities of the target using the "Capabilities" RPC. The "CapabilityRequest" message is sent by the client to interrogate the target. The target MUST reply with a "CapabilityResponse" message that includes its gNMI service version, the versioned data models it supports, and the supported data encodings. This information is used in subsequent RPC messages from the client to indicate the set of models that the client will use (for "Get", "Subscribe" as described in Section 2.6) , and the encoding to be used for the data.

When the client does not specify the models it is using, the target SHOULD use all data schema modules that it supports when considering the data tree to be addressed. If the client does not specify the encoding in an RPC message, it MUST send JSON encoded values (the default encoding).

3.2.1. The CapabilityRequest message

The "CapabilityRequest" message is sent by the client to request capability information from the target. The "CapabilityRequest" message carries no additional fields.

3.2.2. The CapabilityResponse message

The "CapabilityResponse" message has the following fields:

- "supported_models" - a set of "ModelData" messages (as defined in Section 2.6.1) describing each of the models supported by the target
- "supported_encodings" - an enumeration field describing the data encodings supported by the target, as described in Section 2.3.

- "gNMI_version" - the semantic version of the gNMI service supported by the target, specified as a string. The version should be interpreted as per [OPENCONFIG-SEMVER [16]].

3.3. Retrieving Snapshots of State Information

In some cases, a client may require a snapshot of the state that exists on the target. In such cases, a client desires some subtree of the data tree to be serialized by the target and transmitted to it. It is expected that the values that are retrieved (whether writeable by the client or not) are collected immediately and provided to the client.

The "Get" RPC provides an interface by which a client can request a set of paths to be serialized and transmitted to it by the target. The client sends a "GetRequest" message to the target, specifying the data that is to be retrieved. The fields of the "GetRequest" message are described in Section 3.3.1.

Upon reception of a "GetRequest", the target serializes the requested paths, and returns a "GetResponse" message. The target MUST reflect the values of the specified leaves at a particular collection time, which MAY be different for each path specified within the "GetRequest" message.

The target closes the channel established by the "Get" RPC following the transmission of the "GetResponse" message.

3.3.1. The GetRequest Message

The "GetRequest" message contains the following fields:

- "prefix" - a path (specified as per Section 2.2.2), and used as described in Section 2.4.1. The prefix is applied to all paths within the "GetRequest" message.

- "path" - a set of paths (expressed as per Section 2.2.2) for which the client is requesting a data snapshot from the target. The path specified MAY utilize wildcards. In the case that the path specified is not valid, the target MUST populate the "error" field of the "GetResponse" message indicating an error code of "InvalidArgument" and SHOULD provide information about the invalid path in the error message.

- "type" - the type of data that is requested from the target. The valid values for type are described below.

- "encoding" - the encoding that the target should utilise to serialise the subtree of the data tree requested. The type MUST be one of the encodings specified in Section 2.3. If the "Capabilities" RPC has been utilised, the client SHOULD use an encoding advertised as supported by the target. If the encoding is not specified, JSON MUST be used. If the target does not support the specified encoding, the target MUST populate the error field of the "GetResponse" message, specifying an error of "InvalidArgument". The error message MUST indicate that the specified encoding is unsupported.

- "use_models" - a set of "ModelData" messages (defined in Section 2.6.1) indicating the schema definition modules that define the data elements that should be returned in response to the Get RPC call. The semantics of the "use_models" field are defined in Section 2.6.

Since the data tree stored by the target may consist of different types of data (e.g., values that are operational in nature, such as protocol statistics) - the client MAY specify that a subset of values in the tree are of interest. In order for such filtering to be implemented, the data schema on the target MUST be annotated in a manner which specifies the type of data for individual leaves, or subtrees of the data tree.

The types of data currently defined are:

- "CONFIG" - specified to be data that the target considers to be read/write. If the data schema is described in YANG, this corresponds to the "config true" set of leaves on the target.

- "STATE" - specified to be the read-only data on the target. If the data schema is described in YANG, "STATE" data is the "config false" set of leaves on the target.

- "OPERATIONAL" - specified to be the read-only data on the target that is related to software processes operating on the device, or external interactions of the device.

If the "type" field is not specified, the target MUST return CONFIG, STATE and OPERATIONAL data fields in the tree resulting from the client’s query.

3.3.2. The GetResponse message

The "GetResponse" message consists of:

- "notification" - a set of "Notification" messages, as defined in Section 2.1. The target MUST generate a "Notification" message for each path specified in the client's "GetRequest", and hence MUST NOT collapse data from multiple paths into a single "Notification" within the response. The "timestamp" field of the "Notification" message MUST be set to the time at which the target's snapshot of the relevant path was taken.

- "error" - an "Error" message encoded as per the specification in Section 2.5, used to indicate errors in the "GetRequest" received by the target from the client.

3.3.3. Considerations for using Get

The "Get" RPC is intended for clients to retrieve relatively small sets of data as complete objects, for example a part of the configuration. Such requests are not expected to put a significant resource burden on the target. Since the target is expected to return the entire snapshot in the "GetResponse" message, "Get" is not well-suited for retrieving very large data sets, such as the full contents of the routing table, or the entire component inventory. For such operations, the "Subscribe" RPC is the recommended mechanism, e.g. using the "ONCE" mode as described in Section 3.5.

Another consideration for "Get" is that the timestamp returned is associated with entire set of data requested, although individual data items may have been sampled by the target at different times. If the client requires higher accuracy for individual data items, the "Subscribe" RPC is recommended to request a telemetry stream (see Section 3.5.2).

3.4. Modifying State

Modifications to the state of the target are made through the "Set" RPC. A client sends a "SetRequest" message to the target indicating the modifications it desires.

A target receiving a "SetRequest" message processes the operations specified within it - which are treated as a transaction (see Section 3.4.3). The server MUST process deleted paths (within the "delete" field of the "SetRequest"), followed by paths to be replaced (within the "replace" field), and finally updated paths (within the "update" field). The order of the replace and update fields MUST be treated as significant within a single "SetRequest" message. If a single path is specified multiple times for a single operation (i.e., within "update" or "replace"), then the state of the target MUST reflect the application of all of the operations in order, even if they overwrite each other.

In response to a "SetRequest", the target MUST respond with a "SetResponse" message. For each operation specified in the "SetRequest" message, an "UpdateResult" message MUST be included in the response field of the "SetResponse". The order in which the operations are applied MUST be maintained such that "UpdateResult" messages can be correlated to the "SetRequest" operations. In the case of a failure of an operation, the "error" field of the "UpdateResult" message MUST be populated with an "Error" message as per the specification in Section 2.5. In addition, the "error" field of the "SetResponse" message MUST be populated with an error message indicating the success or failure of the set of operations within the "SetRequest" message (again using the error handling behavior defined in Section 2.5).

3.4.1. The SetRequest Message

A "SetRequest" message consists of the following fields:

- "prefix" - specified as per Section 2.4.1. The prefix specified is applied to all paths defined within other fields of the message.
- "delete" - A set of paths, specified as per Section 2.2.2, which are to be removed from the data tree. A specification of the behavior of a delete is defined in Section 3.4.6.
- "replace" - A set of "Update" messages indicating elements of the data tree whose content is to be replaced.
- "update" - A set of "Update" messages indicating elements of the data tree whose content is to be updated.

The semantics of "updating" versus "replacing" content are defined in Section 3.4.4

A re-usable "Update" message is utilised to indicate changes to paths where a new value is required. The "Update" message contains two fields:

- "path" - a path encoded as per Section 2.2.2 indicating the path of the element to be modified.
- "value" - a value encoded as per Section 2.2.3 indicating the value applied to the specified node. The semantics of how the node is updated is dependent upon the context of the update message, as specified in Section 3.4.4.

3.4.2. The SetResponse Message

A "SetResponse" consists of the following fields:

- "prefix" - specified as per Section 2.4.1. The prefix specified is applied to all paths defined within other fields of the message.

- "message" - an error message as specified in Section 2.5. The target SHOULD specify a "message" in the case that the update was successfully applied, in which case an error code of "OK (0)" "MUST" be specified. In cases where an update was not successfully applied, the contents of the error message MUST be specified as per Section 2.5.

- "response" - containing a list of responses, one per operation specified within the "SetRequest" message. Each response consists of an "UpdateResult" message with the following fields:

  * "timestamp" - a timestamp (encoded as per Section 2.2.1) at which the set request message was accepted by the system.

  * "path" - the path (encoded as per Section 2.2.2) specified within the "SetRequest". In the case that a common prefix was not used within the "SetRequest", the target MAY specify a "prefix" to reduce repetition of path elements within multiple "UpdateResult" messages in the "request" field.

  * "op" - the operation corresponding to the path. This value MUST be one of "DELETE", "REPLACE", or "UPDATE".

  * "message" - an error message (as specified in Section 2.5). This field follows the same rules as the message field within the "SetResponse" message specified above.

3.4.3. Transactions

All changes to the state of the target that are included in an individual "SetRequest" message are considered part of a transaction. That is, either all modifications within the request are applied, or the target MUST rollback the state changes to reflect its state before any changes were applied. The state of the target MUST NOT appear to be changed until such time as all changes have been accepted successfully. Hence, telemetry update messages MUST NOT reflect a change in state until such time as the intended modifications have been accepted.

As per the specification in Section 3.4, within an individual transaction ("SetRequest") the order of operations is "delete", "replace", "update".

As the scope of a "transaction" is a single "SetRequest" message, a client desiring a set of changes to be applied together MUST ensure that they are encapsulated within a single "SetRequest" message.

3.4.4. Modes of Update: Replace versus Update

Changes to read-write values on the target are applied based on the "replace" and "update" fields of the "SetRequest" message.

For both replace and update operations, if the path specified does not exist, the target MUST create the data tree element and populate it with the data in the "Update" message, provided the path is valid according to the data tree schema. If invalid values are specified, the target MUST cease processing updates within the "SetRequest" method, return the data tree to the state prior to any changes, and return a "SetResponse" message indicating the error encountered.

For "replace" operations, the behavior regarding omitted data elements in the "Update" depends on whether they refer to non-default values (i.e., set by a previous "SetRequest"), or unmodified defaults. When the "replace" operation omits values that have been previously set, they MUST be treated as deleted from the data tree. Otherwise, omitted data elements MUST be created with their default values on the target.

For "update" operations, only the value of those data elements that are specified explicitly should be treated as changed.

3.4.5. Modifying Paths Identified by Attributes

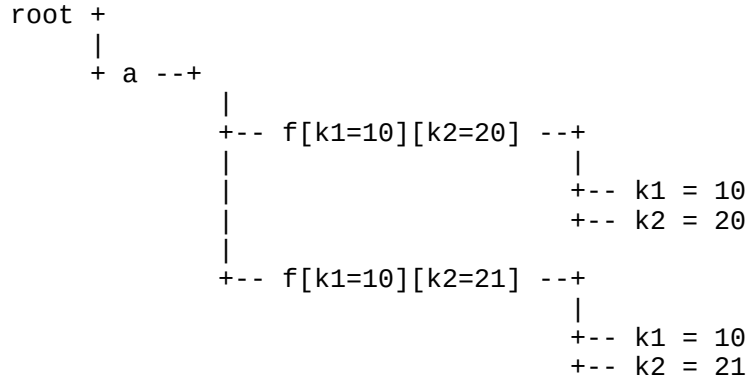
The path convention defined in Section 2.2.2 allows nodes in the data tree to be identified by a unique set of node names (e.g.,"/a/b/c/d") or paths that consist of node names coupled with attributes (e.g., "/a/e[key=10]"). In the case where where a node name plus attribute name is required to uniquely identify an element (i.e., the path within the schema represents a list, map, or array), the following considerations apply:

- In the case that multiple attribute values are required to uniquely address an element - e.g., "/a/f[k1=10][k2=20]"- and a replace or update operation’s path specifies a subset of the attributes (e.g., "/a/f[k1=10]") then this MUST be considered an error by the target system - and an error code of"InvalidArgument (3)" specified.

- Where the path specified refers to a node which itself represents the collection of objects (list, map, or array) a replace operation MUST remove all collection entries that are not supplied in the value provided in the "SetRequest". An update operation MUST be considered to add new entries to the collection if they do not exist.

- In the case that key values are specified both as attributes of a node, and as their own elements within the data tree, update or replace operations that modify instances of the key in conflicting ways MUST be considered an error. The target MUST return an error code of "InvalidArgument (3)".

For example, consider a tree corresponding to the examples above, as illustrated below.



In this case, nodes "k1" and "k2" are standalone nodes within the schema, but also correspond to attribute values for the node ""f"". In this case, an update or replace message specifying a path of "/a/f[k1=10][k2=20]" setting the value of "k1" to 100 MUST be considered erroneous, and an error code of "InvalidArgument (3)" specified.

3.4.6. Deleting Configuration

Where a path is contained within the "delete" field of the "SetRequest" message, it should be removed from the target’s data tree. In the case that the path specified is to an element that has children, these children MUST be recursively deleted. If a wildcard path is utilised, the wildcards MUST be expanded by the target, and the corresponding elements of the data tree deleted. Such wildcards MUST support paths specifying a subset of attributes required to identify entries within a collection (list, array, or map) of the data schema.

In the case that a path specifies an element within the data tree that does not exist, these deletes MUST be silently accepted.

3.4.7. Error Handling

When a client issues a "SetRequest", and the target is unable to apply the specified changes, an error MUST be reported to the client. The error is specified in multiple places:

- Within a "SetResponse" message, the error field indicates the completion status of the entire transaction.
- With a "UpdateResult" message, where the error field indicates the completion status of the individual operation.

The target MUST specify the "message" field within a "SetResponse" message such that the overall status of the transaction is reflected. In the case that no error occurs, the target MUST complete this field specifying the "OK (0)" canonical error code.

In the case that any operation within the "SetRequest" message fails, then (as per Section 3.4.3), the target MUST NOT apply any of the specified changes, and MUST consider the transaction as failed. The target SHOULD set the "message" field of the "SetResponse" message to an error message with the code field set to "Aborted (10)", and MUST set the "message" field of the "UpdateResult" corresponding to the failed operation to an "Error" message indicating failure. In the case that the processed operation is not the only operation within the "SetRequest" the target MUST set the "message" field of the "UpdateResult" messages for all other operations, setting the code field to "Aborted (10)".

For the operation that the target is unable to process, the "message" field MUST be set to a specific error code indicating the reason for failure based on the following mappings to canonical gRPC error codes:

- When the client has specified metadata requiring authentication (see Section 3.1), and the authentication fails - "Unauthenticated (16)".
- When the client does not have permission to modify the path specified by the operation - "PermissionDenied (7)".
- When the operation specifies a path that cannot be parsed by the target - "InvalidArgument (3)". In this case, the "message" field of the "Error" message specified SHOULD specify human-readable text indicating that the path could not be parsed.

- When the operation is an update or replace operation that corresponds to a path that is not valid - "NotFound (5)". In this case the "message" field of the "Error" message specified SHOULD specify human-readable text indicating the path that was invalid.

- When the operation is an update or replace operation that includes an invalid value within the "Update" message specified - "InvalidArgument (3)". This error SHOULD be used in cases where the payload specifies scalar values that do not correspond to the correct schema type, and in the case that multiple values are specified using a particular encoding (e.g., JSON) which cannot be decoded by the target.

3.5. Subscribing to Telemetry Updates

When a client wishes to receive updates relating to the state of data instances on a target, it creates a subscription via the "Subscribe" RPC. A subscription consists of one or more paths, with a specified subscription mode. The mode of each subscription determines the triggers for updates for data sent from the target to the client.

All requests for new subscriptions are encapsulated within a "SubscribeRequest" message - which itself has a mode which describes the longevity of the subscription. A client may create a subscription which has a dedicated stream to return one-off data ("ONCE"); a subscription that utilizes a stream to periodically request a set of data ("POLL"); or a long-lived subscription that streams data according to the triggers specified within the individual subscription’s mode ("STREAM").

The target generates messages according to the type of subscription that has been created, at the frequency requested by the client. The methods to create subscriptions are described in Section 3.5.1.

Subscriptions are created for a set of paths - which cannot be modified throughout the lifetime of the subscription. In order to cancel a subscription, the client closes the gRPC channel over which the "Subscribe" RPC was initiated, or terminates the entire gRPC session.

Subscriptions are fundamentally a set of independent update messages relating to the state of the data tree. That is, it is not possible for a client requesting a subscription to assume that the set of update messages received represent a snapshot of the data tree at a particular point in time. Subscriptions therefore allow a client to:

- Receive ongoing updates from a target which allow synchronization between the client and target for the state of elements within the

data tree. In this case (i.e., a "STREAM" subscription), a client creating a subscription receives an initial set of updates, terminated by a message indicating that initial synchronisation has completed, and then receives subsequent updates indicating changes to the initial state of those elements.

- Receive a single view (polled, or one-off) for elements of the data tree on a per-data element basis according to the state that they are in at the time that the message is transmitted. This can be more resource efficient for both target and client than a "GetRequest" for large subtrees within the data tree. The target does not need to coalesce values into a single snapshot view, or create an in-memory representation of the subtree at the time of the request, and subsequently transmit this entire view to the client.

Based on the fact that subsequent update messages are considered to be independent, and to ensure that the efficiencies described above can be achieved, by default a target MUST NOT aggregate values within an update message.

In some cases, however, elements of the data tree may be known to change together, or need to be interpreted by the subscriber together. Such data MUST be explicitly marked in the schema as being eligible to be aggregated when being published. Additionally, the subscribing client MUST explicitly request aggregation of eligible schema elements for the subscription - by means of the "allow_aggregation" flag within a "SubscriptionList" message. For elements covered by a subscription that are not explicitly marked within the schema as being eligible for aggregation the target MUST NOT coalesce these values, regardless of the value of the "allow_aggregation" flag.

When aggregation is not permitted by the client or the schema each update message MUST contain a (key, value) pair - where the key MUST be a path to a single leaf element within the data tree (encoded according to Section 2.2.2). The value MUST encode only the value of the leaf specified. In most cases, this will be a scalar value (i.e., a JSON value if a JSON encoding is utilised), but in some cases, where an individual leaf element within the schema represents an object, it MAY represent a set of values (i.e., a JSON or Protobuf object).

Where aggregation is permitted by both the client and schema, each update message MUST contain a key value pair, where the key MUST be the path to the element within the data tree which is explicitly marked as being eligible for aggregation. The value MUST be an object which encodes the children of the data tree element specified.

For JSON, the value is therefore a JSON object, and for Protobuf is a series of binary-encoded Protobuf messages.

3.5.1. Managing Subscriptions

3.5.1.1. The SubscribeRequest Message

A "SubscribeRequest" message is sent by a client to request updates from the target for a specified set of paths.

The fields of the "SubscribeRequest" are as follows:

- A group of fields, only one of which may be specified, which indicate the type of operation that the "SubscribeRequest" relates to. These are:
  - "subscribe" - a "SubscriptionList" message specifying a new set of paths that the client wishes to subscribe to.
  - "poll"- a "Poll" message used to specify (on an existing channel) that the client wishes to receive a polled update for the paths specified within the subscription. The semantics of the "Poll" message are described in Section 3.5.1.5.3.
  - "aliases" - used by a client to define (on an existing channel) a new path alias (as described in Section 2.4.2). The use of the aliases message is described in Section 3.5.1.6.

In order to create a new subscription (and its associated channel) a client MUST send a "SubscribeRequest" message, specifying the "subscribe" field. The "SubscriptionList" may create a one-off subscription, a poll-only subscription, or a streaming subscription. In the case of ONCE subscriptions, the channel between client and target MUST be closed following the initial response generation.

Subscriptions are set once, and subsequently not modified by a client. If a client wishes to subscribe to additional paths from a target, it MUST do so by sending an additional "Subscribe" RPC call, specifying a new "SubscriptionList" message. In order to end an existing subscription, a client simply closes the gRPC channel that relates to that subscription. If a channel is initiated with a "SubscribeRequest" message that does not specify a "SubscriptionList" message with the "request" field, the target MUST consider this an error. If an additional "SubscribeRequest" message specifying a "SubscriptionList" is sent via an existing channel, the target MUST respond to this message with "SubscribeResponse" message indicating an error message, with a contained error message indicating an error

code of "InvalidArgument (4)"; existing subscriptions on other gRPC channels MUST not be modified or terminated.

If a client initiates a "Subscribe" RPC with a "SubscribeRequest" message which does not contain a "SubscriptionList" message, this is an error. A "SubscribeResponse" message with the contained "error" message indicating a error code of "InvalidArgument" MUST be sent. The error text SHOULD indicate that an out-of-order operation was requested on a non-existent subscription. The target MUST subsequently close the channel.

3.5.1.2. The SubscriptionList Message

A "SubscriptionList" message is used to indicate a set of paths for which common subscription behavior are required. The fields of the message are:

- "subscription" - a set of "Subscription" messages that indicate the set of paths associated with the subscription list.
- "mode" - the type of subscription that is being created. This may be "ONCE" (described in Section 3.5.1.5.1); "STREAM" (described in Section 3.5.1.5.2); or "POLL" (described in Section 3.5.1.5.3). The default value for the mode field is "STREAM".
- "prefix"- a common prefix that is applied to all paths specified within the message as per the definition in Section 2.4.1. The default prefix is null.
- "use_aliases"- a boolean flag indicating whether the client accepts target aliases via the subscription channel. In the case that such aliases are accepted, the logic described in Section 2.4.2 is utilised. By default, path aliases created by the target are not supported.
- "qos" - a field describing the packet marking that is to be utilised for the responses to the subscription that is being created. This field has a single sub-value, "marking", which indicates the DSCP value as a 32-bit unsigned integer. If the "qos" field is not specified, the device should export telemetry traffic using its default DSCP marking for management-plane traffic.
- "allow_aggregation" - a boolean value used by the client to allow schema elements that are marked as eligible for aggregation to be combined into single telemetry update messages. By default, aggregation MUST NOT be used.

- "use_models" - a "ModelData" message (as specified in Section 2.6.1) specifying the schema definition modules that the target should use when creating a subscription. When specified, the target MUST only consider data elements within the defined set of schema models as defined in Section 2.6. When "use_models" is not specified, the target MUST consider all data elements that are defined in all schema modules that it supports.

A client generating a "SubscriptionList" message MUST include the "subscription" field - which MUST be a non-empty set of "Subscription" messages, all other fields are optional.

3.5.1.3. The Subscription Message

A "Subscription" message generically describes a set of data that is to be subscribed to by a client. It contains a "path", specified as per the definition in Section 2.2.2.

There is no requirement for the path that is specified within the message to exist within the current data tree on the server. Whilst the path within the subscription SHOULD be a valid path within the set of schema modules that the target supports, subscribing to any syntactically valid path within such modules MUST be allowed. In the case that a particular path does not (yet) exist, the target MUST NOT close the channel, and instead should continue to monitor for the existence of the path, and transmit telemetry updates should it exist in the future. The target MAY send a "SubscribeResponse" message populating the error field with "NotFound (5)" to inform the client that the path does not exist at the time of subscription creation.

For "POLL" and "STREAM" subscriptions, a client may optionally specify additional parameters within the "Subscription" message. The semantics of these additional fields are described in the relevant section of this document.

3.5.1.4. The SubscribeResponse Message

A "SubscribeResponse" message is transmitted by a target to a client over an established channel created by the "Subscribe" RPC. The message contains the following fields:

- A set of fields referred to as the "response" fields, only one of which can be specified per "SubscribeResponse" message:
  - "update" - a "Notification" message providing an update value for a subscribed data entity as described in Section 3.5.2. The "update" field is also utilised when a target wishes to

create an alias within a subscription, as described in Section 3.5.2.2.

* "sync_response" - a boolean field indicating that a particular set of data values has been transmitted, used for "POLL" and "STREAM" subscriptions.

* "error" - an "Error" message transmitted to indicate an error has occurred within a particular "Subscribe" RPC call.

3.5.1.5. Creating Subscriptions

3.5.1.5.1. ONCE Subscriptions

A subscription operating in the "ONCE" mode acts as a single request/response channel. The target creates the relevant update messages, transmits them, and subsequently closes the channel.

In order to create a one-off subscription, a client sends a "SubscribeRequest" message to the target. The "subscribe" field within this message specifies a "SubscriptionList" with the mode field set to "ONCE". Updates corresponding to the subscription are generated as per the semantics described in Section 3.5.2.

Following the transmission of all updates which correspond to data items within the set of paths specified within the subscription list, a "SubscribeResponse" message with the "sync_response" field set to "true" MUST be transmitted, and the channel over which the "SubscribeRequest" was received MUST be closed.

3.5.1.5.2. STREAM Subscriptions

Stream subscriptions are long-lived subscriptions which continue to transmit updates relating to the set of paths that are covered within the subscription indefinitely.

A "STREAM" subscription is created by sending a "SubscribeRequest" message with the subscribe field containing a "SubscriptionList" message with the type specified as "STREAM". Each entry within the "Subscription" message is specified with one of the following "modes":

o On Change ("ON_CHANGE") - when a subscription is defined to be "on change", data updates are only sent when the value of the data item changes. A heartbeat interval MAY be specified along with an "on change" subscription - in this case, the value of the data item(s) MUST be re-sent once per heartbeat interval regardless of whether the value has changed or not.

- Sampled ("SAMPLE") - a subscription that is defined to be sampled MUST be specified along with a "sample_interval" encoded as an unsigned 64-bit integer representing nanoseconds. The value of the data item(s) is sent once per sample interval to the client. If the target is unable to support the desired "sample_interval" it MUST reject the subscription by returning a "SubscribeResponse" message with the error field set to an error message indicating the "InvalidArgument (3)" error code. If the "sample_interval" is set to 0, the target MUST create the subscription and send the data with the lowest interval possible for the target.

  * Optionally, the "suppress_redundant" field of the "Subscription" message may be set for a sampled subscription. In the case that it is set to "true", the target SHOULD NOT generate a telemetry update message unless the value of the path being reported on has changed since the last update was generated. Updates MUST only be generated for those individual leaf nodes in the subscription that have changed. That is to say that for a subscription to "/a/b" - where there are leaves "c" and "d" branching from the "b" node - if the value of "c" has changed, but "d" remains unchanged, an update for "d" MUST NOT be generated, whereas an update for "c" MUST be generated.

  * A "heartbeat_interval" MAY be specified to modify the behavior of "suppress_redundant" in a sampled subscription. In this case, the target MUST generate one telemetry update per heartbeat interval, regardless of whether the "suppress_redundant" flag is set to "true". This value is specified as an unsigned 64-bit integer in nanoseconds.

- Target Defined "(TARGET_DEFINED)" - when a client creates a subscription specifying the target defined mode, the target SHOULD determine the best type of subscription to be created on a per-leaf basis. That is to say, if the path specified within the message refers to some leaves which are event driven (e.g., the changing of state of an entity based on an external trigger) then an "ON_CHANGE" subscription may be created, whereas if other data represents counter values, a "SAMPLE" subscription may be created.

3.5.1.5.3. POLL Subscriptions

Polling subscriptions are used for on-demand retrieval of statistics via long-lived channels. A poll subscription relates to a certain set of subscribed paths, and is initiated by sending a "SubscribeRequest" message with encapsulated "SubscriptionList". "Subscription" messages contained within the "SubscriptionList" indicate the set of paths that are of interest to the polling client.

To retrieve data from the target, a client sends a "SubscribeRequest" message to the target, containing a "poll" field, specified to be an empty "Poll" message. On reception of such a message, the target MUST generate updates for all the corresponding paths within the "SubscriptionList". Updates MUST be generated according to Section 3.5.2.

3.5.1.6. Client-defined Aliases within a Subscription

When a client wishes to create an alias that a target should use for a path, the client should send a "SubscribeRequest" message specifying the "aliases" field. The "aliases" field consists of an "AliasList" message. An "AliasList" specifies a list of aliases, each of which consists of:

- "path" - the target path for the alias - encoded as per Section 2.2.2.
- "alias" - the (client-defined) alias for the path, encoded as per Section 2.4.2.

Where a target is unable to support a client-defined alias it SHOULD respond with a "SubscribeResponse" message with the error field indicating an error of the following types:

- "InvalidArgument (3)" where the specified alias is not acceptable to the target.
- "AlreadyExists (6)" where the alias defined is a duplicate of an existing alias for the client.
- "ResourceExhausted (8)" where the target has insufficient memory or processing resources to support the alias.
- "Unknown (2)" in all other cases.

Thus, for a client to create an alias corresponding to the path"/a/b/c/d[id=10]/e" with the name "shortPath", it sends a "SubscribeRequest" message with the following fields specified:

```
subscriberequest: <
  aliases: <
    alias: <
      path: <
        element: "a"
        element: "b"
        element: "c"
        element: "d[id=10]"
        element: "e"
      >
      alias: "#shortPath"
    >
  >
>
```

If the alias is acceptable to the target, subsequent updates are transmitted using the "#shortPath" alias in the same manner as described in Section 3.5.2.2.

3.5.2. Sending Telemetry Updates

3.5.2.1. Bundling of Telemetry Updates

Since multiple "Notification" messages can be included in the update field of a "SubscribeResponse" message, it is possible for a target to bundle messages such that fewer messages are sent to the client. The advantage of such bundling is clearly to reduce the number of bytes on the wire (caused by message overhead). Since "Notification" messages contain the timestamp at which an event occurred, or a sample was taken, such bundling does not affect the sample accuracy to the client. However, bundling does have a negative impact on the freshness of the data in the client - and on the client’s ability to react to events on the target.

Since it is not possible for the target to infer whether its clients are sensitive to the latency introduced by bundling, if a target implements optimizations such that multiple "Notification" messages are bundled together, it MUST provide an ability to disable this functionality within the configuration of the gNMI service. Additionally, a target SHOULD provide means by which the operator can control the maximum number of updates that are to be bundled into a single message, This configuration is expected to be implemented out-of-band to the gNMI protocol itself.

3.5.2.2. Target-defined Aliases within a Subscription

Where the "use_aliases" field of a "SubscriptionList" message has been set to "true", a target MAY create aliases for paths within a subscription. A target-defined alias MUST be created separately from an update to the corresponding data item(s).

To create a target-defined alias, a "SubscribeResponse" message is generated with the "update" field set to a "Notification" message. The "Notification" message specifies the aliased path within the "prefix" field, and a non-null "alias" field, specified according to Section 2.4.2.

Thus, a target wishing to create an alias relating to the path "/a/b/c[id=10]" and subsequently update children of the "c[id=10]"entity must:

1. Generate a "SubscribeResponse" message and transmit it over the channel to the client:

```
subscriberesponse: <
  update: <
    timestamp: (timestamp)
    prefix: <
      element: "a"
      element: "b"
      element: "c[id=10]"
    >
    alias: "#42"
  >
>
```

1. Subsequently, this alias can be used to provide updates for the "child1" leaf corresponding to "/a/b/c[id=10]/child1":

```
subscriberesponse: <
  update: <
    timestamp: (timestamp)
    prefix: <
      element: "#42"
    >
    update: <
      path: <
        element: "child1"
      >
      value: <
        value: 102                      // integer representation
        type: JSON_IETF
      >
    >
  >
>
```

3.5.2.3. Sending Telemetry Updates

When an update for a subscribed telemetry path is to be sent, a "SubscribeResponse" message is sent from the target to the client, on the channel associated with the subscription. The "update" field of the message contains a "Notification" message as per the description in Section 2.1. The "timestamp" field of the "Notification" message MUST be set to the time at which the value of the path that is being updated was collected.

Where a leaf node's value has changed, or a new node has been created, an "Update" message specifying the path and value for the updated data item MUST be appended to the "update" field of the message.

Where a node within the subscribed paths has been removed, the "delete" field of the "Notification" message MUST have the path of the node that has been removed appended to it.

When the target has transmitted the initial updates for all paths specified within the subscription, a "SubscribeResponse" message with the "sync_response" field set to "true" MUST be transmitted to the client to indicate that the initial transmission of updates has concluded. This provides an indication to the client that all of the existing data for the subscription has been sent at least once. For "STREAM" subscriptions, such messages are not required for subsequent updates. For "POLL" subscriptions, after each set of updates for individual poll request, a "SubscribeResponse" message with the "sync_response" field set to "true" MUST be generated.

4. References

4.1. URIs

[1] http://grpc.io

[2] https://developers.google.com/protocol-buffers/

[3] https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi.proto

[4] https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-specification.md

[5] http://www.openconfig.net/

[6] https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-path-conventions.md

[7] https://tools.ietf.org/html/rfc7159

[8] https://tools.ietf.org/html/rfc7159

[9] https://tools.ietf.org/html/rfc7951

[10] http://www.grpc.io/grpc-java/javadoc/index.html

[11] https://godoc.org/google.golang.org/grpc/codes#Code

[12] http://www.grpc.io/grpc/cpp/classgrpc_1_1_status.html

[13] https://datatracker.ietf.org/doc/draft-openconfig-netmod-model-catalog/

[14] https://tools.ietf.org/html/draft-openconfig-netmod-model-catalog-01

[15] https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-authentication.md

[16] http://www.openconfig.net/documentation/semantic-versioning/

[17] https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi.proto

Appendix A. Appendix: Current Protobuf Message and Service Specification

The latest Protobuf IDL gNMI specification is found at [17].

Appendix B. Appendix: Current Outstanding Issues/Future Features

- Ability for the client to exclude paths from a subscription or get.
- "Dial out" for the target to register with an NMS and publish pre-configured subscriptions.

Authors' Addresses

Rob Shakir
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043

Email: robjs@google.com

Anees Shaikh
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: aashaikh@google.com

Paul Borman
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: borman@google.com

Marcus Hines
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: hines@google.com

Carl Lebsack
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: csl@google.com

Chris Morrow
Google

Email: christopher.morrow@gmail.com

RIFT Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 15, 2018

T. Przygienda, Ed.
Juniper Networks
A. Sharma
Comcast
A. Atlas
J. Drake
Juniper Networks
January 11, 2018

RIFT: Routing in Fat Trees
draft-przygienda-rift-04

Abstract

This document outlines a specialized, dynamic routing protocol for Clos and fat-tree network topologies. The protocol (1) deals with automatic construction of fat-tree topologies based on detection of links, (2) minimizes the amount of routing state held at each level, (3) automatically prunes the topology distribution exchanges to a sufficient subset of links, (4) supports automatic disaggregation of prefixes on link and node failures to prevent black-holing and suboptimal routing, (5) allows traffic steering and re-routing policies, (6) allows non-ECMP forwarding and ultimately (7) provides mechanisms to synchronize a limited key-value data-store that can be used after protocol convergence to e.g. bootstrap higher levels of functionality on nodes.

Status of This Memo

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

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

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

This Internet-Draft will expire on July 15, 2018.

Table of Contents

1. Introduction 4
1.1. Requirements Language 4
2. Reference Frame 4
2.1. Terminology 4
2.2. Topology 7
3. Requirement Considerations 8
4. RIFT: Routing in Fat Trees 10
4.1. Overview 10
4.2. Specification 10
4.2.1. Transport 10
4.2.2. Link (Neighbor) Discovery (LIE Exchange) 11
4.2.3. Topology Exchange (TIE Exchange) 12
4.2.3.1. Topology Information Elements 12
4.2.3.2. South- and Northbound Representation 12
4.2.3.3. Flooding 15
4.2.3.4. TIE Flooding Scopes 15
4.2.3.5. Initial and Periodic Database Synchronization . . 17
4.2.3.6. Purging 17
4.2.3.7. Southbound Default Route Origination 17
4.2.3.8. Optional Automatic Flooding Reduction and Partitioning 18
4.2.4. Policy-Guided Prefixes 19
4.2.4.1. Ingress Filtering 20
4.2.4.2. Applying Policy 21
4.2.4.3. Store Policy-Guided Prefix for Route Computation and Regeneration 22
4.2.4.4. Re-origination 22
4.2.4.5. Overlap with Disaggregated Prefixes 22
4.2.5. Reachability Computation 23
4.2.5.1. Northbound SPF 23
4.2.5.2. Southbound SPF 24

4.2.5.3. East-West Forwarding Within a Level 24
4.2.6. Attaching Prefixes 24
4.2.7. Attaching Policy-Guided Prefixes 26
4.2.8. Automatic Disaggregation on Link & Node Failures . . 26
4.2.9. Optional Autoconfiguration 30
4.2.9.1. Terminology 30
4.2.9.2. Automatic SystemID Selection 31
4.2.9.3. Generic Fabric Example 31
4.2.9.4. Level Determination Procedure 32
4.2.9.5. Resulting Topologies 33
4.2.10. Stability Considerations 35
4.3. Further Mechanisms 36
4.3.1. Overload Bit 36
4.3.2. Optimized Route Computation on Leafs 36
4.3.3. Key/Value Store 36
4.3.4. Interactions with BFD 37
4.3.5. Leaf to Leaf Procedures 37
4.3.6. Other End-to-End Services 38
4.3.7. Address Family and Multi Topology Considerations . . 38
4.3.8. Reachability of Internal Nodes in the Fabric 38
4.3.9. One-Hop Healing of Levels with East-West Links . . . 38
5. Examples 39
5.1. Normal Operation 39
5.2. Leaf Link Failure 40
5.3. Partitioned Fabric 41
5.4. Northbound Partitioned Router and Optional East-West Links 43
6. Implementation and Operation: Further Details 44
6.1. Considerations for Leaf-Only Implementation 44
6.2. Adaptations to Other Proposed Data Center Topologies . . 44
6.3. Originating Non-Default Route Southbound 45
7. Security Considerations 45
8. Information Elements Schema 46
8.1. common.thrift 46
8.2. encoding.thrift 50
9. IANA Considerations 55
10. Acknowledgments 55
11. References 56
11.1. Normative References 56
11.2. Informative References 57
Authors' Addresses 58

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 without southbound adjacencies. Its level is 0 (except cases where it is deriving its level via ZTP and is running without LEAF_ONLY which will be explained in Section 4.2.9).

Connected Spine: In case a spine level represents a connected graph (discounting links terminating at different levels), we call it a "connected spine", in case a spine level consists of multiple partitions, we call it a "disconnected" or "partitioned spine". In other terms, a spine without east-west links is disconnected and is the typical configuration forf 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

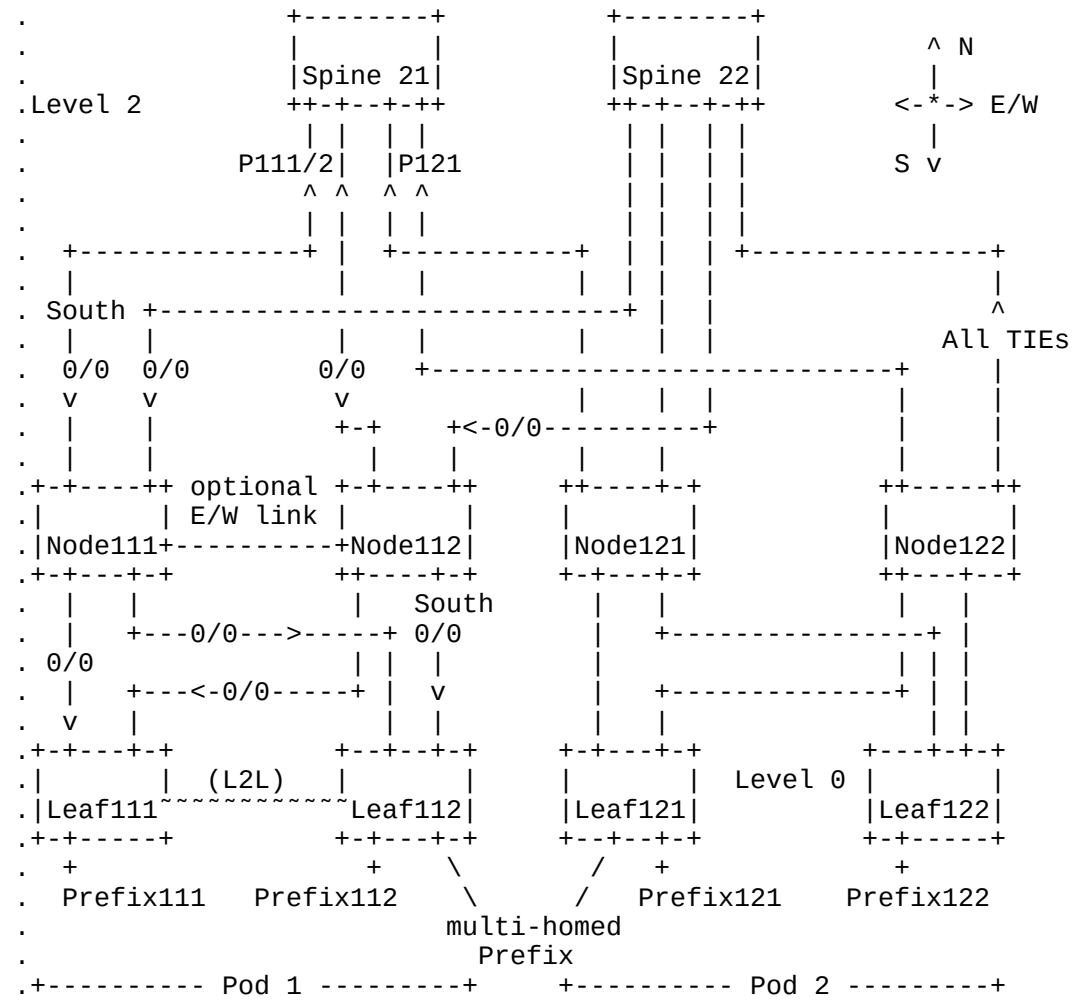


Figure 1: A two level spine-and-leaf topology

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 control protocol should discover the physical links automatically and be able to detect cabling that violates fat-tree topology constraints. It must react accordingly to such mis-cabling attempts, at a minimum preventing adjacencies between nodes from being formed and traffic from being forwarded on those mis-cabled links. E.g. connecting a leaf to a spine at level 2 should be detected and ideally prevented.

REQ2: A node without any configuration beside default values should come up at the correct level in any PoD it is introduced into. Optionally, it must be possible to configure nodes to restrict their participation to the PoD(s) targeted at any level.

REQ3: Optionally, the protocol should allow to provision data centers where the individual switches carry no configuration information and are all deriving their level from a "seed". Observe that this requirement may collide with the desire to detect cabling misconfiguration and with that only one of the requirements can be fully met in a chosen configuration mode.

REQ4: The solution should allow for minimum size routing information base and forwarding tables at leaf level for speed, cost and simplicity reasons. Holding excessive amount of information away from leaf nodes simplifies operation and lowers cost of the underlay.

REQ5: Very high degree of ECMP must be supported. Maximum ECMP is currently understood as the most efficient routing approach to maximize the throughput of switching fabrics [MAKSIC2013].

REQ6: Non equal cost anycast must be supported to allow for easy and robust multi-homing of services without regressing to careful balancing of link costs.

REQ7: Traffic engineering should be allowed by modification of prefixes and/or their next-hops.

REQ8: The solution should allow for access to link states of the whole topology to enable efficient support for modern

control architectures like SPRING [RFC7855] or PCE [RFC4655].

REQ9: The solution should easily accommodate opaque data to be carried throughout the topology to subsets of nodes. This can be used for many purposes, one of them being a key-value store that allows bootstrapping of nodes based right at the time of topology discovery.

REQ10: Nodes should be taken out and introduced into production with minimum wait-times and minimum of "shaking" of the network, i.e. radius of propagation (often called "blast radius") of changed information should be as small as feasible.

REQ11: The protocol should allow for maximum aggregation of carried routing information while at the same time automatically de-aggregating the prefixes to prevent black-holing in case of failures. The de-aggregation should support maximum possible ECMP/N-ECMP remaining after failure.

REQ12: Reducing the scope of communication needed throughout the network on link and state failure, as well as reducing advertisements of repeating, idiomatic or policy-guided information in stable state is highly desirable since it leads to better stability and faster convergence behavior.

REQ13: Once a packet traverses a link in a "southbound" direction, it must not take any further "northbound" steps along its path to delivery to its destination under normal conditions. Taking a path through the spine in cases where a shorter path is available is highly undesirable.

REQ14: Parallel links between same set of nodes must be distinguishable for SPF, failure and traffic engineering purposes.

REQ15: The protocol must not rely on interfaces having discernible unique addresses, i.e. it must operate in presence of unnumbered links (even parallel ones) or links of a single node having same addresses.

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.

Future versions may include a [PROTOBUF] schema.

4.2.2. Link (Neighbor) Discovery (LIE Exchange)

LIE exchange happens over well-known administratively locally scoped IPv4 multicast address [RFC2365] or link-local multicast scope for IPv6 [RFC4291] and SHOULD be sent with a TTL of 1 to prevent RIFT information reaching beyond a single L3 next-hop in the topology. LIEs are exchanged over all links running RIFT.

Unless Section 4.2.9 is used, each node is provisioned with the level at which it is operating and its PoD or otherwise a default level and 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 (definitions of LEAF_ONLY are found in Section 4.2.9)

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

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

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

4. the advertised MTUs match on both sides AND

5. both nodes advertise defined level values AND

6. [

   i) the node is at level 0 and it has no adjacencies to nodes with level higher than this neighboring node OR

   ii) both nodes are at level 0 AND both indicate support for Section 4.3.5 OR

   iii) neither node is at level 0 and the neighboring node is at most one level away

   ].

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.

A node SHOULD NOT send out LIEs without defined level in the header but in certain scenarios it may be beneficial for trouble-shooting purposes.

LIE exchange uses three-way handshake mechanism [RFC5303]. Precise finite state machines will be provided in later versions of this specification. LIE packets contain nonces and may contain an SHA-1 [RFC6234] over nonces and some of the LIE data which prevents corruption and replay attacks. TIE flooding reuses those nonces to prevent mismatches and can use those for security purposes in case it is using QUIC [QUIC]. Section 7 will address the precise security mechanisms in the future.

4.2.3. Topology Exchange (TIE Exchange)

4.2.3.1. Topology Information Elements

Topology and reachability information in RIFT is conveyed by the means of TIEs which have good amount of commonalities with LSAs in OSPF.

TIE exchange mechanism uses port indicated by each node in the LIE exchange and the interface on which the adjacency has been formed as destination. It SHOULD use TTL of 1 as well.

TIEs contain sequence numbers, lifetimes and a type. Each type has a large identifying number space and information is spread across possibly many TIEs of a certain type by the means of a hash function that a node or deployment can individually determine. One extreme point of the design space is a prefix per TIE which leads to BGP-like behavior vs. dense packing into few TIEs leading to more traditional IGP trade-off with fewer TIEs. An implementation may even rehash at the cost of significant amount of re-advertisements of TIEs.

More information about the TIE structure can be found in the schema in Section 8.

4.2.3.2. South- and Northbound Representation

As a central concept to RIFT, each node represents itself differently depending on the direction in which it is advertising information. More precisely, a spine node represents two different databases to its neighbors depending whether it advertises TIEs to the north or to the south/sideways. We call those differing TIE databases either south- or northbound (S-TIEs and N-TIEs) depending on the direction of distribution.

The N-TIEs hold all of the node's adjacencies, local prefixes and northbound policy-guided prefixes while the S-TIEs hold only all of the node's adjacencies and the default prefix with necessary disaggregated prefixes and southbound policy-guided prefixes. We will explain this in detail further in Section 4.2.8 and Section 4.2.4.

The TIE types are symmetric in both directions and Table 1 provides a quick reference to the different TIE types including direction and their function.

TIE-Type	Content
node N-TIE	node properties, adjacencies and information helping in complex disaggregation scenarios
node S-TIE	same content as node N-TIE except the information to help disaggregation
Prefix N-TIE	contains nodes' directly reachable prefixes
Prefix S-TIE	contains originated defaults and de-aggregated prefixes
PGP N-TIE	contains nodes north PGPs
PGP S-TIE	contains nodes south PGPs
KV N-TIE	contains nodes northbound KVs
KV S-TIE	contains nodes southbound KVs

Table 1: TIE Types

As an example illustrating a databases holding both representations, consider the topology in Figure 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-TIEs:
Node S-TIE:
  NodeElement(layer=2, neighbors((Node111, layer 1, cost 1),
  (Node112, layer 1, cost 1), (Node121, layer 1, cost 1),
  (Node122, layer 1, cost 1)))
Prefix S-TIE:
  SouthPrefixesElement(prefixes(0/0, cost 1), (::/0, cost 1))

Node111 S-TIEs:
Node S-TIE:
  NodeElement(layer=1, neighbors((Spine21, layer 2, cost 1, links(...)),
```

```
          (Spine22, layer 2, cost 1, links(...)),
          (Node112, layer 1, cost 1, links(...)),
          (Leaf111, layer 0, cost 1, links(...)),
          (Leaf112, layer 0, cost 1, links(...))))
        Prefix S-TIE:
          SouthPrefixesElement(prefixes(0/0, cost 1), (::/0, cost 1))

        Node111 N-TIEs:
        Node N-TIE:
          NodeElement(layer=1,
          neighbors((Spine21, layer 2, cost 1, links(...)),
          (Spine22, layer 2, cost 1, links(...)),
          (Node112, layer 1, cost 1, links(...)),
          (Leaf111, layer 0, cost 1, links(...)),
          (Leaf112, layer 0, cost 1, links(...))))
        Prefix N-TIE:
          NorthPrefixesElement(prefixes(Node111.loopback)

        Node121 S-TIEs:
        Node S-TIE:
          NodeElement(layer=1, neighbors((Spine21,layer 2,cost 1),
          (Spine22, layer 2, cost 1), (Leaf121, layer 0, cost 1),
          (Leaf122, layer 0, cost 1)))
        Prefix S-TIE:
          SouthPrefixesElement(prefixes(0/0, cost 1), (::/0, cost 1))

        Node121 N-TIEs:
        Node N-TIE:
          NodeLinkElement(layer=1,
          neighbors((Spine21, layer 2, cost 1, links(...)),
          (Spine22, layer 2, cost 1, links(...)),
          (Leaf121, layer 0, cost 1, links(...)),
          (Leaf122, layer 0, cost 1, links(...))))
        Prefix N-TIE:
          NorthPrefixesElement(prefixes(Node121.loopback)

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

Figure 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.

On reception of a TIE with an undefined level value in the packet header the node SHOULD issue a warning and indiscriminately discard the packet.

Precise finite state machines and procedures will be provided in later versions of this specification.

4.2.3.4. TIE Flooding Scopes

In a somewhat analogous fashion to link-local, area and domain flooding scopes, RIFT defines several complex "flooding scopes" depending on the direction and type of TIE propagated.

Every N-TIE is flooded northbound, providing a node at a given level with the complete topology of the Clos or Fat Tree network underneath it, including all specific prefixes. This means that a packet received from a node at the same or lower level whose destination is covered by one of those specific prefixes may be routed directly towards the node advertising that prefix rather than sending the packet to a node at a higher level.

A node's node S-TIEs, consisting of all node's adjacencies and prefix S-TIEs with default IP prefix and disaggregated prefixes, are flooded southbound in order to allow the nodes one level down to see connectivity of the higher level as well as reachability to the rest of the fabric. In order to allow a E-W disconnected node in a given level to receive the S-TIEs of other nodes at its level, every *NODE* S-TIE is "reflected" northbound to level from which it was received. It should be noted that east-west links are included in South TIE flooding; those TIEs need to be flooded to satisfy algorithms in Section 4.2.5. In that way nodes at same level can learn about each other without a lower level, e.g. in case of leaf level. The precise flooding scopes are given in Table 2. Those rules govern as well what SHOULD be included in TIDEs towards neighbors. East-West flooding scopes are identical to South flooding scopes.

Node S-TIE "reflection" allows to support disaggregation on failures describes in Section 4.2.8 and flooding reduction in Section 4.2.3.8.

Packet Type vs. Peer Direction	South	North
node S-TIE	flood self-originated only	flood if TIE originator's level is higher than own level
non-node S-TIE	flood self-originated only	flood only if TIE originator is equal peer
all N-TIEs	never flood	flood always
TIDE	include TIEs in flooding scope	include TIEs in flooding scope
TIRE	include all N-TIEs and all peer's self-originated TIEs and all node S-TIEs	include only if TIE originator is equal peer

Table 2: Flooding Scopes

As an example to illustrate these rules, consider using the topology in Figure 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 3.

Router floods to	Neighbor	TIEs
Leaf111	Node112	Leaf111 N-TIEs, Node111 node S-TIE
Leaf111	Node111	Leaf111 N-TIEs, Node112 node S-TIE
Node111	Leaf111	Node111 S-TIEs
Node111	Leaf112	Node111 S-TIEs
Node111	Node112	Node111 S-TIEs
Node111	Spine21	Node111 N-TIEs, Leaf111 N-TIEs, Leaf112 N-TIEs, Spine22 node S-TIE
Node111	Spine22	Node111 N-TIEs, Leaf111 N-TIEs, Leaf112 N-TIEs, Spine21 node S-TIE
...	...	...
Spine21	Node111	Spine21 S-TIEs
Spine21	Node112	Spine21 S-TIEs
Spine21	Node121	Spine21 S-TIEs
Spine21	Node122	Spine21 S-TIEs
...	...	...

Table 3: Flooding some TIEs from example topology

4.2.3.5. Initial and Periodic Database Synchronization

The initial exchange of RIFT is modeled after ISIS with TIDE being equivalent to CSNP and TIRE playing the role of PSNP. The content of TIDEs and TIREs is governed by Table 2.

4.2.3.6. Purging

RIFT does not purge information that has been distributed by the protocol. Purging mechanisms in other routing protocols have proven to be complex and fragile over many years of experience. Abundant amounts of memory are available today even on low-end platforms. The information will age out and all computations will deliver correct results if a node leaves the network due to the new information distributed by its adjacent nodes.

Once a RIFT node issues a TIE with an ID, it MUST preserve the ID as long as feasible (also when the protocol restarts), even if the TIE 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

1. is NOT overloaded AND

2. has southbound or east-west adjacencies

originates in its south prefix TIE such a default route IIF

1. all other nodes at X’s’ level are overloaded OR

2. all other nodes at X’s’ level have NO northbound adjacencies OR

3. X has computed reachability to a default route during N-SPF.

The term "all other nodes at X’s’ level" describes obviously just the nodes at the same level in the POD with a viable lower layer

(otherwise the node S-TIEs cannot be reflected and the nodes in e.g. POD 1 nad POD 2 are "invisible" to each other).

A node originating a southbound default route MUST install a default discard route if it did not compute a default route during N-SPF.

4.2.3.8. Optional Automatic Flooding Reduction and Partitioning

Several nodes can, but strictly only under conditions defined below, run a hashing function based on TIE originator value and partition flooding between them.

Steps for flooding reduction and partitioning:

1. select all nodes in the same level for which

    A. node S-TIEs have been received AND

    B. which have precisely the same non-empty sets of respectively north and south neighbor adjacencies AND

    C. have at least one shared southern neighbor including backlink verification and

    D. support flooding reduction (overload bits are ignored)

    and then

2. run on the chosen set a hash algorithm using nodes flood priorities and IDs to select flooding leader and backup per TIE originator ID, i.e. each node floods immediately through to all its necessary neighbors TIEs that it received with an originator ID that makes it the flooding leader or backup for this originator. The preference (higher is better) is computed as 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 and east-west adjacencies in North Node TIEs when progressing Dijkstra. Observe that this is really just a one hop variety since South Node TIEs are not re-flooded southbound beyond a single level (or east-west) and with that the computation cannot progress beyond adjacent nodes.

Default route found when crossing an E-W link is used IIF

1. the node itself does NOT have any northbound adjacencies AND

2. the adjacent node has one or more northbound adjacencies

This rule forms a "one-hop default route split-horizon" and prevents looping over default routes while allowing for "one-hop protection" of nodes that lost all northbound adjacencies.

Other south prefixes found when crossing E-W link MAY be used IIF

1. no north neighbors are advertising same or supersuming prefix AND

2. the node does not originate a supersuming prefix itself.

i.e. the E-W link can be used as the gateway of last resort for a specific prefix only. Using south prefixes across E-W link can be beneficial e.g. on automatic de-aggregation in pathological fabric partitioning scenarios.

A detailed example can be found in Section 5.4.

For N-SPF we are using the South Node TIEs to find according adjacencies to verify backlink connectivity. Just as in case of IS-IS or OSPF, two unidirectional links are associated together to confirm bidirectional connectivity.

4.2.5.2. Southbound SPF

S-SPF uses only the southbound adjacencies in the south node TIEs, i.e. progresses towards nodes at lower levels. Observe that E-W adjacencies are NEVER used in the computation. This enforces the requirement that a packet traversing in a southbound direction must never change its direction.

S-SPF uses northbound adjacencies in north node TIEs to verify backlink connectivity.

4.2.5.3. East-West Forwarding Within a Level

Ultimately, it should be observed that in presence of a "ring" of E-W links in a level neither SPF will provide a "ring protection" scheme since such a computation would have to deal necessarily with breaking of "loops" in generic Dijkstra sense; an application for which RIFT is not intended. It is outside the scope of this document how an underlay can be used to provide a full-mesh connectivity between nodes in the same layer that would allow for N-SPF to provide protection for a single node loosing all its northbound adjacencies (as long as any of the other nodes in the level are northbound connected).

Using south prefixes over horizontal links is optional and can protect against pathological fabric partitioning cases that leave only paths to destinations that would necessitate multiple changes of forwarding direction between north and south.

4.2.6. Attaching Prefixes

After the SPF is run, it is necessary to attach according prefixes. For S-SPF, prefixes from an N-TIE are attached to the originating node with that node’s next-hop set and a distance equal to the prefix’s cost plus the node’s minimized path distance. The RIFT route database, a set of (prefix, type=spf, path_distance, next-hop set), accumulates these results. Obviously, the prefix retains its type which is used to tie-break between the same prefix advertised with different types.

In case of N-SPF prefixes from each S-TIE need to also be added to the RIFT route database. The N-SPF is really just a stub so the computing node needs simply to determine, for each prefix in an S-TIE that originated from adjacent node, what next-hops to use to reach that node. Since there may be parallel links, the next-hops to use can be a set; presence of the computing node in the associated Node S-TIE is sufficient to verify that at least one link has bidirectional connectivity. The set of minimum cost next-hops from the computing node X to the originating adjacent node is determined.

Each prefix has its cost adjusted before being added into the RIFT route database. The cost of the prefix is set to the cost received plus the cost of the minimum cost next-hop to that neighbor. Then each prefix can be added into the RIFT route database with the next_hop_set; ties are broken based upon type first and then distance. RIFT route preferences are normalized by the according thrift model type.

An exemplary implementation for node X follows:

```
for each S-TIE
   if S-TIE.layer > X.layer
      next_hop_set = set of minimum cost links to the S-TIE.originator
      next_hop_cost = minimum cost link to S-TIE.originator
      end if
   for each prefix P in the S-TIE
      P.cost = P.cost + next_hop_cost
      if P not in route_database:
        add (P, type=DistVector, P.cost, next_hop_set) to route_database
        end if
      if (P in route_database) and
           (route_database[P].type is not PolicyGuided):
        if route_database[P].cost > P.cost):
          update route_database[P] with (P, DistVector, P.cost, next_hop_set)
        else if route_database[P].cost == P.cost
          update route_database[P] with (P, DistVector, P.cost,
             merge(next_hop_set, route_database[P].next_hop_set))
        else
          // Not preferred route so ignore
          end if
        end if
      end for
   end for
```

Figure 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.

4.2.9. Optional Autoconfiguration

Each RIFT node can optionally operate in zero touch provisioning (ZTP) mode, i.e. a node will fully configure itself after being attached to the topology. This section describes the necessary concepts and procedures.

4.2.9.1. Terminology

Automatic Level Derivation: Procedures which allow nodes without level configured to derive it automatically. Only applied if CONFIGURED_LEVEL is undefined.

UNDEFINED_LEVEL: An imaginary value that indicates that the level has not beeen determined and has not been configured.

LEAF_ONLY: An optional configuration flag that can be configured on a node to make sure it never leaves the "bottom of the hierarchy". SUPERSPINE_FLAG and CONFIGURED_LEVEL cannot be defined at the same time as this flag. It implies CONFIGURED_LEVEL value of 0.

CONFIGURED_LEVEL: A level value provided manually. When this is defined (i.e. not UNDEFINED_LEVEL) a node is not participating in ZTP. SUPERSPINE_FLAG MUST NOT be set when this value is defined. LEAF_ONLY can be set only if this value is undefined or set to 0.

DERIVED_LEVEL: Level value computed via automatic level derivation when CONFIGURED_LEVEL has not been set to something else than UNDEFINED_LEVEL.

LEAF_2_LEAF: An optional configuration flag that can be configured on a node to make sure it supports procedures defined in Section 4.3.5. SUPERSPINE_FLAG cannot be defined at the same time

as this flag. It implies LEAF_ONLY and the according restrictions.

LEVEL_VALUE: In ZTP case the original definition of "level" in Section 2.1 is both extended and relaxed. First, level is defined now as LEVEL_VALUE and is the first defined value of CONFIGURED_LEVEL followed by DERIVED_LEVEL. Second, it is possible for nodes to be more than one level apart to form adjacencies if any of the nodes is at least LEAF_ONLY.

Valid Level Offer: A neighbor’s level received on a valid LIE (i.e. passing all checks for adjacency formation while disregarding all clauses involving level values) within the holdtime interval last received LIE advertises.

Highest Available Level (HAL): Highest defined level value seen from all valid level offers received.

SUPERSPINE_FLAG: Configuration flag provided to all superspines. LEAF_FLAG and CONFIGURED_LEVEL cannot be defined at the same time as this flag. It implies CONFIGURED_LEVEL value of 64. In fact, it is basically a shortcut for configuring same level at all superspine nodes which is unavoidable since an initial ’seed’ is needed for other ZTP nodes to derive their level in the topology.

4.2.9.2. Automatic SystemID Selection

RIFT identifies each node via a SystemID which is 64 bits wide. It is relatively simple to derive a, for all practical purposes collision free, value for each node on startup. As simple examples either system MAC + 2 random bytes can be used or an IPv4/IPv6 router ID interface address. The router MUST ensure that such identifier is not changing very frequently (at least not without sending all its TIEs with fairly short lifetime) since otherwise the network may be left with large amounts of stale TIEs in other nodes albeit this is not necessarily a serious problem if the procedures suggested in Section 7 are implemented.

4.2.9.3. Generic Fabric Example

ZTP forces us to think about miscabled or unusually cabled fabric and how such a topology can be forced into a "lattice" structure which a fabric represents (with further restrictions). Let us consider a necessary and sufficient physical cabling in Figure 6. We assume all nodes being in the same PoD.

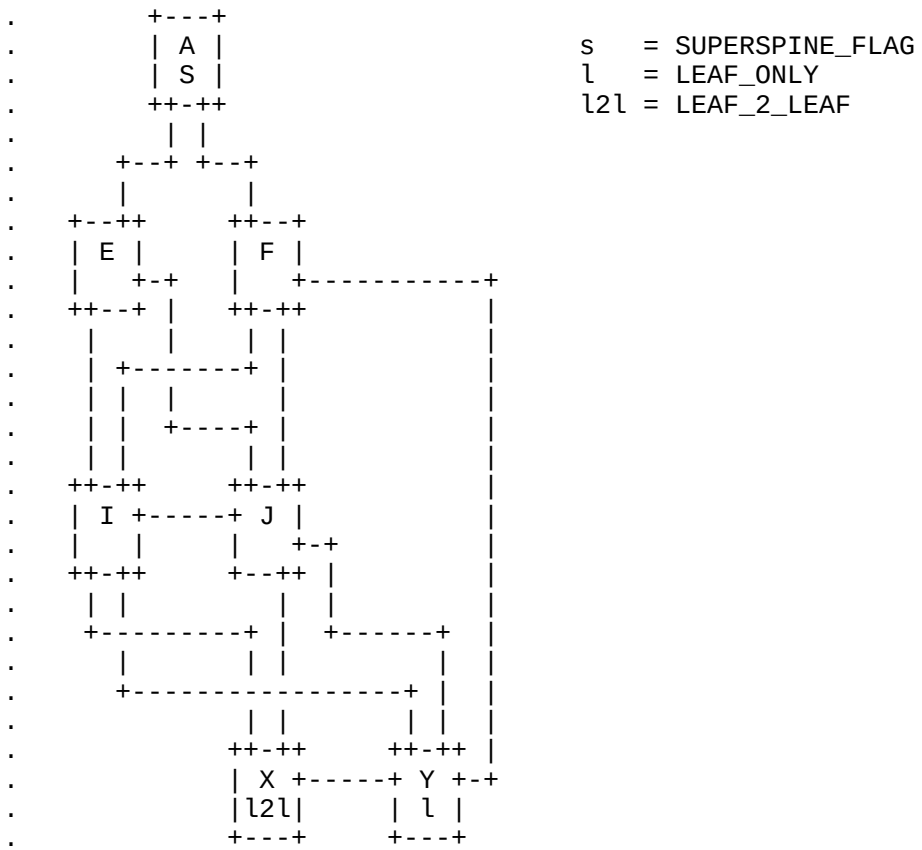


Figure 6: Generic ZTP Cabling Considerations

First, we need to anchor the "top" of the cabling and that’s what the SUPERSPINE_FLAG at node A is for. Then things look smooth until we have to decide whether node Y is at the same level as I, J or at the same level as Y and consequently, X is south of it. This is unresolvable here until we "nail down the bottom" of the topology which the leaf flags are for. We will see further then whether Y chooses to form adjacencies to F or I, J consecutively.

4.2.9.4. Level Determination Procedure

A node starting up with UNDEFINED_VALUE (i.e. without a CONFIGURED_LEVEL or any leaf or superspine flag) MUST follow those additional procedures:

1. It advertises its LEVEL_VALUE on all LIEs (observe that this can be UNDEFINED_LEVEL which in terms of the schema is simply an omitted optional value).

2. It chooses on an ongoing basis from all received valid level offers the value of MAX(HAL-1,0) as its DERIVED_LEVEL. The node then starts to advertise this derived level.

3. A node that lost all adjacencies with HAL value MUST revert to UNDEFINED_VALUE as its level and hold down computation of new DERIVED_LEVEL for a short period of time.

4. A node MUST reset any adjacency that has changed the level it is offering and is in three way state.

5. A node that changed its defined level value MUST readvertise its own TIEs (since the new ‘PacketHeader‘ will contain a different level than before). Sequence number of each TIE MUST be increased.

A node starting with LEVEL_VALUE being 0 (i.e. it assumes a leaf function or has a CONFIGURED_LEVEL of 0) MUST follow those additional procedures:

1. It computes HAL per procedures above but does NOT use it to compute DERIVED_LEVEL. HAL is used to limit adjacency formation per Section 4.2.2.

Precise finite state machines will be provided in later versions of this specification.

4.2.9.5. Resulting Topologies

The procedures defined in Section 4.2.9.4 will lead to the RIFT topology and levels depicted in Figure 7.

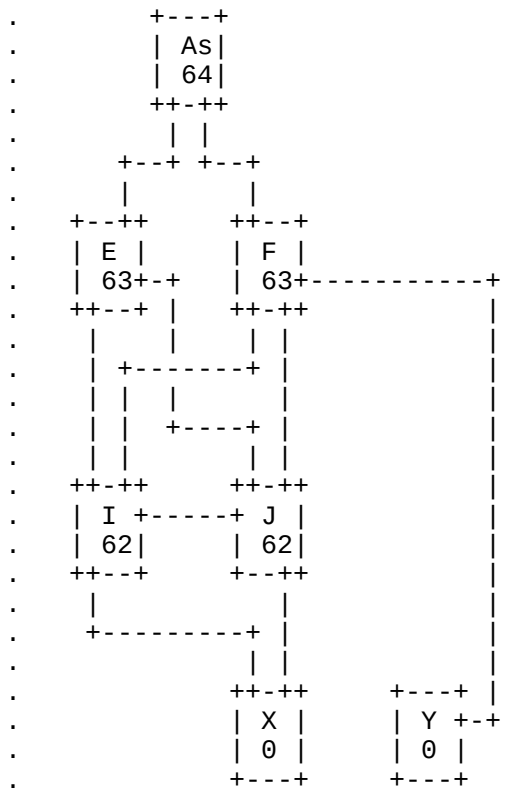


Figure 7: Generic ZTP Topology Autoconfigured

In case we imagine the LEAF_ONLY restriction on Y is removed the outcome would be very different however and result in Figure 8. This demonstrates basically that auto configuration prevents miscabling detection and with that can lead to undesirable effects when leafs are not "nailed" and arbitrarily cabled.

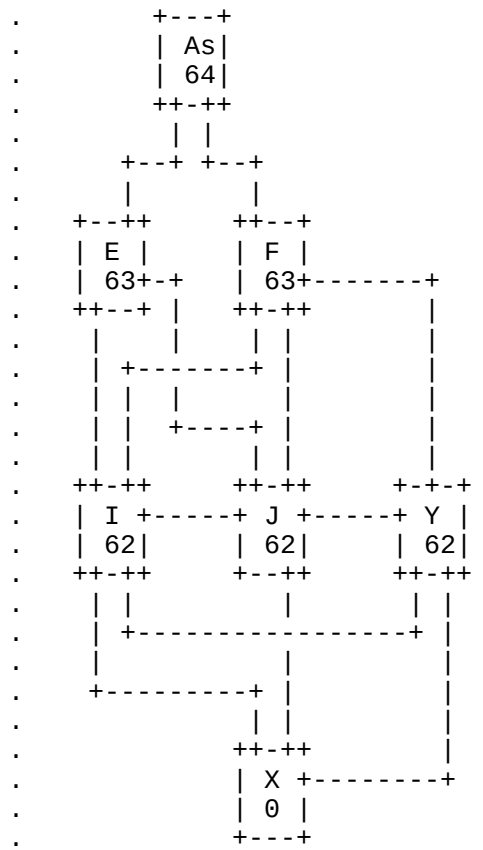


Figure 8: Generic ZTP Topology Autoconfigured

4.2.10. Stability Considerations

The autoconfiguration mechanism computes a global maximum of levels by diffusion. The achieved equilibrium can be disturbed massively by all nodes with highest level either leaving or entering the domain (with some finer distinctions not explained further). It is therefore recommended that each node is multi-homed towards nodes with respective HAL offerings. Fortuntately, this is the natural state of things for the topology variants considered in RIFT.

4.3. Further Mechanisms

4.3.1. Overload Bit

Overload Bit MUST be respected in all according reachability computations. A node with overload bit set SHOULD NOT advertise any reachability prefixes southbound except locally hosted ones.

The leaf node SHOULD set the ’overload’ bit on its node TIEs, since if the spine nodes were to forward traffic not meant for the local node, the leaf node does not have the topology information to prevent a routing/forwarding loop.

4.3.2. Optimized Route Computation on Leafs

Since the leafs do see only "one hop away" they do not need to run a full SPF but can simply gather prefix candidates from their neighbors and build the according routing table.

A leaf will have no N-TIEs except its own and optionally from its east-west neighbors. A leaf will have S-TIEs from its neighbors.

Instead of creating a network graph from its N-TIEs and neighbor’s S-TIEs and then running an SPF, a leaf node can simply compute the minimum cost and next_hop_set to each leaf neighbor by examining its local interfaces, determining bi-directionality from the associated N-TIE, and specifying the neighbor’s next_hop_set set and cost from the minimum cost local interfaces to that neighbor.

Then a leaf attaches prefixes as in Section 4.2.6 as well as the policy-guided prefixes as in Section 4.2.7.

4.3.3. Key/Value Store

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 optionally allow special leaf East-West adjacencies under additional set of rules. The leaf supporting those procedures MUST:

advertise the LEAF_2_LEAF flag in node capabilities AND

set the overload bit on all leaf’s node TIEs AND

flood only node’s own north and south TIEs over E-W leaf adjacencies AND

always use E-W leaf adjacency in both north as well as south computation AND

install a discard route for any advertised aggregate in leaf’s TIEs AND

never form southbound adjacencies.

This will allow the E-W leaf nodes to exchange traffic strictly for the prefixes advertised in each other's north prefix TIEs (since the southbound computation will find the reverse direction in the other node's TIE and install its north prefixes).

4.3.6. 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 or in other complex link set failure scenarios. Section 5.4 explains the resulting behavior based on one such example.

5. Examples

5.1. Normal Operation

This section describes RIFT deployment in the example topology without any node or link failures. We disregard flooding reduction for simplicity's sake.

As first step, the following bi-directional adjacencies will be created (and any other links that do not fulfill LIE rules in Section 4.2.2 disregarded):

1. Spine 21 (PoD 0) to Node 111, Node 112, Node 121, and Node 122

2. Spine 22 (PoD 0) to Node 111, Node 112, Node 121, and Node 122

3. Node 111 to Leaf 111, Leaf 112

4. Node 112 to Leaf 111, Leaf 112

5. Node 121 to Leaf 121, Leaf 122

6. Node 122 to Leaf 121, Leaf 122

Consequently, N-TIEs would be originated by Node 111 and Node 112 and each set would be sent to both Spine 21 and Spine 22. N-TIEs also would be originated by Leaf 111 (w/ Prefix 111) and Leaf 112 (w/ Prefix 112 and the multi-homed prefix) and each set would be sent to Node 111 and Node 112. Node 111 and Node 112 would then flood these N-TIEs to Spine 21 and Spine 22.

Similarly, N-TIEs would be originated by Node 121 and Node 122 and each set would be sent to both Spine 21 and Spine 22. N-TIEs also would be originated by Leaf 121 (w/ Prefix 121 and the multi-homed prefix) and Leaf 122 (w/ Prefix 122) and each set would be sent to Node 121 and Node 122. Node 121 and Node 122 would then flood these N-TIEs to Spine 21 and Spine 22.

At this point both Spine 21 and Spine 22, as well as any controller to which they are connected, would have the complete network topology. At the same time, Node 111/112/121/122 hold only the N-ties of level 0 of their respective PoD. Leafs hold only their own N-TIEs.

S-TIEs with adjacencies and a default IP prefix would then be originated by Spine 21 and Spine 22 and each would be flooded to Node 111, Node 112, Node 121, and Node 122. Node 111, Node 112, Node 121, and Node 122 would each send the S-TIE from Spine 21 to Spine 22 and

the S-TIE from Spine 22 to Spine 21. (S-TIEs are reflected up to level from which they are received but they are NOT propagated southbound.)

An S Tie with a default IP prefix would be originated by Node 111 and Node 112 and each would be sent to Leaf 111 and Leaf 112. Leaf 111 and Leaf 112 would each send the S-TIE from Node 111 to Node 112 and the S-TIE from Node 112 to Node 111.

Similarly, an S Tie with a default IP prefix would be originated by Node 121 and Node 122 and each would be sent to Leaf 121 and Leaf 122. Leaf 121 and Leaf 122 would each send the S-TIE from Node 121 to Node 122 and the S-TIE from Node 122 to Node 121. At this point IP connectivity with maximum possible ECMP has been established between the leafs while constraining the amount of information held by each node to the minimum necessary for normal operation and dealing with failures.

5.2. Leaf Link Failure

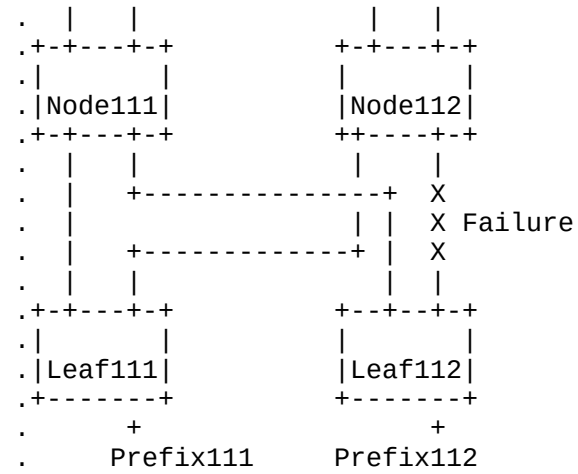


Figure 9: Single Leaf link failure

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

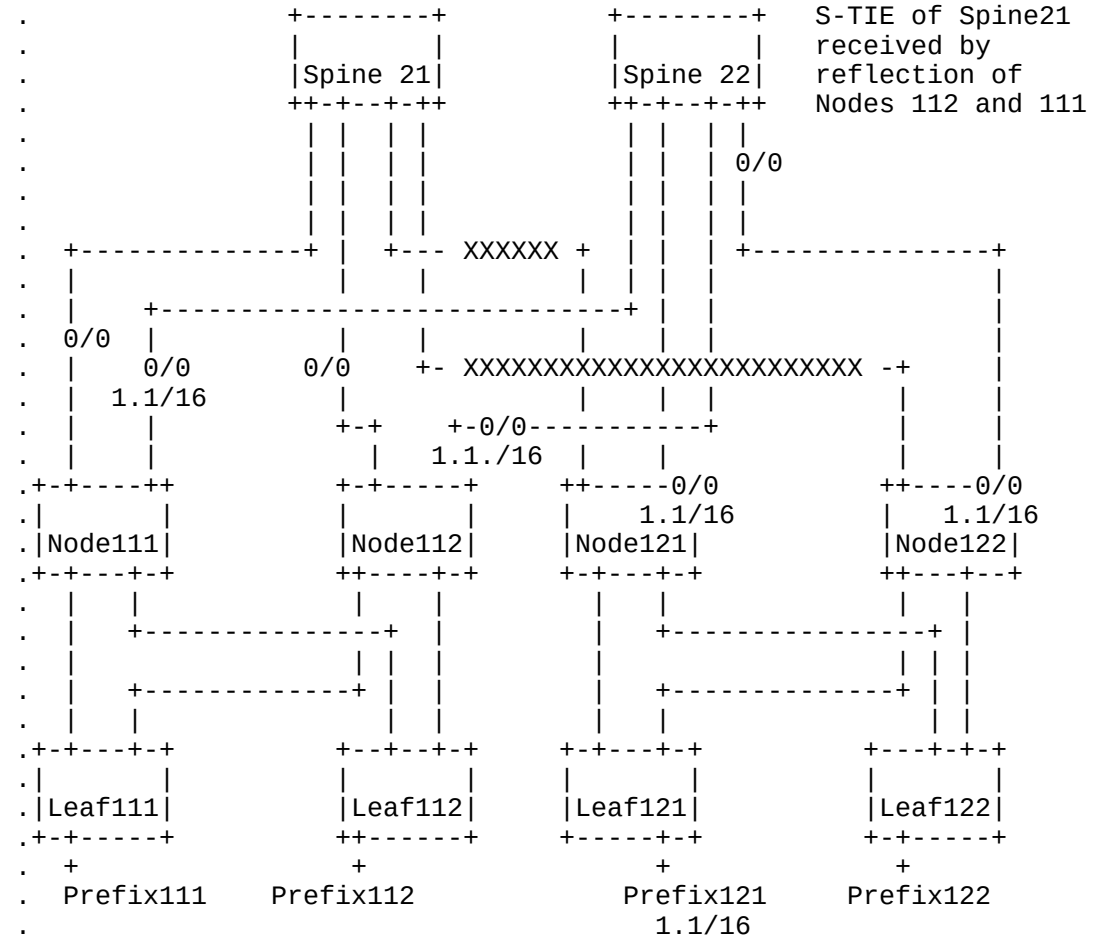


Figure 10: Fabric partition

Figure 10 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

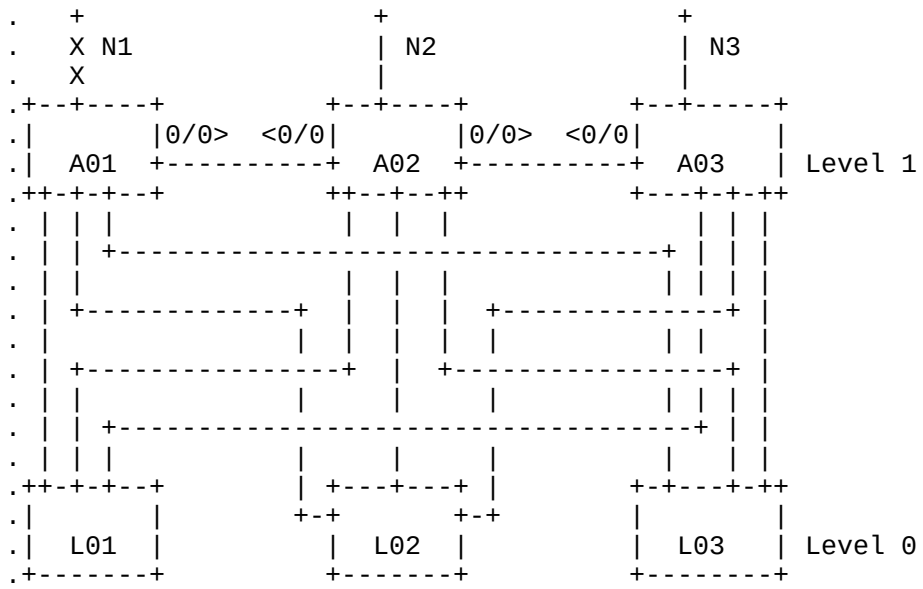


Figure 11: North Partitioned Router

Figure 11 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

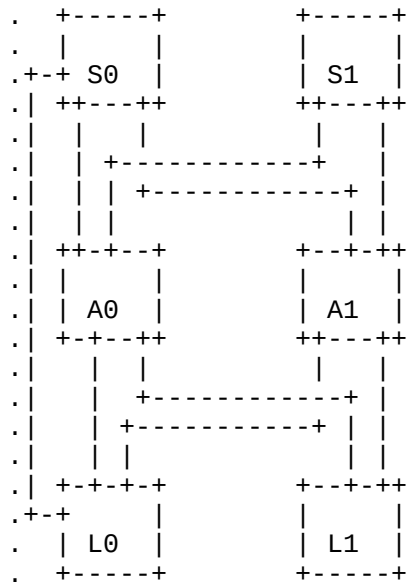


Figure 12: Level Shortcut

Strictly speaking, RIFT is not limited to Clos variations only. The protocol preconditions only a sense of 'compass rose direction' achieved by configuration (or derivation) of levels and other topologies are possible within this framework. So, conceptually, one could include leaf to leaf links and even shortcut between layers but certain requirements in Section 3 will not be met anymore. As an example, shortcutting levels illustrated in Figure 12 will lead either to suboptimal routing when L0 sends traffic to L1 (since using S0's default route will lead to the traffic being sent back to A0 or A1) or the leafs need each other's routes installed to understand that only A0 and A1 should be used to talk to each other.

Whether such modifications of topology constraints make sense is dependent on many technology variables and the exhausting treatment of the topic is definitely outside the scope of this document.

6.3. Originating Non-Default Route Southbound

Obviously, an implementation may choose to originate southbound instead of a strict default route (as described in Section 4.2.3.7) a shorter prefix P' but in such a scenario all addresses carried within the RIFT domain must be contained within P'.

7. Security Considerations

The protocol has provisions for nonces and can include authentication mechanisms in the future comparable to [RFC5709] and [RFC7987].

One can consider additionally attack vectors where a router may reboot many times while changing its system ID and pollute the network with many stale TIEs or TIEs are sent with very long lifetimes and not cleaned up when the routes vanishes. Those attack vectors are not unique to RIFT. Given large memory footprints available today those attacks should be relatively benign. Otherwise a node can implement a strategy of e.g. discarding contents of all TIEs of nodes that were not present in the SPF tree over a certain period of time. Since the protocol, like all modern link-state protocols, is self-stabilizing and will advertise the presence of such TIEs to its neighbors, they can be re-requested again if a computation finds that it sees an adjacency formed towards the system ID of the discarded TIEs.

Section 4.2.9 presents many attack vectors in untrusted environments, starting with nodes that oscillate their level offers to the possiblity of a node offering the highest available level value to put itself "on top of the lattice" and with that observing the whole topology. Session authentication mechanisms are necessary in environments where this is possible.

8. Information Elements Schema

This section introduces the schema for information elements.

On schema changes that

1. change field numbers or

2. add new required fields or

3. remove fields or

4. change lists into sets, unions into structures or

5. change multiplicity of fields or

6. change datatypes of any field or

7. changes default value of any field

major version of the schema MUST increase. All other changes MUST increase minor version within the same major.

Thrift serializer/deserializer MUST not discard optional, unknown fields but preserve and serialize them again when re-flooding whereas missing optional fields MAY be replaced with according default values if present.

All signed integer as forced by Thrift support must be cast for internal purposes to equivalent unsigned values without discarding the signedness bit. An implementation SHOULD try to avoid using the signedness bit when generating values.

The schema is normative.

8.1. common.thrift

```
/**
    Thrift file with common definitions for RIFT
*/

namespace * common

/** @note MUST be interpreted in implementation as unsigned 64 bits.
 *        The implementation SHOULD NOT use the MSB.
 */
typedef i64    SystemIDType
typedef i32    IPv4Address
```

```
/** this has to be of length long enough to accomodate prefix */
typedef binary IPv6Address
/** @note MUST be interpreted in implementation as unsigned 16 bits */
typedef i16    UDPPortType
/** @note MUST be interpreted in implementation as unsigned 32 bits */
typedef i32    TIENrType
/** @note MUST be interpreted in implementation as unsigned 32 bits */
typedef i32    MTUSizeType
/** @note MUST be interpreted in implementation as unsigned 32 bits */
typedef i32    SeqNrType
/** @note MUST be interpreted in implementation as unsigned 32 bits */
typedef i32    LifeTimeInSecType
/** @note MUST be interpreted in implementation as unsigned 16 bits */
typedef i16    LevelType
/** @note MUST be interpreted in implementation as unsigned 32 bits */
typedef i32    PodType
/** @note MUST be interpreted in implementation as unsigned 16 bits */
typedef i16    VersionType
/** @note MUST be interpreted in implementation as unsigned 32 bits */
typedef i32    MetricType
typedef string KeyIDType
/** node local, unique identification for a link (interface/tunnel
  * etc. Basically anything RIFT runs on). This is kept
  * at 32 bits so it aligns with BFD [RFC5880] discriminator size.
  */
typedef i32    LinkIDType
typedef string KeyNameType
typedef i8     PrefixLenType
/** timestamp in seconds since the epoch */
typedef i64    TimestampInSecsType
/** security nonce */
typedef i64    NonceType
/** adjacency holdtime */
typedef i16    HoldTimeInSecType

/** Flags indicating nodes behavior in case of ZTP and support
    for special optimization procedures.
 */
enum LeafIndications {
    NO_RESTRICTIONS                     =0,
    LEAF_ONLY                           =1,
    LEAF_ONLY_AND_LEAF_2_LEAF_PROCEDURES =2,
}

/** fixed leaf level when ZTP is not used */
const LevelType   leaf_level              = 0
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 LeafIndications leaf_indication_default = LeafIndications.NO_RESTRICTIONS
const bool south_transitive_default       = false
const HoldTimeInSecType default_holdtime  = 3
/** 0 is illegal for SystemID */
const SystemIDType IllegalSystemID        = 0
/** empty set of nodes */
const set<SystemIDType> empty_set_of_nodeids = {}

/** default UDP port to run LIEs on */
const UDPPortType     default_lie_udp_port       =  6949
const UDPPortType     default_tie_udp_flood_port =  6950

/** default MTU size to use */
const MTUSizeType     default_mtu_size           =  1400
/** default mcast is v4 224.0.1.150, we make it i64 to
 *  help languages struggling with highest bit */
const i64             default_lie_v4_mcast_group =  3758096790

/** indicates whether the direction is northbound/east-west
  * or southbound */
enum TieDirectionType {
    Illegal           = 0,
    South             = 1,
    North             = 2,
    DirectionMaxValue = 3,
}

enum AddressFamilyType {
   Illegal                = 0,
   AddressFamilyMinValue  = 1,
   IPv4     = 2,
   IPv6     = 3,
   AddressFamilyMaxValue  = 4,
}

struct IPv4PrefixType {
    1: required IPv4Address    address;
    2: required PrefixLenType  prefixlen;
```

```
}

struct IPv6PrefixType {
    1: required IPv6Address    address;
    2: required PrefixLenType  prefixlen;
}

union IPAddressType {
    1: optional IPv4Address   ipv4address;
    2: optional IPv6Address   ipv6address;
}

union IPPrefixType {
    1: optional IPv4PrefixType   ipv4prefix;
    2: optional IPv6PrefixType   ipv6prefix;
}

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

/** @note: route types which MUST be ordered on their preference
 *  PGP prefixes are most preferred attracting
 *  traffic north (towards spine) and then south
 *  normal prefixes are attracting traffic south (towards leafs),
 *  i.e. prefix in NORTH PREFIX TIE is preferred over SOUTH PREFIX TIE
 */
enum RouteType {
    Illegal               = 0,
    RouteTypeMinValue     = 1,
    /** First legal value. */
    /** Discard routes are most prefered */
    Discard               = 2,

    /** Local prefixes are directly attached prefixes on the
     *  system such as e.g. interface routes.
     */
    LocalPrefix           = 3,
    /** advertised in S-TIEs */
    SouthPGPPrefix        = 4,
    /** advertised in N-TIEs */
```

```
    NorthPGPPrefix        = 5,
    /** advertised in N-TIEs */
    NorthPrefix           = 6,
    /** advertised in S-TIEs */
    SouthPrefix           = 7,
    /** transitive southbound are least preferred */
    TransitiveSouthPrefix = 8,
    RouteTypeMaxValue     = 9
}
```

8.2. encoding.thrift

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

include "common.thrift"

namespace * encoding

/** represents protocol encoding schema major version */
const i32 protocol_major_version = 5
/** represents protocol encoding schema minor version */
const i32 protocol_minor_version = 0

/** common RIFT packet header */
struct PacketHeader {
    1: required common.VersionType major_version = protocol_major_version;
    2: required common.VersionType minor_version = protocol_minor_version;
    /** this is the node sending the packet, in case of LIE/TIRE/TIDE
        also the originator of it */
    3: required common.SystemIDType  sender;
    /** level of the node sending the packet, required on everything except
      * LIEs. Lack of presence on LIEs indicates UNDEFINED_LEVEL and is used
      * in ZTP procedures.
     */
    4: optional common.LevelType     level;
}

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

/** Neighbor structure  */
struct Neighbor {
    1: required common.SystemIDType        originator;
```

```
    2: required common.LinkIDType          remote_id;
}

/** Capabilities the node supports */
struct NodeCapabilities {
    /** can this node participate in flood reduction,
        only relevant at level > 0 */
    1: optional bool                       flood_reduction =
            common.flood_reduction_default;
    /** does this node restrict itself to be leaf only (in ZTP) and
        does it support leaf-2-leaf procedures */
    2: optional common.LeafIndications     leaf_indications =
            LeafIndications.NO_RESTRICTIONS;
}

/** RIFT LIE packet

    @note this node's level is already included on the packet header */
struct LIEPacket {
    /** optional node or adjacency name */
    1: optional string                     name;
    /** local link ID */
    2: required common.LinkIDType          local_id;
    /** UDP port to which we can receive flooded TIEs */
    3: required common.UDPPortType         flood_port =
            common.default_tie_udp_flood_port;
    /** layer 3 MTU */
    4: optional common.MTUSizeType         link_mtu_size =
            common.default_mtu_size;
    /** this will reflect the neighbor once received */
    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;
    /** required holdtime of the adjacency, i.e. how much time
        MUST expire without LIE for the adjacency to drop
     */
    9: required common.HoldTimeInSecType   holdtime =
            common.default_holdtime;
}

/** LinkID pair describes one of parallel links between two nodes */
```

```
struct LinkIDPair {
    /** node-wide unique value for the local link */
    1: required common.LinkIDType      local_id;
    /** received remote link ID for this link */
    2: required common.LinkIDType      remote_id;
    /** more properties of the link can go in here */
}

/** ID of a TIE

    @note: TIEID space is a total order achieved by comparing the elements
           in sequence defined and comparing each value as an
           unsigned integer of according length
*/
struct TIEID {
    /** indicates direction of the TIE */
    1: required common.TieDirectionType    direction;
    /** indicates originator of the TIE */
    2: required common.SystemIDType        originator;
    3: required common.TIETypeType         tietype;
    4: required common.TIENrType           tie_nr;
}

/** Header of a TIE */
struct TIEHeader {
    2: required TIEID                      tieid;
    3: required common.SeqNrType           seq_nr;
    /** lifetime expires down to 0 just like in ISIS */
    4: required common.LifeTimeInSecType   lifetime;
}

/** A sorted TIDE packet, if unsorted, behavior is undefined */
struct TIDEPacket {
    /** all 00s marks starts */
    1: required TIEID           start_range;
    /** all FFs mark end */
    2: required TIEID           end_range;
    /** _sorted_ list of headers */
    3: required list<TIEHeader> headers;
}

/** A TIRE packet */
struct TIREPacket {
    1: required set<TIEHeader> headers;
}

/** Neighbor of a node */
struct NodeNeighborsTIEElement {
```

```
    2: required common.LevelType       level;
    /**  Cost to neighbor.

         @note: All parallel links to same node
         incur same cost, in case the neighbor has multiple
         parallel links at different cost, the largest distance
         (highest numerical value) MUST be advertised
         @note: any neighbor with cost <= 0 MUST be ignored in computations */
    3: optional common.MetricType      cost = common.default_distance;
    /** can carry description of multiple parallel links in a TIE */
    4: optional set<LinkIDPair>        link_ids;
}

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

/** Description of a node.

    It may occur multiple times in different TIEs but if either
        * capabilities values do not match or
        * flags values do not match or
        * neighbors repeat with different values or
        * visible in same level/having partition upper do not match
    the behavior is undefined and a warning SHOULD be generated.
    Neighbors can be distributed across multiple TIEs however if
    the sets are disjoint.

    @note: observe that absence of fields implies defined defaults
*/
struct NodeTIEElement {
    1: required common.LevelType            level;
    /** if neighbor systemID repeats in other node TIEs of same node
        the behavior is undefined. Equivalent to |A_(n,s)(N) in spec. */
    2: required map<common.SystemIDType,
                NodeNeighborsTIEElement>    neighbors;
    3: optional NodeCapabilities            capabilities;
    4: optional NodeFlags                   flags;
    /** optional node name for easier operations */
    5: optional string                      name;

    /** Nodes seen an the same level through reflection through nodes
        having backlink to both nodes. They are equivalent to |V(N) in
        future specifications. Ignored in Node S-TIEs if present.
      */
    6: optional set<common.SystemIDType>    visible_in_same_level
```

```
            = common.empty_set_of_nodeids;
    /** Non-overloaded nodes in |V seen as attached to another north
      * level partition due to the fact that some nodes in its |V have
      * adjacencies to higher level nodes that this node doesn't see.
      * This may be used in the computation at higher levels to prevent
      * blackholing. Ignored in Node S-TIEs if present.
      * Equivalent to |PUL(N) in spec. */
    7: optional set<common.SystemIDType>    same_level_unknown_north_partitions
            = common.empty_set_of_nodeids;
}

struct PrefixAttributes {
    2: required common.MetricType       metric = common.default_distance;
    /** Respected only on prefixes advertised in S-TIEs, indicates that
     *  prefix SHOULD be propagated southwards towards lower levels to heal
     *  upper level partitioning, otherwise blackholes may occur.
     *  Required since very implementation MUST understand it.
     */
    3: required bool                    south_transitive =
            common.south_transitive_default;
}

/** multiple prefixes */
struct PrefixTIEElement {
    /** prefixes with the associated attributes.
        if the same prefix repeats in multiple TIEs of same node
        behavior is unspecified */
    1: required map<common.IPPrefixType, PrefixAttributes> prefixes;
}

/** keys with their values */
struct KeyValueTIEElement {
    /** if the same key repeats in multiple TIEs of same node
        or with different values, behavior is unspecified */
    1: required map<common.KeyIDType,string>    keyvalues;
}

/** single element in a TIE. enum common.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 */
}

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

union PacketContent {
    1: optional LIEPacket    lie;
    2: optional TIDEPacket   tide;
    3: optional TIREPacket   tire;
    4: optional TIEPacket    tie;
}

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

9. IANA Considerations

This specification will request at an opportune time multiple registry points to exchange protocol packets in a standardized way, amongst them multicast address assignments and standard port numbers. The schema itself defines many values and codepoints which can be considered registries themselves.

10. Acknowledgments

Many thanks to Naiming Shen for some of the early discussions around the topic of using IGPs for routing in topologies related to Clos. Adrian Farrel, Joel Halpern and Jeffrey Zhang provided thoughtful comments that improved the readability of the document and found good amount of corners where the light failed to shine. Kris Price was first to mention single router, single arm default considerations. Jeff Tantsura helped out with some initial thoughts on BFD interactions while Jeff Haas corrected several misconceptions about BFD’s finer points. Artur Makutunowicz pointed out many possible improvements and acted as sounding board in regard to modern protocol implementation techniques RIFT is exploring. Barak Gafni formalized first time clearly the problem of partitioned spine on a (clean) napkin in Singapore.

11. References

11.1. Normative References

[ISO10589] ISO "International Organization for Standardization", "Intermediate system to Intermediate system intra-domain routeing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode Network Service (ISO 8473), ISO/IEC 10589:2002, Second Edition.", Nov 2002.

[RFC1142] Oran, D., Ed., "OSI IS-IS Intra-domain Routing Protocol", RFC 1142, DOI 10.17487/RFC1142, February 1990, <https://www.rfc-editor.org/info/rfc1142>.

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

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

[RFC2365] Meyer, D., "Administratively Scoped IP Multicast", BCP 23, RFC 2365, DOI 10.17487/RFC2365, July 1998, <https://www.rfc-editor.org/info/rfc2365>.

[RFC4271] Rekhter, Y., Ed., Li, T., Ed., and S. Hares, Ed., "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, DOI 10.17487/RFC4271, January 2006, <https://www.rfc-editor.org/info/rfc4271>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/info/rfc4291>.

[RFC4655] Farrel, A., Vasseur, J., and J. Ash, "A Path Computation Element (PCE)-Based Architecture", RFC 4655, DOI 10.17487/RFC4655, August 2006, <https://www.rfc-editor.org/info/rfc4655>.

[RFC5120] Przygienda, T., Shen, N., and N. Sheth, "M-ISIS: Multi Topology (MT) Routing in Intermediate System to Intermediate Systems (IS-ISs)", RFC 5120, DOI 10.17487/RFC5120, February 2008, <https://www.rfc-editor.org/info/rfc5120>.

[RFC5303] Katz, D., Saluja, R., and D. Eastlake 3rd, "Three-Way Handshake for IS-IS Point-to-Point Adjacencies", RFC 5303, DOI 10.17487/RFC5303, October 2008, <https://www.rfc-editor.org/info/rfc5303>.

[RFC5709] Bhatia, M., Manral, V., Fanto, M., White, R., Barnes, M., Li, T., and R. Atkinson, "OSPFv2 HMAC-SHA Cryptographic Authentication", RFC 5709, DOI 10.17487/RFC5709, October 2009, <https://www.rfc-editor.org/info/rfc5709>.

[RFC5881] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD) for IPv4 and IPv6 (Single Hop)", RFC 5881, DOI 10.17487/RFC5881, June 2010, <https://www.rfc-editor.org/info/rfc5881>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[RFC6822] Previdi, S., Ed., Ginsberg, L., Shand, M., Roy, A., and D. Ward, "IS-IS Multi-Instance", RFC 6822, DOI 10.17487/RFC6822, December 2012, <https://www.rfc-editor.org/info/rfc6822>.

[RFC7855] Previdi, S., Ed., Filsfils, C., Ed., Decraene, B., Litkowski, S., Horneffer, M., and R. Shakir, "Source Packet Routing in Networking (SPRING) Problem Statement and Requirements", RFC 7855, DOI 10.17487/RFC7855, May 2016, <https://www.rfc-editor.org/info/rfc7855>.

[RFC7938] Lapukhov, P., Premji, A., and J. Mitchell, Ed., "Use of BGP for Routing in Large-Scale Data Centers", RFC 7938, DOI 10.17487/RFC7938, August 2016, <https://www.rfc-editor.org/info/rfc7938>.

[RFC7987] Ginsberg, L., Wells, P., Decraene, B., Przygienda, T., and H. Gredler, "IS-IS Minimum Remaining Lifetime", RFC 7987, DOI 10.17487/RFC7987, October 2016, <https://www.rfc-editor.org/info/rfc7987>.

11.2. Informative References

[CLOS] Yuan, X., "On Nonblocking Folded-Clos Networks in Computer Communication Environments", IEEE International Parallel & Distributed Processing Symposium, 2011.

[DIJKSTRA]
Dijkstra, E., "A Note on Two Problems in Connexion with Graphs", Journal Numer. Math. , 1959.

[DYNAMO] De Candia et al., G., "Dynamo: amazon's highly available key-value store", ACM SIGOPS symposium on Operating systems principles (SOSP '07), 2007.

[EPPSTEIN]
Eppstein, D., "Finding the k-Shortest Paths", 1997.

[FATTREE] Leiserson, C., "Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing", 1985.

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

[PROTOBUF]
Google, Inc., "Protocol Buffers, https://developers.google.com/protocol-buffers".

[QUIC] Iyengar et al., J., "QUIC: A UDP-Based Multiplexed and Secure Transport", 2016.

[VAHDAT08]
Al-Fares, M., Loukissas, A., and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture", SIGCOMM , 2008.

Authors' Addresses

Tony Przygienda (editor)
Juniper Networks
1194 N. Mathilda Ave
Sunnyvale, CA 94089
US

Email: prz@juniper.net

Alankar Sharma
Comcast
1800 Bishops Gate Blvd
Mount Laurel, NJ 08054
US

Email: Alankar_Sharma@comcast.com

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

Email: akatlas@juniper.net

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

Email: jdrake@juniper.net

Networking Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 6, 2018

N. Shen
L. Ginsberg
Cisco Systems
S. Thyamagundalu

January 2, 2018

IS-IS Routing for Spine-Leaf Topology
draft-shen-isis-spine-leaf-ext-05

Abstract

This document describes a mechanism for routers and switches in a Spine-Leaf type topology to have non-reciprocal Intermediate System to Intermediate System (IS-IS) routing relationships between the leafs and spines. The leaf nodes do not need to have the topology information of other nodes and exact prefixes in the network. This extension also has application in the Internet of Things (IoT).

Table of Contents

1. Introduction 3
1.1. Requirements Language 3
2. Motivations 3
3. Spine-Leaf (SL) Extension 4
3.1. Topology Examples 4
3.2. Applicability Statement 5
3.3. Extension Encoding 6
3.3.1. Spine-Leaf Sub-TLVs 7
3.3.1.1. Leaf-Set Sub-TLV 8
3.3.1.2. Info-Req Sub-TLV 8
3.3.2. Advertising IPv4/IPv6 Reachability 8
3.4. Mechanism 8
3.4.1. Pure CLOS Topology 10
3.5. Implementation and Operation 11
3.5.1. CSNP PDU 11
3.5.2. Leaf to Leaf connection 11
3.5.3. Overload Bit 11
3.5.4. Spine Node Hostname 12
3.5.5. IS-IS Reverse Metric 12
3.5.6. Spine-Leaf Traffic Engineering 12
3.5.7. Other End-to-End Services 12
3.5.8. Address Family and Topology 13
3.5.9. Migration 13
4. IANA Considerations 13
5. Security Considerations 14
6. Acknowledgments 14
7. Document Change Log 14
7.1. Changes to draft-shen-isis-spine-leaf-ext-05.txt 14
7.2. Changes to draft-shen-isis-spine-leaf-ext-04.txt 14
7.3. Changes to draft-shen-isis-spine-leaf-ext-03.txt 14
7.4. Changes to draft-shen-isis-spine-leaf-ext-02.txt 14
7.5. Changes to draft-shen-isis-spine-leaf-ext-01.txt 15
7.6. Changes to draft-shen-isis-spine-leaf-ext-00.txt 15
8. References 15
8.1. Normative References 15
8.2. Informative References 16
Authors' Addresses 17

1. Introduction

The IS-IS routing protocol defined by [ISO10589] has been widely deployed in provider networks, data centers and enterprise campus environments. In the data center and enterprise switching networks, a Spine-Leaf topology is commonly used. This document describes a mechanism where IS-IS routing can be optimized for a Spine-Leaf topology.

In a Spine-Leaf topology, normally a leaf node connects to a number of spine nodes. Data traffic going from one leaf node to another leaf node needs to pass through one of the spine nodes. Also, the decision to choose one of the spine nodes is usually part of equal cost multi-path (ECMP) load sharing. The spine nodes can be considered as gateway devices to reach destinations on other leaf nodes. In this type of topology, the spine nodes have to know the topology and routing information of the entire network, but the leaf nodes only need to know how to reach the gateway devices to which are the spine nodes they are uplinked.

This document describes the IS-IS Spine-Leaf extension that allows the spine nodes to have all the topology and routing information, while keeping the leaf nodes free of topology information other than the default gateway routing information. The leaf nodes do not even need to run a Shortest Path First (SPF) calculation since they have no topology information.

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. Motivations

- The leaf nodes in a Spine-Leaf topology do not require complete topology and routing information of the entire domain since their forwarding decision is to use ECMP with spine nodes as default gateways

- The spine nodes in a Spine-Leaf topology are richly connected to leaf nodes, which introduces significant flooding duplication if they flood all Link State PDUs (LSPs) to all the leaf nodes. It saves both spine and leaf nodes' CPU and link bandwidth resources if flooding is blocked to leaf nodes. For small Top of the Rack (ToR) leaf switches in data centers, it is meaningful to prevent full topology routing information and massive database flooding through those devices.

- When a spine node advertises a topology change, every leaf node connected to it will flood the update to all the other spine nodes, and those spine nodes will further flood them to all the leaf nodes, causing a O(n^2) flooding storm which is largely redundant.

- Similar to some of the overlay technologies which are popular in data centers, the edge devices (leaf nodes) may not need to contain all the routing and forwarding information on the device's control and forwarding planes. "Conversational Learning" can be utilized to get the specific routing and forwarding information in the case of pure CLOS topology and in the events of link and node down.

- Small devices and appliances of Internet of Things (IoT) can be considered as leafs in the routing topology sense. They have CPU and memory constrains in design, and those IoT devices do not have to know the exact network topology and prefixes as long as there are ways to reach the cloud servers or other devices.

3. Spine-Leaf (SL) Extension

3.1. Topology Examples

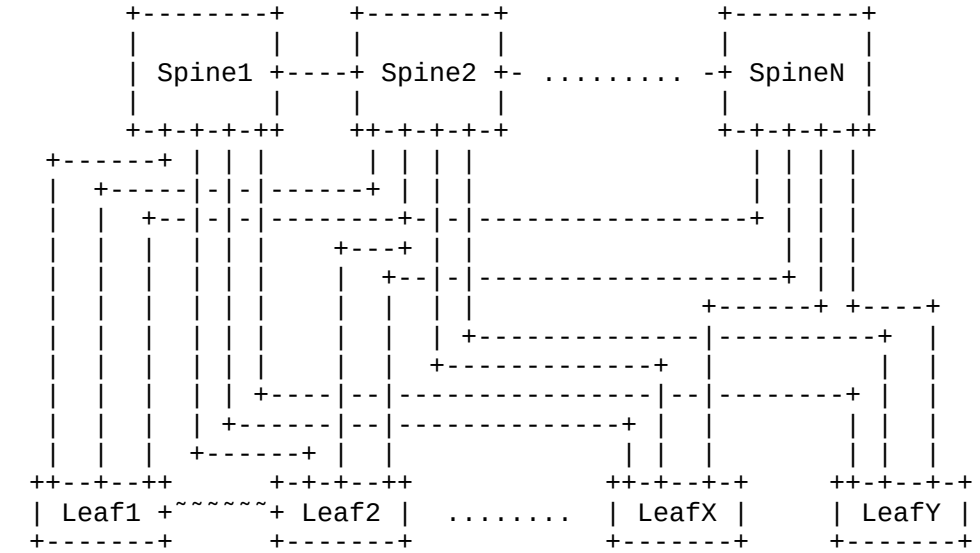


Figure 1: A Spine-Leaf Topology

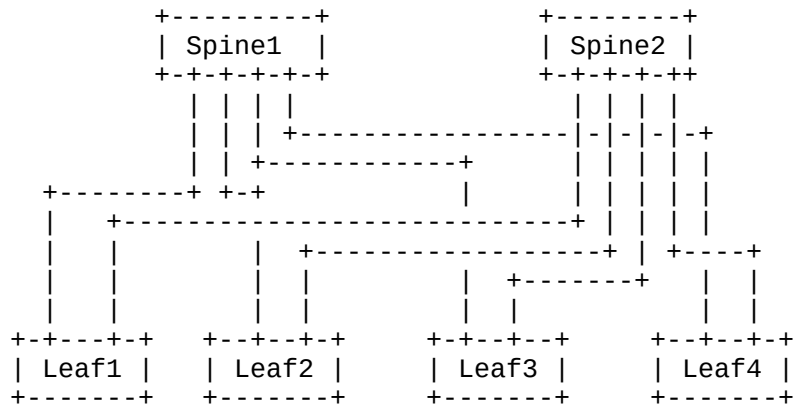


Figure 2: A CLOS Topology

3.2. Applicability Statement

This extension assumes the network is a Spine-Leaf topology, and it should not be applied in an arbitrary network setup. The spine nodes can be viewed as the aggregation layer of the network, and the leaf nodes as the access layer of the network. The leaf nodes use a load sharing algorithm with spine nodes as nexthops in routing and forwarding.

This extension works when the spine nodes are inter-connected, and it works with a pure CLOS or Fat Tree topology based network where the spines are NOT horizontally interconnected.

Although the example diagram in Figure 1 shows a fully meshed Spine-Leaf topology, this extension also works in the case where they are partially meshed. For instance, leaf1 through leaf10 may be fully meshed with spine1 through spine5 while leaf11 through leaf20 is fully meshed with spine4 through spine8, and all the spines are inter-connected in a redundant fashion.

This extension can also work in multi-level spine-leaf topology. The lower level spine node can be a 'leaf' node to the upper level spine node. A spine-leaf 'Tier' can be exchanged with IS-IS hello packets to allow tier X to be connected with tier X+1 using this extension. Normally tier-0 will be the TOR routers and switches if provisioned.

This extension also works with normal IS-IS routing in a topology with more than two layers of spine and leaf. For instance, in example diagrams Figure 1 and Figure 2, there can be another Core layer of routers/switches on top of the aggregation layer. From an IS-IS routing point of view, the Core nodes are not affected by this

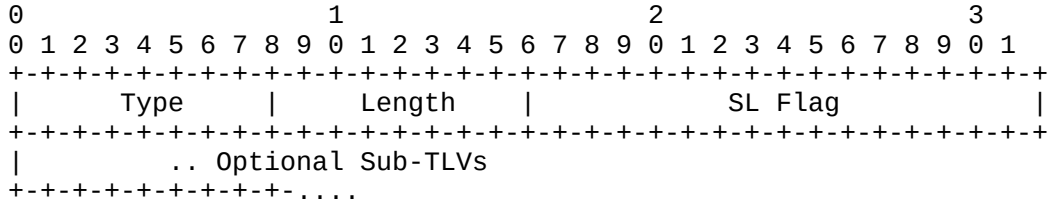
extension and will have the complete topology and routing information just like the spine nodes. To make the network even more scalable, the Core layer can operate as a level-2 IS-IS sub-domain while the Spine and Leaf layers operate as stays at the level-1 IS-IS domain.

This extension also supports the leaf nodes having local connections to other leaf nodes, in the example diagram Figure 1 there is a connection between 'Leaf1' node and 'Leaf2' node, and an external host can be dual homed into both of the leaf nodes.

This extension assumes the link between the spine and leaf nodes are point-to-point, or point-to-point over LAN [RFC5309]. The links connecting among the spine nodes or the links between the leaf nodes can be any type.

3.3. Extension Encoding

This extension introduces one new TLV which may be used in IS-IS Hello (IIH) PDUs, LSPs, or in Circuit Scoped Link State PDUs (CS-LSP) [RFC7356]. It is used by both spine and leaf nodes in this Spine-Leaf mechanism.

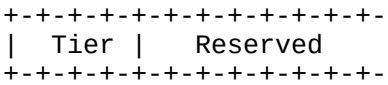


The fields of this TLV are defined as follows:

Type: 1 octet Suggested value 150 (to be assigned by IANA)

Length: 1 octet (2 + length of sub-TLVs).

Flags 16 bits



Tier: A 4 bits value range from 0 to 15. It is used to represent the spine-leaf tier level when the 'T' bit is set. If the 'T' is cleared, this value MUST be set to zero from the sender, and it MUST be ignored on the receiver. The value 15 is reserved to indicate the tier level is unknown or not configured.

L bit (0x01): Only leaf node sets this bit. If the L bit is set in the SL flag, the node indicates it is in 'Leaf-Mode'.

R bit (0x02): Only Spine node sets this bit. If the R bit is set, the node indicates to the leaf neighbor that it can be used as the default route gateway.

B bit (0x04): Only leaf node sets this bit on Leaf-Leaf link, in additional to the 'L' bit setting. If the B bit is set, the node indicates to its leaf neighbor that it can be used as the backup default route gateway.

T bit (0x08): If set, the value in the 'Tier' field represents the spine-leaf tier level in the topology.

Optional Sub-TLV: Not defined in this document, for future extension

sub-TLVs MAY be included when the TLV is in a CS-LSP.
sub-TLVs MUST NOT be included when the TLV is in an IIH

3.3.1. Spine-Leaf Sub-TLVs

If the data center topology is a pure CLOS or Fat Tree, there are no link connections among the spine nodes. If we also assume there is not another Core layer on top of the aggregation layer, then the traffic from one leaf node to another may have a problem if there is a link outage between a spine node and a leaf node. For instance, in the diagram of Figure 2, if Leaf1 sends data traffic to Leaf3 through Spine1 node, and the Spine1-Leaf3 link is down, the data traffic will be dropped on the Spine1 node.

To address this issue spine and leaf nodes may send/request specific reachability information via the sub-TLVs defined below.

Two Spine-Leaf sub-TLVs are defined. The Leaf-Set sub-TLV and the Info-Req sub-TLV.

3.3.1.1. Leaf-Set Sub-TLV

This sub-TLV is used by spine nodes to optionally advertise Leaf neighbors to other Leaf nodes. The fields of this sub-TLV are defined as follows:

Type: 1 octet Suggested value 1 (to be assigned by IANA)

Length: 1 octet MUST be a multiple of 6 octets.

Leaf-Set: A list of IS-IS System-ID of the leaf node neighbors of this spine node.

3.3.1.2. Info-Req Sub-TLV

This sub-TLV is used by leaf nodes to request more specific prefix information from a selected spine node, upon detecting one of the spine node has lost the connection to a leaf node. The fields of this sub-TLV are defined as follows:

Type: 1 octet Suggested value 2 (to be assigned by IANA)

Length: 1 octet. It MUST be a multiple of 6 octets.

Info-Req: List of IS-IS System-IDs of leaf nodes for which connectivity information is being requested.

3.3.2. Advertising IPv4/IPv6 Reachability

In cases where connectivity between a leaf node and a spine node is down, the leaf node MAY request reachability information from a spine node as described in Section 3.3.1.2. The spine node utilizes TLVs 135 [RFC5305] and TLVs 236 [RFC5308] to advertise this information. These TLVs MAY be included either in IIHs or CS-LSPs sent from the spine to the requesting leaf node. Sending such information in IIHs has limited scale - all reachability information MUST fit within a single IIH. It is therefore recommended that CS-LSPs be used.

3.4. Mechanism

Leaf nodes in a spine-leaf application using this extension are provisioned with two attributes:

1)Tier level of 0. This indicates the node is a Leaf Node. The value 0 is advertised in the Tier field of Spine-Leaf TLV defined above.

2)Flooding reduction enabled/disabled. If flooding reduction is enabled the L-bit is set to one in the Spine-Leaf TLV defined above

A spine node does not need explicit configuration. Spine nodes can dynamically discover their tier level by computing the number of hops to a leaf node. Until a spine node determines its tier level it MUST advertise level 15 (unknown tier level) in the Spine-Leaf TLV defined above.

When a spine node receives an IIH which includes the Spine-Leaf TLV with Tier level 0 and 'L' bit set, it labels the point-to-point interface and adjacency to be a 'Reduced Flooding Leaf-Peer (RF-Leaf)'. IIHs sent by a spine node on a link to an RF-Leaf include the Spine-Leaf TLV with the 'R' bit set in the flags field. The 'R' bit indicates to the RF-Leaf neighbor that the spine node can be used as a default routing nexthop.

There is no change to the IS-IS adjacency bring-up mechanism for Spine-Leaf peers.

A spine node blocks LSP flooding to RF-Leaf adjacencies, except for the LSP PDUs in which the IS-IS System-ID matches the System-ID of the RF-Leaf neighbor. This exception is needed since when the leaf node reboots, the spine node needs to forward to the leaf node non-purged LSPs from the RF-Leaf's previous incarnation.

Leaf nodes will perform IS-IS LSP flooding as normal over all of its IS-IS adjacencies, but in the case of RF-Leafs only self-originated LSPs will exist in its LSP database.

Spine nodes will receive all the LSP PDUs in the network, including all the spine nodes and leaf nodes. It will perform Shortest Path First (SPF) as a normal IS-IS node does. There is no change to the route calculation and forwarding on the spine nodes.

RF-Leaf nodes do not have any LSP in the network except for its own. Therefore there is no need to perform SPF calculation on the RF-Leaf node. It only needs to download the default route with the nexthops of those Spine Neighbors which have the 'R' bit set in the Spine-Leaf TLV in IIH PDUs. IS-IS can perform equal cost or unequal cost load sharing while using the spine nodes as nexthops. The aggregated metric of the outbound interface and the 'Reverse Metric' [REVERSE-METRIC] can be used for this purpose.

3.4.1. Pure CLOS Topology

In a data center where the topology is pure CLOS or Fat Tree, there is no interconnection among the spine nodes, and there is not another Core layer above the aggregation layer with reachability to the leaf nodes. When flooding reduction to RF-Leafs is in use, if the link between a spine and a leaf goes down, there is then a possibility of black holing the data traffic in the network.

As in the diagram Figure 2, if the link Spine1-Leaf3 goes down, there needs to be a way for Leaf1, Leaf2 and Leaf4 to avoid the Spine1 if the destination of data traffic is to Leaf3 node.

In the above example, the Spine1 and Spine2 are provisioned to advertise the Leaf-Set sub-TLV of the Spine-Leaf TLV. Originally both Spines will advertise Leaf1 through Leaf4 as their Leaf-Set. When the Spine1-Leaf3 link is down, Spine1 will only have Leaf1, Leaf2 and Leaf4 in its Leaf-Set. This allows the other leaf nodes to know that Spine1 has lost connectivity to the leaf node of Leaf3.

Each RF-Leaf node can select another spine node to request for some prefix information associated with the lost leaf node. In this diagram of Figure 2, there are only two spine nodes (Spine-Leaf topology can have more than two spine nodes in general). Each RF-Leaf node can independently select a spine node for the leaf information. The RF-Leaf nodes will include the Info-Req sub-TLV in the Spine-Leaf TLV in hellos sent to the selected spine node, Spine2 in this case.

The spine node, upon receiving the request from one or more leaf nodes, will find the IPv6/IPv4 prefixes advertised by the leaf nodes listed in the Info-Req sub-TLV. The spine node will use the mechanism defined in Section 3.3.2 to advertise these prefixes to the RF-Leaf node. For instance, it will include the IPv4 loopback prefix of leaf3 based on the policy configured or administrative tag attached to the prefixes. When the leaf nodes receive the more specific prefixes, they will install the advertised prefixes towards the other spine nodes (Spine2 in this example).

For instance in the data center overlay scenario, when any IP destination or MAC destination uses the leaf3’s loopback as the tunnel nexthop, the overlay tunnel from leaf nodes will only select Spine2 as the gateway to reach leaf3 as long as the Spine1-Leaf3 link is still down.

This negative routing is only relevant between tier 0 and tier 1 spine-leaf levels in a multi-level spine-leaf topology when the

reduced flooding extension is in use. Nodes in tiers 1 or greater have the full topology information.

3.5. Implementation and Operation

3.5.1. CSNP PDU

In Spine-Leaf extension, Complete Sequence Number PDU (CSNP) does not need to be transmitted over the Spine-Leaf link to an RF-Leaf. Some IS-IS implementations send periodic CSNPs after the initial adjacency bring-up over a point-to-point interface. There is no need for this optimization here since the RF-Leaf does not need to receive any other LSPs from the network, and the only LSPs transmitted across the Spine-Leaf link is the leaf node LSP.

Also in the graceful restart case[RFC5306], for the same reason, there is no need to send the CSNPs over the Spine-Leaf interface to an RF-Leaf. Spine nodes only need to set the SRMflag on the LSPs belonging to the RF-Leaf.

3.5.2. Leaf to Leaf connection

Leaf to leaf node links are useful in host redundancy cases in switching networks, and normally there is no flooding extensions are required in this case. Each leaf node will set tier level = 0 in the Spine-Leaf TLV included in hellos to leaf neighbors. LSP will be exchanged over this link. In the example diagram Figure 1, the Leaf1 will get Leaf2’s LSP and Leaf2 will get Leaf1’s LSP. They will install more specific routes towards each other using this local Leaf-Leaf link. SPF will be performed in this case just like when the entire network only involves with those two IS-IS nodes. This does not affect the normal Spine-Leaf mechanism they perform toward the spine nodes.

Besides the local leaf-to-leaf traffic, the leaf node can serve as a backup gateway for its leaf neighbor. It needs to remove the ’Overload-Bit’ setting in its LSP, and it sets both the ’L’ bit and the ’B’ bit in the SL-flag with a high ’Reverse Metric’ value.

3.5.3. Overload Bit

The leaf node SHOULD set the ’overload’ bit on its LSP PDU, 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.

3.5.4. Spine Node Hostname

This extension creates a non-reciprocal relationship between the spine node and leaf node. The spine node will receive leaf's LSP and will know the leaf's hostname, but the leaf does not have spine's LSP. This extension allows the Dynamic Hostname TLV [RFC5301] to be optionally included in spine's IIH PDU when sending to a 'Leaf-Peer'. This is useful in troubleshooting cases.

3.5.5. IS-IS Reverse Metric

This metric is part of the aggregated metric for leaf's default route installation with load sharing among the spine nodes. When a spine node is in 'overload' condition, it should use the IS-IS Reverse Metric TLV in IIH [REVERSE-METRIC] to set this metric to maximum to discourage the leaf using it as part of the loadsharing.

In some cases, certain spine nodes may have less bandwidth in link provisioning or in real-time condition, and it can use this metric to signal to the leaf nodes dynamically.

In other cases, such as when the spine node loses a link to a particular leaf node, although it can redirect the traffic to other spine nodes to reach that destination leaf node, but it MAY want to increase this metric value if the inter-spine connection becomes over utilized, or the latency becomes an issue.

In the leaf-leaf link as a backup gateway use case, the 'Reverse Metric' SHOULD always be set to very high value.

3.5.6. Spine-Leaf Traffic Engineering

Besides using the IS-IS Reverse Metric by the spine nodes to affect the traffic pattern for leaf default gateway towards multiple spine nodes, the IPv6/IPv4 Info-Advertise sub-TLVs can be selectively used by traffic engineering controllers to move data traffic around the data center fabric to alleviate congestion and to reduce the latency of a certain class of traffic pairs. By injecting more specific leaf node prefixes, it will allow the spine nodes to attract more traffic on some underutilized links.

3.5.7. Other End-to-End Services

Losing the topology information will have an impact on some of the end-to-end network services, for instance, MPLS TE or end-to-end segment routing. Some other mechanisms such as those described in PCE [RFC4655] based solution may be used. In this Spine-Leaf extension, the role of the leaf node is not too much different from

the multi-level IS-IS routing while the level-1 IS-IS nodes only have the default route information towards the node which has the Attach Bit (ATT) set, and the level-2 backbone does not have any topology information of the level-1 areas. The exact mechanism to enable certain end-to-end network services in Spine-Leaf network is outside the scope of this document.

3.5.8. Address Family and Topology

IPv6 Address families[RFC5308], Multi-Topology (MT)[RFC5120] and Multi-Instance (MI)[RFC8202] information is carried over the IIH PDU. Since the goal is to simplify the operation of IS-IS network, for the simplicity of this extension, the Spine-Leaf mechanism is applied the same way to all the address families, MTs and MIs.

3.5.9. Migration

For this extension to be deployed in existing networks, a simple migration scheme is needed. To support any leaf node in the network, all the involved spine nodes have to be upgraded first. So the first step is to migrate all the involved spine nodes to support this extension, then the leaf nodes can be enabled with 'Leaf-Mode' one by one. No flag day is needed for the extension migration.

4. IANA Considerations

A new TLV codepoint is defined in this document and needs to be assigned by IANA from the "IS-IS TLV Codepoints" registry. It is referred to as the Spine-Leaf TLV and the suggested value is 150. This TLV is only to be optionally inserted either in the IIH PDU or in the Circuit Flooding Scoped LSP PDU. IANA is also requested to maintain the SL-flag bit values in this TLV, and 0x01, 0x02 and 0x04 bits are defined in this document.

Value	Name	IIH	LSP	SNP	Purge	CS-LSP
150	Spine-Leaf	y	y	n	n	y

This extension also proposes to have the Dynamic Hostname TLV, already assigned as code 137, to be allowed in IIH PDU.

Value	Name	IIH	LSP	SNP	Purge
137	Dynamic Name	y	y	n	y

Two new sub-TLVs are defined in this document and needs to be added assigned by IANA from the "IS-IS TLV Codepoints". They are referred

to in this document as the Leaf-Set sub-TLV and the Info-Req sub-TLV. It is suggested to have the values 1 and 2 respectively.

5. Security Considerations

Security concerns for IS-IS are addressed in [ISO10589], [RFC5304], [RFC5310], and [RFC7602]. This extension does not raise additional security issues.

6. Acknowledgments

TBD.

7. Document Change Log

7.1. Changes to draft-shen-isis-spine-leaf-ext-05.txt

- Submitted January 2018.
- Just a refresh.

7.2. Changes to draft-shen-isis-spine-leaf-ext-04.txt

- Submitted June 2017.
- Added the Tier level information to handle the multi-level spine-leaf topology using this extension.

7.3. Changes to draft-shen-isis-spine-leaf-ext-03.txt

- Submitted March 2017.
- Added the Spine-Leaf sub-TLVs to handle the case of data center pure CLOS topology and mechanism.
- Added the Spine-Leaf TLV and sub-TLVs can be optionally inserted in either IIH PDU or CS-LSP PDU.
- Allow use of prefix Reachability TLVs 135 and 236 in IIHs/CS-LSPs sent from spine to leaf.

7.4. Changes to draft-shen-isis-spine-leaf-ext-02.txt

- Submitted October 2016.
- Removed the ’Default Route Metric’ field in the Spine-Leaf TLV and changed to using the IS-IS Reverse Metric in IIH.

7.5. Changes to draft-shen-isis-spine-leaf-ext-01.txt

- Submitted April 2016.
- No change. Refresh the draft version.

7.6. Changes to draft-shen-isis-spine-leaf-ext-00.txt

- Initial version of the draft is published in November 2015.

8. References

8.1. 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.

[REVERSE-METRIC] Shen, N., Amante, S., and M. Abrahamsson, "IS-IS Routing with Reverse Metric", draft-ietf-isis-reverse-metric-07 (work in progress), 2017.

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

[RFC5120] Przygienda, T., Shen, N., and N. Sheth, "M-ISIS: Multi Topology (MT) Routing in Intermediate System to Intermediate Systems (IS-ISs)", RFC 5120, DOI 10.17487/RFC5120, February 2008, <https://www.rfc-editor.org/info/rfc5120>.

[RFC5301] McPherson, D. and N. Shen, "Dynamic Hostname Exchange Mechanism for IS-IS", RFC 5301, DOI 10.17487/RFC5301, October 2008, <https://www.rfc-editor.org/info/rfc5301>.

[RFC5304] Li, T. and R. Atkinson, "IS-IS Cryptographic Authentication", RFC 5304, DOI 10.17487/RFC5304, October 2008, <https://www.rfc-editor.org/info/rfc5304>.

[RFC5305] Li, T. and H. Smit, "IS-IS Extensions for Traffic Engineering", RFC 5305, DOI 10.17487/RFC5305, October 2008, <https://www.rfc-editor.org/info/rfc5305>.

[RFC5306] Shand, M. and L. Ginsberg, "Restart Signaling for IS-IS", RFC 5306, DOI 10.17487/RFC5306, October 2008, <https://www.rfc-editor.org/info/rfc5306>.

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

[RFC5310] Bhatia, M., Manral, V., Li, T., Atkinson, R., White, R., and M. Fanto, "IS-IS Generic Cryptographic Authentication", RFC 5310, DOI 10.17487/RFC5310, February 2009, <https://www.rfc-editor.org/info/rfc5310>.

[RFC7356] Ginsberg, L., Previdi, S., and Y. Yang, "IS-IS Flooding Scope Link State PDUs (LSPs)", RFC 7356, DOI 10.17487/RFC7356, September 2014, <https://www.rfc-editor.org/info/rfc7356>.

[RFC7602] Chunduri, U., Lu, W., Tian, A., and N. Shen, "IS-IS Extended Sequence Number TLV", RFC 7602, DOI 10.17487/RFC7602, July 2015, <https://www.rfc-editor.org/info/rfc7602>.

[RFC8202] Ginsberg, L., Previdi, S., and W. Henderickx, "IS-IS Multi-Instance", RFC 8202, DOI 10.17487/RFC8202, June 2017, <https://www.rfc-editor.org/info/rfc8202>.

8.2. Informative References

[OPENFABRIC] White, R. and S. Zandi, "Openfabric", draft-white-openfabric-04 (work in progress), October 2017.

[RFC4655] Farrel, A., Vasseur, J., and J. Ash, "A Path Computation Element (PCE)-Based Architecture", RFC 4655, DOI 10.17487/RFC4655, August 2006, <https://www.rfc-editor.org/info/rfc4655>.

[RFC5309] Shen, N., Ed. and A. Zinin, Ed., "Point-to-Point Operation over LAN in Link State Routing Protocols", RFC 5309, DOI 10.17487/RFC5309, October 2008, <https://www.rfc-editor.org/info/rfc5309>.

Authors' Addresses

Naiming Shen
Cisco Systems
560 McCarthy Blvd.
Milpitas, CA 95035
US

Email: naiming@cisco.com

Les Ginsberg
Cisco Systems
821 Alder Drive
Milpitas, CA 95035
US

Email: ginsberg@cisco.com

Sanjay Thyamagundalu

Email: tsanjay@gmail.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: July 21, 2017

V. Talwar
J. Kolhe
A. Shaikh
Google
J. George
Cisco

January 17, 2017

Use cases for gRPC in network management
draft-talwar-rtgwg-grpc-use-cases-01

Abstract

gRPC is an open, high-performance RPC framework designed for efficient low-latency cross-service communications. This document describes use cases for gRPC in network management and other services, particularly streaming telemetry.

Status of This Memo

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

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

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

This Internet-Draft will expire on July 21, 2017.

Table of Contents

1. Introduction 2
2. gRPC use cases 3
2.1. Network management 3
2.1.1. Streaming telemetry motivation and overview 3
2.1.2. Streaming telemetry with gRPC 5
2.1.3. Network configuration management 5
2.2. Additional use cases 6
2.2.1. Client Libraries for connecting polyglot systems . . 6
2.2.2. MicroServices 6
2.2.3. Browser and mobile applications communicating to gRPC Services 6
2.2.4. High performance access to Cloud Services 6
2.2.5. Secure and low overhead communications in embedded systems 6
2.2.6. Unified inter-process and remote communication . . . 7
3. Security Considerations 7
4. IANA Considerations 7
5. References 7
5.1. Normative references 7
5.2. Informative references 8
Appendix A. Change summary 9
A.1. Changes between revisions -00 and -01 9
Authors' Addresses 9

1. Introduction

gRPC is a high performance universal RPC framework to connect distributed systems [GRPC-WWW]. gRPC emerged from an internal Google framework called Stubby which has been used to connect large numbers microservices running within and across data centers for over a decade. Having a uniform, cross-platform RPC infrastructure allowed Google to deploy fleet-wide improvements in efficiency, security, reliability and behavioral analysis critical to supporting the incredible growth of these services. gRPC is the next generation of Stubby, built in the open originally for services, as well as last mile computing use cases like mobile, browser, IOT [GRPC-DESIGN]. It is based on standards like HTTP/2 [RFC7540] and is extensible and pluggable by design.

This document describes use cases for the gRPC protocol [I-D.kumar-rtgwg-grpc-protocol] in network management, including monitoring, configuration management and programmatic operations. We

also summarize a number of additional use cases where gRPC is currently being applied.

2. gRPC use cases

2.1. Network management

Below we discuss several gRPC applications related to network management with a focus on monitoring and telemetry. gRPC is already implemented by several network device vendors as a primary transport for monitoring data based on the streaming telemetry paradigm.

2.1.1. Streaming telemetry motivation and overview

Network operations depend fundamentally on the availability of accurate, near real-time data to drive a variety of management systems, including traffic control systems, fault recovery systems, and demand and capacity forecasting systems. This data consists of information about the control plane (e.g., protocol operations), management plane (e.g., system availability, statistics, and counters), and data plane (e.g., packet and flow statistics).

In addition to the variety of data, the volume of monitoring and management data continues to increase significantly. Modern, high-density platforms with thousands of interfaces and numerous hardware and software modules means potentially collecting millions of objects and running tens of thousands of CLI commands every few minutes in a large-scale network. Network monitoring data is increasingly used to manage mission-critical systems such as real-time monitoring, centralized traffic engineering, server selection and load balancing. Hence it requires efficient, secure, and scalable mechanisms for data transport, encoding, and control.

Most networks rely on traditional management protocols such as SNMP [RFC1157] [RFC3410] for collecting monitoring data about the control and management planes, and SFLOW [RFC3176] or IPFIX [RFC7011] for the data plane. For control and management data in particular, SNMP is the primary tool, despite limitations which make it ill-suited for modern, large-scale networks, especially Web- and Internet-scale backbones, and large, high-capacity data center networks.

While SNMP is widely deployed and implemented in a variety of network environment, it suffers from a number of drawbacks:

- legacy implementations -- designed for devices with limited memory and little processing power; e.g., SNMPv2 supports multiple data items in a message, but is not optimized for high-volume data collection

- lack of discoverability -- discovering new elements requires walking the SNMP MIB periodically; on high-density platforms this is extremely computationally expensive

- lack of capability advertisements -- each object ID must be checked to know whether it is supported by the target platform

- rigid data structures -- whether using standard or vendor proprietary MIBs, the structure and format of the data cannot be easily extended or augmented

To address these drawbacks, a number of network operators proposed a new approach for network monitoring based on streaming telemetry (see an early proposal in [I-D.swhyte-i2rs-data-collection-system]). Streaming telemetry is based on a pub/sub push model in which target devices send data of interest over a streaming channel to a data collection system.

Some notable features of a streaming telemetry system include:

- targets stream data continuously based on a specified period (or as frequently as the target supports), or on a state change

- data is sent as soon as it is available, reducing the need to buffer, or to handle a single large for all data at once

- data may be sent incrementally, e.g., only for those data items that have changed

- ability to distribute the telemetry sources (e.g., directly to linecards) to avoid burdening the management CPU

- users issue subscription requests via RPC to the target to request only the data of interest

- data is exported in a well-structured, common format, e.g., based on YANG models of operational state data [I-D.openconfig-netmod-opstate]

- the target and collector communicate over a secure, authenticated, reliable channel that is long-lived and efficient

Streaming telemetry allows the network behavior to be observed through a time-series data stream. This is in contrast to the polling mechanism used in SNMP in which a monitoring client must periodically request the set of desired data, and walk the MIB to discover changes. The polling frequency is limited since the device

must be able to handle large requests for all interface or QoS counters, for example.

Open source implementations of streaming telemetry are currently being developed by several network vendors, including adapters to deliver data into time-series databases, messaging systems, and data visualization systems [ST-CISCO] [ST-JUNIPER] [ST-ARISTA].

2.1.2. Streaming telemetry with gRPC

gRPC provides a number of capabilities that makes it well-suited for network telemetry. Since its underlying transport is based on HTTP/2, it can exploit several key features:

- binary framing and header compression -- highly efficient encoding on the wire to enable bulk data transfer
- bidirectional streaming RPCs -- the target and collector can stream their data independently, and leverage application-level flow control
- flexible data encoding -- gRPC is payload agnostic, and can be used to transfer data encoded as XML, JSON, protocol buffer, or Thrift; as new data formats and encodings emerge for network data, the RPC layer can be easily adapted
- multi-language support -- open source gRPC IDLs are available for 10 programming languages, and service endpoints can be created on a number of operating systems, giving device vendors flexibility in implementation

gRPC-based telemetry stacks are now being implemented with some available as open source [ST-ARISTA]. A protocol specification for streaming telemetry based on gRPC is also available [GNMI-SPEC].

2.1.3. Network configuration management

gRPC offers a non-proprietary, modern alternative to vendor-specific configuration protocols or standards such as NETCONF [RFC6241] or TL1 [TL1]. Some of the benefits of using gRPC for configuration management include more flexible data encodings (e.g., no requirement to use XML), easier integration based on the large number of language implementations available, and more options for securing connections.

Several platforms now support gRPC configuration protocols using data based on YANG models [GRPC-CISCO] [GRPC-JUNIPER].

2.2. Additional use cases

2.2.1. Client Libraries for connecting polyglot systems

gRPC generates client libraries in 10 languages and thus allows developers to operate in their language of choice and system to communicate with any other system. These libraries offer idiomatic-to-language API surface such that every developer feels they are in their language native environment.

2.2.2. MicroServices

Designed as a general, high performance protocol to interconnect polyglot systems, gRPC is ideal for microservices communication, independent of where the services are deployed. A protocol that offers flow control, bidirectional streaming and a very compact serialization mechanism is ideally suited for connecting microservices at scale. It is already being adopted by large organizations like Square and Netflix for their microservices communications.

2.2.3. Browser and mobile applications communicating to gRPC Services

Mobile and Browser applications are becoming feature rich and more demanding by the day. User expectation is that apps are performant in various network conditions and drain minimal battery and computing power of device. gRPC provides native iOS and Android Java libraries for more efficient communication for applications with backend services such that battery, data are efficiently used and developers have more control of communication with servers using gRPC APIs.

2.2.4. High performance access to Cloud Services

The expectations from high request-rate cloud services like storage and pub/sub messaging systems are to be very efficient and low cost from a compute and networking point of view. Hence, gRPC based APIs are being used for services like Google Cloud BigTable and Google Cloud PubSub. External products like etcd (underlying storage system for kubernetes) also relies on gRPC.

2.2.5. Secure and low overhead communications in embedded systems

With its integrated authentication model and a IDL like nano-protobuf, gRPC could be ideal for secure device-to-device and device-to-cloud communication as well. This use case is still under development.

2.2.6. Unified inter-process and remote communication

gRPC can provide a unified programming model for both inter-process communication and remote service communication. This use case is still under development.

3. Security Considerations

As applied to network configuration and monitoring, any transport protocol and RPC framework must have support for secure, authenticated communication. gRPC supports a number of security mechanisms that are suitable for use in network management, including TLS-based transport, and client and server authentication. These will be detailed further in subsequent drafts.

4. IANA Considerations

None at this time. In the future, there may be proposals to designate specific application ports for gRPC-based telemetry and configuration traffic.

5. References

5.1. Normative references

[RFC1157] Case, J., Fedor, M., Schoffstall, M., and J. Davin, "Simple Network Management Protocol (SNMP)", RFC 1157, DOI 10.17487/RFC1157, May 1990, <http://www.rfc-editor.org/info/rfc1157>.

[RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, DOI 10.17487/RFC3410, December 2002, <http://www.rfc-editor.org/info/rfc3410>.

[RFC3176] Phaal, P., Panchen, S., and N. McKee, "InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks", RFC 3176, DOI 10.17487/RFC3176, September 2001, <http://www.rfc-editor.org/info/rfc3176>.

[RFC7011] Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, DOI 10.17487/RFC7011, September 2013, <http://www.rfc-editor.org/info/rfc7011>.

[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>.

[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>.

5.2. Informative references

[GRPC-DESIGN]
Ryan, L., "gRPC Motivation and Design Principles", September 2015, <http://www.grpc.io/posts/principles>.

[GRPC-WWW]
"gRPC Web site (grpc.io)", September 2015, <http://www.grpc.io>.

[ST-ARISTA]
"Arista Networks goarista GitHub repository", July 2016, <https://github.com/aristanetworks/goarista>.

[ST-CISCO]
"Cisco Systems bigmuddy GitHub repository", April 2016, <https://github.com/cisco/bigmuddy-network-telemetry-stacks>.

[GRPC-CISCO]
"Cisco Systems grpc GitHub repository", February 2016, <https://github.com/CiscoDevNet/grpc-getting-started>.

[ST-JUNIPER]
"Juniper Networks open-nti GitHub repository", July 2016, <https://github.com/Juniper/open-nti>.

[GRPC-JUNIPER]
Juniper Networks, "Next-Generation Network Configuration and Management", May 2016, <https://www.juniper.net/assets/us/en/local/pdf/whitepapers/2000632-en.pdf>.

[GNMI-SPEC]
Borman, P., Hines, M., Lebsack, C., Morrow, C., Shaikh, A., and R. Shakir, "gRPC Network Management Interface", November 2016, <https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-specification.md>.

[TL1] Telcordia, "GR-831-CORE - Operations Application Messages - Language for Operations Application Messages", November 1996, <http://telecom-info.telcordia.com/site-cgi/ido/docs.cgi?ID=SEARCH&DOCUMENT=GR-831&>.

[I-D.kumar-rtgwg-grpc-protocol]
Kumar, A., Kolhe, J., Ghemawat, S., and L. Ryan, "gRPC Protocol", kumar-rtgwg-grpc-protocol-00 (work in progress), July 2016.

[I-D.swhyte-i2rs-data-collection-system]
Whyte, S., Hines, M., and W. Kumari, "Bulk Network Data Collection System", draft-swhyte-i2rs-data-collection-system-00 (work in progress), October 2013.

[I-D.openconfig-netmod-opstate]
Shakir, R., Shaikh, A., and M. Hines, "Consistent Modeling of Operational State Data in YANG", draft-openconfig-netmod-opstate-01 (work in progress), July 2015.

Appendix A. Change summary

A.1. Changes between revisions -00 and -01

- Added reference to gRPC Network Management Interface specification.
- Updated author contact information.

Authors' Addresses

Varun Talwar
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: varuntalwar@google.com

Jayant Kolhe
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: jkolhe@google.com

Anees Shaikh
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: aashaikh@google.com

Joshua George
Cisco
170 W Tasman Dr
San Jose, CA 95134
US

Email: jgeorge@cisco.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 3, 2018

F. Templin, Ed.
Boeing Research & Technology
G. Dawra
A. Lindem
Cisco Systems, Inc.
October 30, 2017

A Simple BGP-based Mobile Routing System for the Aeronautical Telecommunications Network
draft-templin-atn-bgp-04.txt

Abstract

The International Civil Aviation Organization (ICAO) is investigating mobile routing solutions for a worldwide Aeronautical Telecommunications Network with Internet Protocol Services (ATN/IPS). The ATN/IPS will eventually replace existing communication services with an IPv6-based service supporting pervasive Air Traffic Management (ATM) for Air Traffic Controllers (ATC), Airline Operations Controllers (AOC), and all commercial aircraft worldwide. This informational document describes a simple mobile routing service based on mature industry standards to address the ATN/IPS requirements.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 3, 2018.

Copyright Notice

Table of Contents

1. Introduction 2
2. Proposed BGP-based ATN/IPS Routing System 4
3. Route Optimization 8
4. Route Availability 10
5. BGP Protocol Considerations 11
6. Implementation Status 12
7. IANA Considerations 12
8. Security Considerations 12
9. Acknowledgements 12
10. References 12
10.1. Normative References 13
10.2. Informative References 13
Authors' Addresses 15

1. Introduction

The International Civil Aviation Organization [ICAO] is investigating mobile routing solutions for a worldwide Aeronautical Telecommunications Network with Internet Protocol Services (ATN/IPS). The ATN/IPS will eventually replace existing communication services with an IPv6-based service supporting pervasive Air Traffic Management (ATM) for Air Traffic Controllers (ATC), Airline Operations Controllers (AOC), and all commercial aircraft worldwide. This informational document describes a simple mobile routing service based on mature industry standards to address the ATN/IPS requirements.

Aircraft communicate via wireless aviation data links that typically support much lower data rates than terrestrial wireless and wired-line communications. For example, VHF-based data links only support data rates on the order of 32Kbps and an emerging L-Band data link that is expected to play a key role in future aeronautical communications only supports rates on the order of 1Mbps. Although satellite data links can provide much higher data rates during optimal conditions, they (like all other aviation data links) are subject to errors, delay, disruption, signal intermittence,

degradation due to atmospheric conditions, etc. The well-connected ground domain ATN/IPS network should therefore treat each safety-of-flight critical packet produced by (or destined to) an aircraft as a precious commodity and strive for a "better-than-best-effort" service that provides the highest possible degree of reliability.

The ATN/IPS assumes a worldwide connected Internetwork for carrying ATM communications. The Internetwork could be manifested as a private collection of long-haul backbone links (e.g., fiberoptics, copper, SATCOM, etc.) interconnected by high-performance networking gear such as bridges, switches and routers. Such a private Internetwork would need to connect all ATN/IPS participants worldwide, and could therefore present a considerable cost for a large-scale deployment of new infrastructure. Alternatively, the ATN/IPS could be deployed as an overlay over the existing global public Internet itself as long as sufficient security and reliability provisions are met. For example, ATN/IPS nodes could be deployed as part of an SD-WAN or an MPLS-WAN that rides over the public Internet via secured tunnels.

The ATN/IPS further assumes that each aircraft will receive an IPv6 Mobile Network Prefix (MNP) that accompanies the aircraft wherever it travels. ATCs and AOCs will likewise receive IPv6 prefixes, but they would typically appear in static (not mobile) deployments. Throughout the rest of this document, we therefore use the term "MNP" when discussing an IPv6 prefix that is delegated to any ATN/IPS end system, including ATCs, AOCs and aircraft. We also use the term Mobility Service Prefix (MSP) to refer to an aggregated prefix assigned to the ATN/IPS by an Internet assigned numbers authority, and from which all MNPs are delegated (e.g., up to 2**32 IPv6 /64 MNPs could be delegated from the MSP 2001:db8::/32).

[CBB] describes an aviation mobile routing service based on dynamic updates in the global public Internet Border Gateway Protocol (BGP) [RFC4271] routing system. Practical experience with the approach has shown that frequent injections and withdrawals of MNPs in the Internet routing system results in excessive BGP update messaging, slow routing table convergence times, and extended outages when no route is available. This is due to both conservative default BGP protocol timing parameters (see Section 5) and the complex peering interconnections of BGP routers within the global Internet infrastructure. The situation is further exacerbated by frequent aircraft mobility events that each result in BGP updates that must be propagated to all BGP routers in the Internet that carry a full routing table.

We therefore consider an approach using a BGP overlay network routing system where a private BGP routing protocol instance is maintained

between ATN/IPS Autonomous System (AS) Border Routers (ASBRs). The private BGP instance does not interact with the Internetwork BGP routing system, and BGP updates are unidirectional from "stub" ASBRs (s-ASBRs) to a very small set of "core" ASBRs (c-ASBRs) in a hub-and-spokes arrangement. For the AERO proposal [I-D.templin-aerolink], the s-ASBRs correspond to AERO Servers. For the LISP proposal [I-D.ietf-lisp-rfc6830bis], the s-ASBRs correspond to xTRs that connect directly to the BGP system instead of via Map Servers and Resolvers. No non-standard extensions of the BGP protocol are necessary.

The s-ASBRs for each stub AS connect to a small number of c-ASBRs via dedicated high speed links and/or tunnels across the Internetwork using industry-standard encapsulations (e.g., Generic Routing Encapsulation (GRE) [RFC2784], IPsec [RFC4301] etc.). The s-ASBRs engage in external BGP (eBGP) peerings with their respective c-ASBRs, and only maintain routing table entries for the MNPs currently active within the stub AS. A stub AS may connect to the core via multiple s-ASBRs, in which case the s-ASBRs would engage in an Interior Gateway Protocol (IGP) among themselves to maintain a common view of the stub AS MNPs. (The s-ASBRs need not engage in internal BGP (iBGP) peerings, since they do not receive any BGP updates from c-ASBRs and therefore have no BGP information to share with each other.) Finally, the s-ASBRs also maintain default routes with their c-ASBRs as the next hop, and therefore hold only partial topology information.

The c-ASBRs connect to other c-ASBRs using iBGP peerings over which they collaboratively maintain a full routing table for all active MNPs currently in service. Therefore, only the c-ASBRs maintain a full BGP routing table and never send any BGP updates to s-ASBRs. This simple arrangement therefore greatly reduces the number of BGP updates that need to be synchronized among peers, and the number is reduced further still when localized mobility events within stub ASes (i.e., "intradomain" mobility events) are mitigated within the AS instead of being propagated to the core.

The following section provides a detailed discussion of the proposed BGP-based ATN/IPS routing system.

2. Proposed BGP-based ATN/IPS Routing System

The proposed ATN/IPS routing system comprises a private BGP instance coordinated between ASBRs in an overlay network. The overlay does not interact with the native Internetwork BGP routing system, and each c-ASBR advertises only a small and unchanging set of MSPs into the Internetwork instead of the full dynamically changing set of MNPs. The system corresponds to the framework first specified by the

LISP+ALT proposal [RFC6836] and later also adopted by the AERO proposal [I-D.templin-aerolink]. The system differs from the LISP Delegated Database Tree (DDT) proposal [I-D.ietf-lisp-ddt] that is designed with scalability as the primary consideration.

In a reference deployment, one or more s-ASBRs connect each stub AS to the overlay using a shared stub AS Number (ASN). Each s-ASBR further uses eBGP to peer with one or more c-ASBRs. All c-ASBRs are members of the same core AS, and use a shared core ASN. The c-ASBRs further use iBGP to maintain a synchronized consistent view of all active MNPs currently in service. Figure 1 below represents the reference deployment. Note that in the figure only two s-ASBRs show detail, but similar arrangements are implied for all other s-ASBRs. Note also that each stub AS shows only a single s-ASBR with a single c-ASBR connection, but in practical deployments each stub AS may have multiple s-ASBRs that peer with multiple c-ASBRs via eBGP, e.g., for fault tolerance.

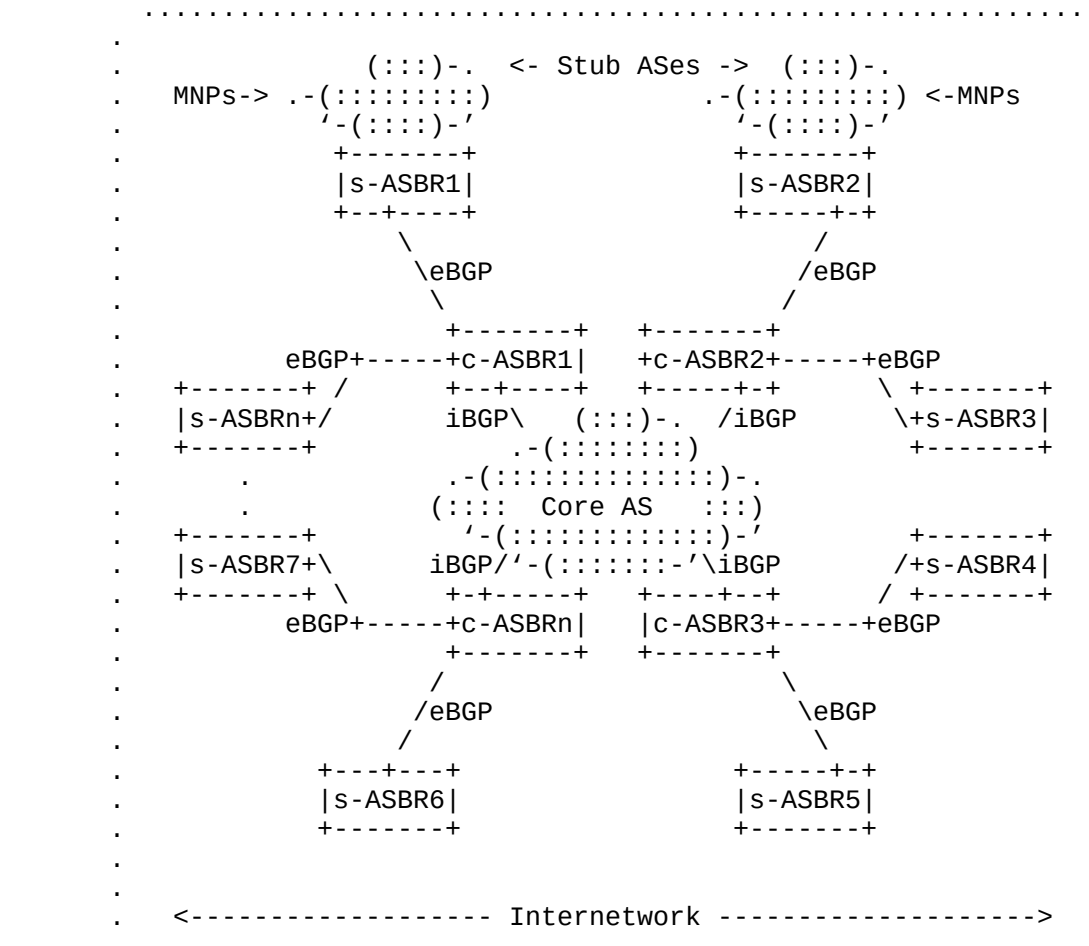


Figure 1: Reference Deployment

In the reference deployment, each s-ASBR maintains routes for active MNPs that currently belong to its stub AS, and dynamically announces new MNPs and withdraws departed MNPs in its eBGP updates to c-ASBRs in response to "interdomain" mobility events. Since ATN/IPS end systems are expected to remain within the same stub AS for extended timeframes, however, intradomain mobility events (such as an aircraft handing off between cell towers) are handled locally within the stub AS instead of being propagated as interdomain eBGP updates.

Each c-ASBR configures a black-hole route for each of its MSPs. By black-holing the MSPs, the c-ASBR will maintain forwarding table entries only for the MNPs that are currently active, and packets

destined to all other MNPs will correctly incur ICMPv6 Destination Unreachable messages [RFC4443] due to the black hole route. The c-ASBRs do not send eBGP updates for MNPs to s-ASBRs, but instead originate a default route. In this way, s-ASBRs have only partial topology knowledge (i.e., they know only about the active MNPs currently within their stub ASes) and they forward all other packets to c-ASBRs which have full topology knowledge.

Scaling properties of this ATN/IPS routing system are limited by the number of BGP routes that can be carried by the c-ASBRs. A 2015 study showed that BGP routers in the global public Internet at that time carried more than 500K routes with linear growth and no signs of router resource exhaustion [BGP]. A more recent network emulation study also showed that a single c-ASBR can accommodate at least 1M dynamically changing BGP routes even on a lightweight virtual machine, with the expectation that high-performance dedicated router hardware can support even more.

Therefore, assuming each c-ASBR can carry 1M or more routes, this means that at least 1M ATN/IPS end system MNPs can be serviced by a single set of c-ASBRs. A means of increasing scaling would be to assign a different set of c-ASBRs for each set of MSPs. In that case, each s-ASBR still peers with one or more c-ASBRs from each set of c-ASBRs, but the s-ASBR institutes route filters so that it only sends BGP updates to the specific set of c-ASBRs that aggregate the MSP. For example, if the MSP for the ATN/IPS deployment is 2001:db8::/32, a first set of c-ASBRs could service the MSP segment 2001:db8::/40, a second set could service 2001:db8:0100::/40, a third set could service 2001:db8:0200::/40, etc.

Assuming a sufficient number of c-ASBR sets, the ATN/IPS routing system can then accommodate 1B or more MNPs. In this way, each set of c-ASBRs services a specific set of MSPs that they advertise to the native Internetwork routing system, and each s-ASBR configures MSP-specific routes that list the correct set of c-ASBRs as next hops. This arrangement also allows for natural incremental deployment, and can support small scale initial deployments followed by dynamic deployment of additional ATN/IPS infrastructure elements without disturbing the already-deployed base.

Finally, c-ASBRs may have multiple routing table entries for a single MNP advertised by multiple s-ASBRs. Each s-ASBR can be assigned a MULTI_EXIT_DISC (MED) metric for routes that it originates in its eBGP peering configurations [RFC4451] so that c-ASBRs can determine preferences for MNPs learned from multiple s-ASBRs. In this way, c-ASBRs can select the neighboring s-ASBR with the lowest MED value, i.e., even if it is not on the shortest path. The c-ASBR can then fail over to a s-ASBR with a larger MED value in case of MNP

withdrawal or s-ASBR failure. Such an event could correspond to an aviation data link handover, e.g., when an aircraft switches over from a satellite link to an L-Band link.

3. Route Optimization

ATN/IPS end systems will frequently need to communicate with correspondents located in other stub ASes. In the ASBR peering arrangement discussed in Section 2, this can initially only be accommodated by having the source s-ASBR forward packets to a c-ASBR which then forwards the packets toward the destination s-ASBR where the destination ATN/IPS end system resides. In many cases, it would be desirable to eliminate c-ASBRs from this "dogleg" route so that the source s-ASBR can send packets directly to the destination s-ASBR through tunneling across the Internetwork. This can be accomplished using a mapping resolution service such as proposed in AERO [I-D.templin-aerolink] or LISP [I-D.ietf-lisp-rfc6830bis][I-D.ietf-lisp-rfc6833bis]. Employment of the mapping resolution service results in a condition known as route optimization.

A route optimization example is shown in Figure 2 and Figure 3 below. In the first figure, the dogleg route between correspondents in the stub ASes traverses the path from s-ASBR1 to c-ASBR1 to c-ASBR2 to S-ASBR2. In the second figure, the optimized route goes directly from s-ASBR1 to s-ASBR2, i.e., the c-ASBRs are not included in the path.

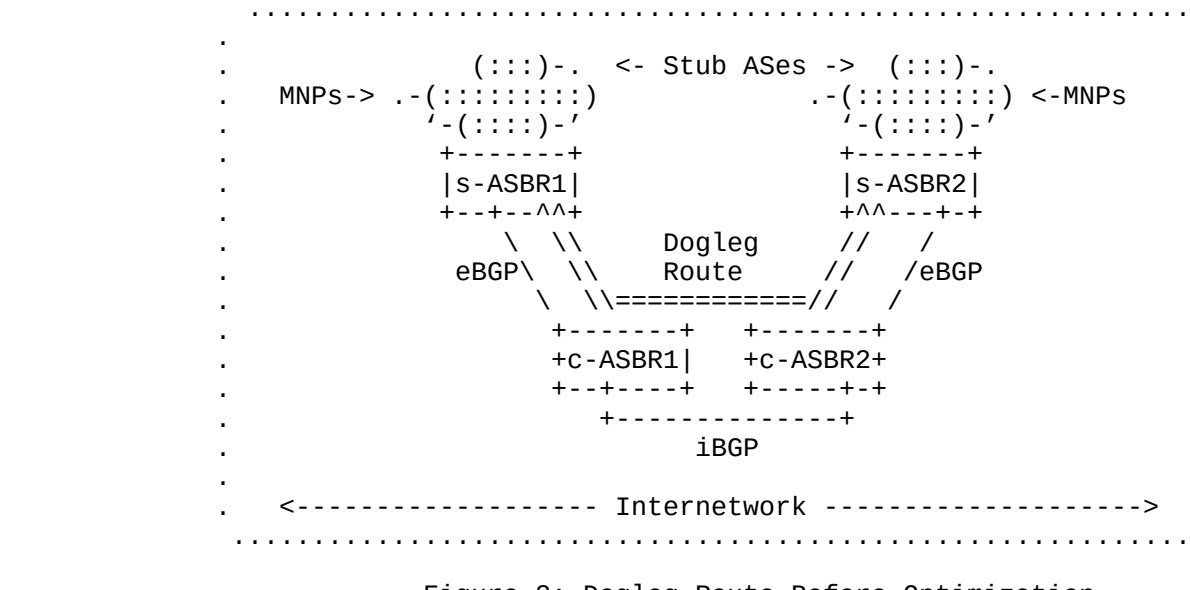


Figure 2: Dogleg Route Before Optimization

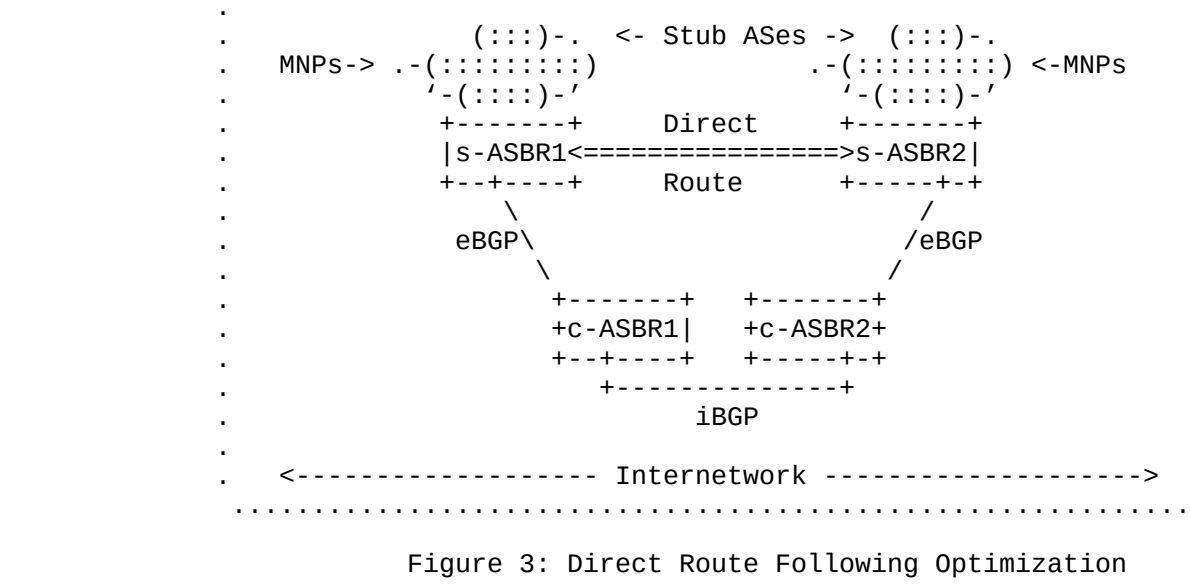


Figure 3: Direct Route Following Optimization

Note that route optimization can fail if the source s-ASBR cannot tunnel packets directly to the destination s-ASBR due to some form of Internetwork blockage such as filtering middleboxes. It is also necessary for the source s-ASBR to quickly detect and adjust to

failure of the destination s-ASBR and/or movement of the destination MNP. In both of these cases, significant packet loss could occur before the source s-ASBR can detect that the destination MNP is no longer reachable via the route-optimized path. This implies that route optimized paths may not always be the best choice for carrying safety-of-flight critical packets with high reliability requirements.

In all cases, s-ASBRs do not advertise MNPs discovered via route optimization to c-ASBRs. Instead, s-ASBRs keep MNPs discovered via route optimization in a local table that is kept separate from the MNPs of ATN/IPS end systems within their own stub AS.

4. Route Availability

In the ATN/IPS BGP-based routing system proposed in this document, each s-ASBR always has a default route and can therefore always send packets via the dogleg route through a c-ASBR even if a route optimized path has been established. The direct paths between s-ASBRs and c-ASBRs are maintained by BGP peering session keepalives such that, if a link or an ASBR goes down, BGP will detect the failure and readjust the routing tables. However, ASBRs and the links that interconnect them are expected to be secured as highly-available and fault tolerant critical infrastructure such that peering session failures should be extremely rare.

This represents a distinct architectural difference from other approaches that only operate over route optimized paths. With the approach described herein the source s-ASBR will always have a working route, even if only via a dogleg path through a c-ASBR. This gives rise to the possibility of sending {high-priority, low-data-rate} packets via the assured dogleg route and {low-priority, high-data-rate} packets via the optimized route, e.g., based on per-packet quality of service indications. This could also give rise to a fair pricing model that would charge more for use of the high-assurance dogleg path and less for use of the lesser-assured route-optimized path.

This distinction is important to aviation networking, where isolated safety-of-flight critical packets such as produced by the Controller Pilot Data Link Communications (CPDLC) facility may not be eligible for retransmission, e.g., if an aviation data link is failing. If there is no route available, the packet can be dropped or delayed and safety-of-flight parameters could be lost. Even when an optimized route is discovered on-demand, the route may not work and again safety-of-flight critical packets could be lost.

5. BGP Protocol Considerations

The number of eBGP peering sessions that each c-ASBR must service is proportional to the number of s-ASBRs in the system. Network emulations with lightweight virtual machines have shown that a single c-ASBR can service at least 100 eBGP peerings from s-ASBRs that each advertise 10K MNP routes (i.e., 1M total). It is expected that robust c-ASBRs can service many more peerings than this - possibly by multiple orders of magnitude. But even assuming a conservative limit, the number of s-ASBRs could be increased by also increasing the number of c-ASBRs. Since c-ASBRs also peer with each other using iBGP, however, larger-scale c-ASBR deployments may need to employ an adjunct facility such as BGP route reflectors [RFC4456].

The number of aircraft in operation at a given time wordlwide is likely to be significantly less than 1M, but we will assume this number for a worst-case analysis. Assuming an average 1hour flight profile from gate-to-gate, and 10 data link transitions per flight, the entire system will need to service at most 10M BGP updates per hour (2778 updates per second). This number is within the realm of the peak BGP update messaging seen in the global public Internet today [BGP2]. Assuming a BGP update message size of 100 bytes (800bits), the total amount of BGP control message traffic to a single c-ASBR will be less than 2.5Mbps which is a nominal rate for modern data links.

Industry standard BGP routers provide configurable parameters with conservative default values. For example, the default hold time is 90 seconds, the default keepalive time is 1/3 of the hold time, and the default MinRouteAdvertisementinterval is 30 seconds for eBGP peers and 5 seconds for iBGP peers (see Section 10 of [RFC4271]). For the simple mobile routing system described herein, these parameters can and should be set to more aggressive values to support faster neighbor/link failure detection and faster routing protocol convergence times. For example, a hold time of 3 seconds and a MinRouteAdvertisementinterval of 0 seconds for both iBGP and eBGP.

By default, MED only compares metrics that originate from multiple neighbors within the same AS [RFC4451]. In order to compare MED metrics that come from different ASes, a router configuration file entry may be needed (e.g., Cisco routers require the configuration file entry "bgp always-compare-med"). Furthermore, in order for the MED discriminator to be applied correctly, the AS_PATH phase in the BGP route selection process must be disabled (e.g., Cisco routers use the configuration file entry "bgp bestpath as-path ignore").

6. Implementation Status

The BGP routing arrangement described in this document has been modeled in realistic network emulations showing that the MED process results in selection of the best peer when multiple peers advertise the same MNP. Modeling has also shown that at least 1 million MNPs can be propagated to each c-ASBR even on lightweight virtual machines.

7. IANA Considerations

This document does not introduce any IANA considerations.

8. Security Considerations

ATN/IPS ASBRs on the open Internet are susceptible to the same attack profiles as for any Internet nodes. For this reason, ASBRs should employ physical security and/or IP securing mechanisms such as IPsec [RFC4301], TLS [RFC5246], etc.

ATN/IPS ASBRs present targets for Distributed Denial of Service (DDoS) attacks. This concern is no different than for any node on the open Internet, where attackers could send spoofed packets to the node at high data rates. This can be mitigated by connecting ATN/IPS ASBRs over dedicated links with no connections to the Internet and/or when ASBR connections to the Internet are only permitted through well-managed firewalls.

ATN/IPS s-ASBRs should institute rate limits to protect low data rate aviation data links from receiving DDoS packet floods.

9. Acknowledgements

This work is aligned with the FAA as per the SE2025 contract number DTFAWA-15-D-00030.

This work is aligned with the NASA Safe Autonomous Systems Operation (SASO) program under NASA contract number NNA16BD84C.

This work is aligned with the Boeing Information Technology (BIT) MobileNet program.

10. References

10.1. Normative References

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <https://www.rfc-editor.org/info/rfc2460>.

[RFC4271] Rekhter, Y., Ed., Li, T., Ed., and S. Hares, Ed., "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, DOI 10.17487/RFC4271, January 2006, <https://www.rfc-editor.org/info/rfc4271>.

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <https://www.rfc-editor.org/info/rfc4443>.

[RFC4451] McPherson, D. and V. Gill, "BGP MULTI_EXIT_DISC (MED) Considerations", RFC 4451, DOI 10.17487/RFC4451, March 2006, <https://www.rfc-editor.org/info/rfc4451>.

[RFC4456] Bates, T., Chen, E., and R. Chandra, "BGP Route Reflection: An Alternative to Full Mesh Internal BGP (IBGP)", RFC 4456, DOI 10.17487/RFC4456, April 2006, <https://www.rfc-editor.org/info/rfc4456>.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, DOI 10.17487/RFC4861, September 2007, <https://www.rfc-editor.org/info/rfc4861>.

10.2. Informative References

[BGP] Huston, G., "BGP in 2015, http://potaroo.net", January 2016.

[BGP2] Huston, G., "BGP Instability Report, http://bgpupdates.potaroo.net/instability/bgpupd.html", May 2017.

[CBB] Dul, A., "Global IP Network Mobility using Border Gateway Protocol (BGP), http://www.quark.net/docs/Global_IP_Network_Mobility_using_BGP.pdf", March 2006.

[I-D.ietf-lisp-ddt]
Fuller, V., Lewis, D., Ermagan, V., Jain, A., and A. Smirnov, "LISP Delegated Database Tree", draft-ietf-lisp-ddt-09 (work in progress), January 2017.

[I-D.ietf-lisp-rfc6830bis]
Farinacci, D., Fuller, V., Meyer, D., Lewis, D., and A. Cabellos-Aparicio, "The Locator/ID Separation Protocol (LISP)", draft-ietf-lisp-rfc6830bis-06 (work in progress), October 2017.

[I-D.ietf-lisp-rfc6833bis]
Fuller, V., Farinacci, D., and A. Cabellos-Aparicio, "Locator/ID Separation Protocol (LISP) Control-Plane", draft-ietf-lisp-rfc6833bis-06 (work in progress), October 2017.

[I-D.templin-aerolink]
Templin, F., "Asymmetric Extended Route Optimization (AERO)", draft-templin-aerolink-75 (work in progress), May 2017.

[ICAO] ICAO, I., "http://www.icao.int/Pages/default.aspx", February 2017.

[RFC2784] Farinacci, D., Li, T., Hanks, S., Meyer, D., and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 2784, DOI 10.17487/RFC2784, March 2000, <https://www.rfc-editor.org/info/rfc2784>.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <https://www.rfc-editor.org/info/rfc4301>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC6836] Fuller, V., Farinacci, D., Meyer, D., and D. Lewis, "Locator/ID Separation Protocol Alternative Logical Topology (LISP+ALT)", RFC 6836, DOI 10.17487/RFC6836, January 2013, <https://www.rfc-editor.org/info/rfc6836>.

Authors' Addresses

Fred L. Templin (editor)
Boeing Research & Technology
P.O. Box 3707
Seattle, WA 98124
USA

Email: fltemplin@acm.org

Gaurav Dawra
Cisco Systems, Inc.
USA

Email: gdawra@cisco.com

Acee Lindem
Cisco Systems, Inc.
USA

Email: acee@cisco.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 9, 2017

J. White
D. Black
Dell EMC
J. Leung
Intel Corporation
March 9, 2017

YANG Data Center Baseline Switch Profile
draft-wbl-rtgwg-baseline-switch-model-01

Abstract

[Insert abstract here]

Status of This Memo

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

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

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

This Internet-Draft will expire on September 9, 2017.

Copyright Notice

1. Introduction

Disclaimer - this is a -00 draft.

This is a normative profile for Baseline Switch Profile (send into IETF RTG) intended to be published as RFC on completion of DMTF spec to wrap Baseline Switch Profile.

2. What is a Redfish Baseline Switch?

The baseline switch profile contains basic system, interface, L2, and L3 configuration elements sufficient to set up the device for use in a controller based converged infrastructure environment.

The following list of IETF drafts, RFCs, and Redfish models will constitute the management interface to the baseline switch.

3. Core YANG RFCs

RFC6020 [1] provides the YANG modeling language definition.

RFC6991 [2] provides the Common YANG Data Types used by many other IETF YANG modules.

Interface management requires at set of RFCs to provide all relevant capabilities:

https://tools.ietf.org/html/rfc7223
https://tools.ietf.org/html/rfc7277
https://tools.ietf.org/html/rfc7224
https://tools.ietf.org/html/rfc7317

3.1. RFC7223 provides:

```
     +--rw interfaces
     |  +--rw interface* [name]
     |     +--rw name                        string
     |     +--rw description?                string
     |     +--rw type                        identityref
     |     +--rw enabled?                    boolean
     |     +--rw link-up-down-trap-enable?   enumeration
     +--ro interfaces-state
        +--ro interface* [name]
           +--ro name               string
           +--ro type               identityref
           +--ro admin-status       enumeration
           +--ro oper-status        enumeration
           +--ro last-change?       YANG:date-and-time
           +--ro if-index           int32
           +--ro phys-address?      YANG:phys-address
           +--ro higher-layer-if*   interface-state-ref
           +--ro lower-layer-if*    interface-state-ref
           +--ro speed?             YANG:gauge64
           +--ro statistics
              +--ro discontinuity-time    YANG:date-and-time
              +--ro in-octets?            YANG:counter64
              +--ro in-unicast-pkts?      YANG:counter64
              +--ro in-broadcast-pkts?    YANG:counter64
              +--ro in-multicast-pkts?    YANG:counter64
              +--ro in-discards?          YANG:counter32
              +--ro in-errors?            YANG:counter32
              +--ro in-unknown-protos?    YANG:counter32
              +--ro out-octets?           YANG:counter64
              +--ro out-unicast-pkts?     YANG:counter64
              +--ro out-broadcast-pkts?   YANG:counter64
              +--ro out-multicast-pkts?   YANG:counter64
              +--ro out-discards?         YANG:counter32
              +--ro out-errors?           YANG:counter32
```

3.2. RFC7277 adds:

```
     +--rw if:interfaces
       +--rw if:interface* [name]
          ...
          +--rw ipv4!
          |  +--rw enabled?            boolean
          |  +--rw forwarding?         boolean
          |  +--rw mtu?                uint16
          |  +--rw address* [ip]
          |  |  +--rw ip               inet:ipv4-address-no-zone
          |  |  +--rw (subnet)
          |  |     +--:(prefix-length)
```

```
         |  |     |  +--rw ip:prefix-length?   uint8
         |  |     +--:(netmask)
         |  |        +--rw ip:netmask?         YANG:dotted-quad
         |  +--rw neighbor* [ip]
         |     +--rw ip                    inet:ipv4-address-no-zone
         |     +--rw link-layer-address    YANG:phys-address
         +--rw ipv6!
            +--rw enabled?            boolean
            +--rw forwarding?         boolean
            +--rw mtu?                uint32
            +--rw address* [ip]
            |  +--rw ip               inet:ipv6-address-no-zone
            |  +--rw prefix-length    uint8
            +--rw neighbor* [ip]
            |  +--rw ip                    inet:ipv6-address-no-zone
            |  +--rw link-layer-address    YANG:phys-address
            +--rw dup-addr-detect-transmits?   uint32
            +--rw autoconf
               +--rw create-global-addresses?        boolean
               +--rw create-temporary-addresses?     boolean
               +--rw temporary-valid-lifetime?       uint32
               +--rw temporary-preferred-lifetime?   uint32
```

AND

```
 +--ro if:interfaces-state
    +--ro if:interface* [name]
       ...
       +--ro ipv4!
       |  +--ro forwarding?   boolean
       |  +--ro mtu?          uint16
       |  +--ro address* [ip]
       |  |  +--ro ip               inet:ipv4-address-no-zone
       |  |  +--ro (subnet)?
       |  |  |  +--:(prefix-length)
       |  |  |  |  +--ro prefix-length?   uint8
       |  |  |  +--:(netmask)
       |  |  |     +--ro netmask?         YANG:dotted-quad
       |  |  +--ro origin?          ip-address-origin
       |  +--ro neighbor* [ip]
       |     +--ro ip                    inet:ipv4-address-no-zone
       |     +--ro link-layer-address?   YANG:phys-address
       |     +--ro origin?               neighbor-origin
       +--ro ipv6!
          +--ro forwarding?   boolean
          +--ro mtu?          uint32
          +--ro address* [ip]
          |  +--ro ip               inet:ipv6-address-no-zone
```

```
            |  +--ro prefix-length    uint8
            |  +--ro origin?          ip-address-origin
            |  +--ro status?          enumeration
            +--ro neighbor* [ip]
               +--ro ip                     inet:ipv6-address-no-zone
               +--ro link-layer-address?    YANG:phys-address
               +--ro origin?                neighbor-origin
               +--ro is-router?             empty
               +--ro state?                 enumeration
```

3.3. RFC7224 provides:

The set of YANG identity statement for the IANA defined interface types.

3.4. RFC7317 provides:

- System Identification
- System Time Date
- NTP
- DNS Client

System Identification

```
  +--rw system
  |  +--rw contact?          string
  |  +--rw hostname?         inet:domain-name
  |  +--rw location?         string
  +--ro system-state
     +--ro platform
        +--ro os-name?       string
        +--ro os-release?    string
        +--ro os-version?    string
        +--ro machine?       string
```

System Time

```
   +--rw system
   |  +--rw clock
   |  |  +--rw (timezone)?
   |  |     +--:(timezone-name)
   |  |     |  +--rw timezone-name?     timezone-name
   |  |     +--:(timezone-utc-offset)
   |  |        +--rw timezone-utc-offset?   int16
   |  +--rw ntp!
   |     +--rw enabled?   boolean
   |     +--rw server* [name]
   |        +--rw name                 string
   |        +--rw (transport)
   |        |  +--:(udp)
   |        |     +--rw udp
   |        |        +--rw address    inet:host
   |        |        +--rw port?      inet:port-number
   |        +--rw association-type?   enumeration
   |        +--rw iburst?             boolean
   |        +--rw prefer?             boolean
   +--ro system-state
      +--ro clock
         +--ro current-datetime?      YANG:date-and-time
         +--ro boot-datetime?         YANG:date-and-time
```

DNS Client

```
   +--rw system
      +--rw dns-resolver
         +--rw search*    inet:domain-name
         +--rw server* [name]
         |  +--rw name    string
         |  +--rw (transport)
         |     +--:(udp-and-tcp)
         |        +--udp-and-tcp
         |           +--rw address    inet:ip-address
         |           +--rw port?      inet:port-number
         +--rw options
            +--rw timeout?    uint8
            +--rw attempts?   uint8
```

User Authentication

```
     +--rw system
        +--rw authentication
           +--rw user-authentication-order*   identityref
           +--rw user* [name]
              +--rw name        string
              +--rw password?   ianach:crypt-hash
              +--rw authorized-key* [name]
                 +--rw name         string
                 +--rw algorithm    string
                 +--rw key-data     binary
```

4. Additional YANG models

In addition to the above RFCs, the baseline switch models needs to cover:

- VLANs
- ACLs
- Syslog

The following lists of IETF drafts sets our recommendation to cover the above three areas.

4.1. VLAN and interface extensions:

To handle VLANs and with related interface configuration the following YANG models are under evaluation.

- https://tools.ietf.org/html/draft-ietf-netmod-intf-ext-yang-03
- https://tools.ietf.org/html/draft-wilton-intf-vlan-yang-00.txt ## ACL To handle ACL configuration the following YANG model is under consideration.
- https://tools.ietf.org/html/draft-ietf-netmod-acl-model-09

4.2. Syslog

To handle configuration and access to syslog the following YANG model is under consideration.

- https://tools.ietf.org/html/draft-ietf-netmod-syslog-model-11

5. Applicable Redfish system management models

The following standard Redfish systems management models apply to the baseline network switch profile. Reference: Redfish schema index [3]. The use of these Redfish management models allows a converged infrastructure manager to have a consistent view of server, storage and network systems.

- Chassis
- ComputerSystem
- Manager
- ManagerAccount
- Power
- Thermal
- SoftwareInventory plus UpdateService
- Event configuration using Event, EventDestination, and Event Service
- Access to logs using LogEntry, and LogService
- Management interface configuration using EthernetInterface and related
- Console configuration using SerialInterface
- PrivilegeRegistery and Privileges

Where YANG and Redfish overlap, the commonality of YANG vs Redfish is TBD.

6. Overall Baseline Switch Profile Structure

```
./redfish/v1/Systems
./redfish/v1/Chassis
./redfish/v1/NetworkDevices/BaselineSwitch/...
... other redfish resource blocks...
(resource from RFCs and Redfish bullet list, above)
```

7. References

7.1. Normative References

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

7.2. URIs

[1] https://tools.ietf.org/html/rfc6020

[2] https://tools.ietf.org/html/rfc6991

[3] http://redfish.dmtf.org/redfish/schema_index

Authors' Addresses

Joseph White
Dell EMC

Email: joseph.l.white@dell.com

David Black
Dell EMC

Email: david.black@dell.com

John Leung
Intel Corporation

Email: john.leung@intel.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 9, 2017

J. White
D. Black
Dell EMC
J. Leung
Intel Corporation

March 9, 2017

YANG Baseline Switch Profile Background
draft-wbl-rtgwg-yang-ci-profile-bkgd-01

Abstract

A YANG device profile is primarily a group of YANG models that are appropriate for use with a particular class of device or in specific device roles. This document provides background and describes the rationale for a baseline data center switch device profile, e.g., for top-of-rack switches in data center converged infrastructure. This rationale is based on the reuse of YANG models by the DMTF Redfish standard for management of converged infrastructure, but the resulting YANG device profile is intended to be useable by any YANG-based network management approach or protocol, as opposed to being specific to Redfish.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 9, 2017.

1. Introduction

Disclaimer - this is a -00 draft, whose primary goal is to capture the rationale for use of YANG for Redfish network management and the desirability of a baseline data center network switch profile, including providing technical background on the Redfish standard and its approach to network management. The draft content is not polished, and the organization of the material is likely to change.

A YANG device profile is primarily a group of YANG models that are appropriate for use with a particular class of device or in specific device roles. This document provides background and describes the rationale for a baseline data center switch device profile, e.g., for top-of-rack switches in data center converged infrastructure. The rationale is based on the reuse of YANG models by the DMTF Redfish standard for management of converged infrastructure, but the resulting YANG device profile is intended to be useable by any YANG-based network management approach or protocol; it is not intended to be Redfish-specific.

In support of this rationale, this document provides background on how YANG is used in the Redfish management framework; the YANG modules are translated to Redfish schema definitions in order to enable reuse of the models are with Redfish-defined management protocols and related functionality.

2. Motivation

A common management framework with accompanying set of protocols and device models is desirable in systems management use cases. A good example of this is in a converged infrastructure deployment within a data center. Applications, servers, storage, appliances, and networking elements are assembled to create a combined coherent platform. For the networking components in such an environment, there are platform management elements that are common with other types of systems such as thermal monitoring, physical enclosure, fans, and power supplies, as well as networking specific management elements to control the forwarding and filtering of network packets or the networking services. The common elements should be accessed and managed in a single way across all systems within the deployment.

Management, orchestration, and control of such a system benefits from a single approach that enables unified tools sets and simplifies operations.

The networking specific configuration within the converged infrastructure environment only needs a subset of all the possible networking protocols and services giving the converged controller only the minimum spanning control surface in terms of the models it can access. A breakdown of the needs of such a switch result in about 5-10 YANG modules (see Appendix A). These 5-10 modules should lead to common YANG-based data center network switch management across vendors and products.

As a contrast, a full function edge router would need many more protocols and services along with full function virtualization resulting in the use of about 80 YANG modules.

2.1. Redfish Background

The DMTF (Distributed Management Task Force) Redfish standard is becoming the common standard for converged infrastructure (CI) management. Converged Infrastructure consists of rack-scale (partial or full-rack) integrated compute, network and storage infrastructure that is procured and deployed as rack scale systems.

Redfish data center storage management functionality has been extended in partnership with SNIA (Storage Networking Industry Association) resulting in the recently released Swordfish specification that extends Redfish for networked storage and storage network management. The authors hope to work with the IETF to extend and align Redfish network management with IETF’s YANG framework.

Redfish is a management standard using a data model representation inside of a RESTful interface. The model is expressed in terms of a standard, machine-readable schema, with the payload of the messages being expressed in JSON.

Because it is a model-based API, Redfish is capable of representing a variety of implementations via a consistent interface. It has mechanisms for managing data center resources, handling events, long lived tasks and discovery, as well.

In Redfish, every URL represents a resource. This could be a service, a collection, an entity or some other construct. But in RESTful terms, these URIs point to resources and Redfish clients interact with these resources. For example, the content of a resource, in JSON format, is returned when the Redfish client performs a HTTP GET.

OData is an OASIS standard for expressing the schema of a JSON document. OData allows a fuller description of the JSON document, than json-schema. The format of OData schema is specified in CSDL (Common Schema Definition Language).

OData also describes directives that can appear on the URI to control the contents of the HTTP response. In Redfish, these directives (i.e. $top and $skip) are stated as 'should'. Networking extension may change the requirement to 'shall'.

Redfish releases include both OData and json-schema schema. With json-schema, users can take advantage of its larger tool chain.

Additional information about OData can be found at the OData site [1].

Additional information about Redfish can be found at DMTF's Redfish site [2]. Specifically, the Redfish White Paper [3] provides a good overview.

2.2. YANG and Redfish

Within the networking world, YANG has become the preferred management framework. YANG expresses each section of the overall model as a module containing a tree of nodes where each node is either a container node that builds the hierarchy or a leaf node containing data elements for the model. Redfish network management leverages YANG as the core model definition language to maintain consistency with other YANG based network management approaches. Using a common model structure results in equivalent data elements yielding the same data or content when accessed via different interfaces or protocols.

Redfish's network management supports this consistency by reusing YANG modules as Redfish models for network functionality and services, mechanically translating those modules to the Redfish interface, based on HTTP, JSON, and OData.

The Redfish approach to network management enables definitions of a specific system views or profiles that are aligned with the configuration functionality required in a specific scenario or for a specific class of network devices. A particularly relevant example is a baseline switch model description with a minimum set of model elements; this is useful when configuring a switch within a larger converged infrastructure system. The model elements of the baseline switch should be the smallest set of models necessary to configure a data center switch in a converged infrastructure environment; the corresponding set of YANG modules would be a simple data center network device profile. A more complex network router might need

tunnel models, overlay models, extensive QoS models, and other elements.

The top level resource structure of Redfish is show below.

```
./redfish/v1/Systems
./redfish/v1/Chassis
./redfish/v1/NetworkDevices
(other redfish resources)
```

Within this structure a network switch is viewed as a data center element for its common subsystems such as chassis, power, thermal, cooling, management access setup, etc while the network modeling is specified within the instances of the NetworkDevices[] collection. Each member of the NetworkDevices[] collection has implements its own set translated YANG modules. For consistency, the set of modules grouped under a NetworkDevice instance can follow one of multiple standard groupings expressed as a profile to manage different classes of equipment and satisfy different use cases. Two profile examples are the basic switch and virtualized edge router.

2.3. YANG mapping to Redfish

The notion of schema is different in Redfish and YANG.

In YANG, a schema is device specific. The user determines the YANG modules utilized by a system, and may consult a module library as part of doing so (e.g., RFC7895 [4]). The YANG schema is realized as a set of YANG modules, each with a prescribed node tree structure.

In Redfish, there is one schema that encompasses the entire namespace. In other words, Redfish has a global namespace for its schema, of which the device implements a subset. The Redfish schema is realized as resources accessed via a URI, so the namespace contains the information about which YANG modules are utilized. The OData directives $expand and $filter allows the client to discovery this directly from the parent namespace node above the modules.

That functionality obviates any Redfish need to use YANG module combination techniques such as YANG Schema-mount [5].

Despite these differences, the proposed profiles should be usable by both YANG based protocols (e.g., NETCONF, RESTCONF) and Redfish, as the core content of each profile is a set of YANG modules.

To allow Redfish to manage network devices, the YANG modules needs to be translated into OData CSDL (Common Schema Definition Language). The translation is specified in the YANG-to-Redfish Mapping

Specification [6]. The translation has the following characteristics:

- Includes, imports, and augments, are compiled out to create a single consistent schema block constituting a particular instance of a NetworkDevice.
- The YANG node tree layout is reflected in the URI layout
- The individual YANG container nodes and list nodes are rendered as resources with the YANG tree hierarchy reflected as navigation properties.

Access to the YANG data model elements uses a Redfish JSON accessed via a provider on the URI target.

Leaf nodes representing common back end system "features or elements" return consistent data independent of node name and network device hierarchy.

The NetworkDevices[] collection allows

- Multiple co-existing and consistent views onto a system.
  - Horizontally extensible
  - Vertical hierarchy allowing for control interface delegation
- This is similar to a "view class" or facade approach in software.

2.4. Example Mapping

The following shows the resource which results from mapping RFC7223 (ietf_interface module) to the Redfish schema. Below is a fragment of the data model from the RFC.

```
+--rw interfaces
| +--rw interface* [name]
| +--rw name string
| +--rw description? string
| +--rw type identityref
| +--rw enabled? boolean
| +--rw link-up-down-trap-enable? enumeration
+--ro interfaces-state
+--ro interface* [name]
+--ro name string
+--ro type identityref
+--ro admin-status enumeration
```

The translation to Redfish CSDL is performed using the RFC's YANG code. The translation will generate the CSDL files for the ietf_interfaces resource and each YANG container. The path to these resources mirror the above data model.

```
./redfish/v1/NetworkDevices/Switch1
./redfish/v1/NetworkDevices/Switch1/ietf_interfaces
./redfish/v1/NetworkDevices/Switch1/ietf_interfaces/interfaces
./redfish/v1/NetworkDevices/Switch1/ietf_interfaces/interfaces/ethernet1
./redfish/v1/NetworkDevices/Switch1/ietf_interfaces/interfaces_state
...
```

A HTTP GET of the "ethernet1" singleton resource will return the following JSON document. Note that each property from the above data model is present in the resource.

```
{
    "Id": "ethernet1",
    "Name": "ethernet1",
    "Description": "Ethernet interface on slot 1",
    "type": "iana_if_type:ethernetCsmacd",
    "enabled": "true",
    "link_up_down_trap_enable": "true"

    "@odata.context":
        "/redfish/v1/$metadata#ietf_interfaces.interfaces.interface.
            interface",
    "@odata.type": "#interface.v1_0_0.interfaces",
    "@odata.id":
        "/redfish/v1/NetworkDevices/Switch1/ietf_interfaces
            /interfaces/ethernet1"
}
```

The three properties at the end of the JSON document are OData annotations.

3. Security Considerations

Redfish also improves security control since there is a single point of management contact for a device to control all of its functions.

(Additional security discussion will be provided later.)

4. Appendix A

YANG models needed to managed a network switch:

- RFC7223 (Interfaces)

- RFC7224 (IANA)
- RFC7277 (IPv4, IPv6)
- RFC7317 (System Identification, Time-Date, NTP)
- VLANs
- ACLs
- Syslog

5. Appendix B

The following describes how the Redfish NetworkDevices[] collection resource allows multiple co-existing and consistent views onto a system.

As an example, a router could have the following:

//redfish.example.net/redfish/v1/NetworkDevices/masterRouter
//redfish.example.net/redfish/v1/NetworkDevices/vrf1
//redfish.example.net/redfish/v1/NetworkDevices/vrf2

In this example, masterRouter represents the complete system with all interfaces, all tables, all system level configuration, and a model structure for assigning resources to virtual instances. The resources, vrf1 and vrf2, represent a particular partitioning of the system to create virtual router instances each assigned a subset of the total resource pool.

The above structure has similarities with that expressed by the device model from the following references:

- https://tools.ietf.org/html/draft-ietf-rtgwg-device-model-01
- https://tools.ietf.org/html/draft-ietf-rtgwg-ni-model-01
- https://tools.ietf.org/html/draft-ietf-rtgwg-lne-model-01
- https://tools.ietf.org/html/draft-ietf-netmod-schema-mount-03

In these references a Network Device contains Logical Network Elements which, in turn, contain Network Instances. From the device model reference, the Network Device represents the system as a whole. The Logical Network Element represents a partition of a physical system. The Logical Network Element represents a VRF or VSI (virtual switching instance).

The Redfish NetworkDevices collection resource would map this modeling approach by using an element of the collection for the Network Device and one for each of the Logical Network Elements and Network Instances. These collection elements would add references at the NetworkDevices element level to map the containment of of the device model. The overall ./redfish/v1/ root maps to the Routing Area Network Device.

6. Appendix C

The following is the ietf_interfaces.interfaces.interface_v1.xml CSDL metadata file, which is referenced in @odata.context annotation in the example mapping. The entity type referenced in the @odata.type annotation is in the second Namespace.

When mapping YANG code to CDSL, values are mapped to existing OData core properties, when possible. Otherwise, new annotations are defined in RedfishYangExtensions.xml. This file is referenced at the beginning of the document.

```
<?xml version="1.0" encoding="UTF-8"?>
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
        Version="4.0">
    <Edmx:Reference
        Uri="http://docs.oasis-open.org/odata/odata/v4.0/cs01/
            vocabularies/Org.OData.Core.V1.xml">
        <Edmx:Include Alias="Odata" Namespace="Org.OData.Core.V1"/>
    </Edmx:Reference>
    <Edmx:Reference
        Uri="http://docs.oasis-open.org/odata/odata/v4.0/cs01/
            vocabularies/Org.OData.Capabilities.V1.xml">
        <Edmx:Include Alias="Odata"
            Namespace="Org.OData.Capabilities.V1"/>
    </Edmx:Reference>
    <Edmx:Reference
        Uri="http://redfish.dmtf.org/schemas/v1/
            RedfishYangExtensions.xml">
        <Edmx:Include Alias="Redfish.Yang"
            Namespace="Redfish.Yang"/>
    </Edmx:Reference>

    <Edmx:DataServices>

    <Schema Namespace="interface"
            xmlns="http://docs.oasis-open.org/odata/ns/edm" >
        <EntityType Name="interface"
                BaseType="Resource.v1_0_0.Resource">
            <Annotation Term="OData.Description"
```

```
                    String="<manual input>." />
                <Annotation Term="OData.AdditionalProperties"
                    Bool="False"/>
            </EntityType>
        </Schema>

        <Schema Namespace="interface.v1_0_0"
                xmlns="http://docs.oasis-open.org/odata/ns/edm" >
            <EntityType Name="interface" BaseType=
                    "ietf_interfaces.interfaces.interface.interface" >
                <Annotation Term="OData.Description"
                    String="<manual input>." />
                <Annotation Term="OData.AdditionalProperties"
                    Bool="False"/>
                <Annotation Term="Redfish.Yang.NodeType"
                    EnumMember="Redfish.Yang.NodeTypes/list" />
                <Annotation Term="Redfish.Yang.key"
                    String=" the yang key string"/>
                <Key>
                    <PropertyRef Name="name"/>
                </Key>
                <Property Name="name" Type="Edm:String">
                    <Annotation Term="OData.Description"
                        String="..." />
                    <Annotation Term="OData.Permissions"
                        EnumMember="OData.Permissions/Read"/>
                    <Annotation Term="Redfish.Yang.NodeType"
                        EnumMember="Redfish.Yang.NodeTypes/leaf"/>
                    <Annotation Term="Redfish.Yang.YangType"
                        String="string" />
                </Property>
                <Property Name="description" Type="Edm:String">
                    <Annotation Term="OData.Description"
                        String="..." />
                    <Annotation Term="OData.Permissions"
                        EnumMember="OData.Permissions/Read"/>
                    <Annotation Term="Redfish.Yang.NodeType"
                        EnumMember="Redfish.Yang.NodeTypes/leaf" />
                    <Annotation Term="Redfish.Yang.YangType"
                        String="string" />
                    <Annotation Term="Redfish.Yang.reference"
                        String="RFC 2863: The Interfaces Group..." />
                </Property>
                <Property Name="type" Type="Edm:String">
                    <Annotation Term="OData.Description" String="..."/>
                    <Annotation Term="Redfish.Yang.NodeType"
                        EnumMember="Redfish.Yang.NodeTypes/leaf" />
                    <Annotation Term="Redfish.Yang.YangType"
```

```
                        String="identityref"/>
                    <Annotation Term="Redfish.Yang.base"
                        String="interface-type"/>
                    <Annotation Term="Redfish.Yang.mandatory"
                        EnumMember="Redfish.Yang.Mandatory/true"/>
                    <Annotation Term="Redfish.Yang.reference"
                        String="RFC 2863: The Interfaces Group..." />
                </Property>
                <Property DefaultValue="true" Name="enabled"
                        Type="Edm:Boolean">
                    <Annotation Term="OData.Description"
                        String="This leaf contains..." />
                    <Annotation Term="Redfish.Yang.NodeType"
                        EnumMember="Redfish.Yang.NodeTypes/leaf" />
                    <Annotation Term="Redfish.Yang.YangType"
                        String="boolean"/>
                    <Annotation Term="Redfish.Yang.reference"
                        String="RFC 2863: The Interfaces..."/>
                </Property>
                <Property Name="link_up_down_trap_enable"
                        Type="Edm:Enumeration">
                    <Annotation Term="OData.Description"
                        String="Controls whether..."/>
                    <Annotation Term="Redfish.Yang.NodeType"
                        EnumMember="Redfish.Yang.NodeTypes/leaf"/>
                    <Annotation Term="Redfish.Yang.YangType"
                        String="enumeration"/>
                    <Annotation Term="Redfish.Yang.if_feature"
                        String="if-mib"/>
                    <Annotation Term="Redfish.Yang.reference"
                        String="RFC 2863: The Interfaces..." />
                    <EnumType
                            Name="link_up_down_trap_enableEnumeration">
                        <Member Name="enabled" Value="1">
                            <Annotation Term="Redfish.Yang.enum"
                                String="enabled"/>
                        </Member>
                        <Member Name="disabled" Value="2">
                            <Annotation Term="Redfish.Yang.enum"
                                String="disabled"/>
                        </Member>
                    </EnumType>
                </Property>
            </EntityType>
        </Schema>

        </Edmx:DataServices>
```

```
    </edmx:Edmx>
```

7. Appendix D

The following is the IETF YANG source XML from RFC7223 used for the example mapping.

```
<CODE BEGINS> file "ietf-interfaces@2014-05-08.yang"
module ietf-interfaces {
    namespace "urn:ietf:params:xml:ns:yang:ietf-interfaces";
    prefix if;
    import ietf-yang-types {
        prefix yang;
    }
    organization
        "IETF NETMOD (NETCONF Data Modeling Language) Working Group";
    . . .
```

After the typedef, identity, and feature statements, the data model is defined. Below is the fragment that becomes ietf_interfaces.interfaces.interface_v1.xml.

```
    /*
     * Configuration data nodes
     */
    container interfaces {
        description
            "Interface configuration parameters.";
        list interface {
            key "name";
            description
                "The list of configured interfaces...";
        leaf name {
            type string;
            description
                "The name of the interface...";
        }
        leaf description {
            type string;
            description
                "A textual description of the interface...";
            reference
                "RFC 2863: The Interfaces Group MIB - ifAlias";
        }
        leaf type {
            type identityref {
                base interface-type;
            }
```

```
                mandatory true;
                description
                    "The type of the interface...";
                reference
                    "RFC 2863: The Interfaces Group MIB - ifType";
            }
            leaf enabled {
                type boolean;
                default "true";
                description
                    "This leaf contains the configured,...";
                reference
                    "RFC 2863: The Interfaces Group MIB - ifAdminStatus";

            leaf link-up-down-trap-enable {
                if-feature if-mib;
                type enumeration {
                    enum enabled {
                        value 1;
                    }
                    enum disabled {
                        value 2;
                    }
                }
                description
                    "Controls whether linkUp/linkDown SNMP...";
                reference
                    "RFC 2863: The Interfaces Group MIB -
                        ifLinkUpDownTrapEnable";
            }
        }
        . . .
```

8. References

8.1. Normative References

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

8.2. URIs

[1] http://odata.org

[2] http://dmtf.org/redfish

[3] http://www.dmtf.org/sites/default/files/standards/documents/DSP2044_1.0.0.pdf

[4] http://www.rfc-editor.org/info/rfc7895

[5] https://tools.ietf.org/html/draft-ietf-netmod-schema-mount-03

[6] http://www.dmtf.org/sites/default/files/standards/documents/
DSP0271_0.5.6.pdf

Authors' Addresses

Joseph White
Dell EMC

Email: joseph.l.white@dell.com

David Black
Dell EMC

Email: david.black@dell.com

John Leung
Intel Corporation

Email: john.leung@intel.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: July 6, 2018

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

January 2, 2018

IS-IS Support for Openfabric
draft-white-openfabric-05

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 Intermediate System (IS) (also called a 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 https://datatracker.ietf.org/drafts/current/.

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

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

This Internet-Draft will expire on July 6, 2018.

Table of Contents

1. Introduction 3
1.1. Goals 3
1.2. Contributors 3
1.3. Simplification 3
1.4. Additions and Requirements 4
1.5. Sample Network 4
2. Modified Adjacency Formation 6
2.1. Level 2 Adjacencies Only 6
2.2. Point-to-point Adjacencies 6
2.3. Three Way Handshake Support 7
2.4. Adjacency Formation Optimization 7
3. Advertisement of Reachability Information 8
4. Determining and Advertising Location on the Fabric 9
4.1. Calculating Tier Number with a Fixed T0 9
4.2. Calculating the Tier Number in a Five Stage Spine and Leaf 10
5. Flooding Optimization 12
5.1. Flooding Failures 13
6. Other Optimizations 13
6.1. Transit Link Reachability 13
6.2. Transiting T0 Intermediate Systems 14
7. Openfabric and Route Aggregation 14
8. Security Considerations 14
9. References 15
9.1. Normative References 15
9.2. Informative References 16

Authors' Addresses 18

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 IS 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:

- 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 IS in the fabric, such as PCEP [RFC5440] or I2RS [RFC7921]. Traffic engineering is assumed to be provided through Segment Routing (SR) [I-D.ietf-spring-segment-routing].

1.4. Additions and Requirements

To create a scalable link state fabric, openfabric includes the following:

- A slightly modified adjacency formation process.
- Mechanisms for determining which tier within a spine and leaf fabric in which the IS is located.
- A mechanism that reduces flooding to the minimum possible, while still ensuring complete database synchronization among the intermediate systems within the fabric.

Three general requirements are placed here; more specific requirements are considered in the following sections. Openfabric implementations:

- MUST support [RFC5301] and enable hostname advertisement by default if a hostname is configured on the intermediate system.
- SHOULD support [RFC6232], purge originator identification for IS-IS.
- 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) intermediate systems. The remaining tiers, T1 and T2, are connected only to the fabric itself. Note there are no "cross links," or "east west" links in the illustrated fabric. The fabric locality detection mechanism described here will not work if there are cross links running east/

west through the fabric. Locality detection may be possible in such a fabric; this is an area for further study.

2. Modified Adjacency Formation

Because Openfabric operates in a tightly controlled data center environment, various modifications can be made to the IS-IS neighbor formation process to increase efficencicy and simplify the protocol. Specifically, Openfabric implementations SHOULD support [RFC3719], section 4, hello padding for IS-IS. Variable hello padding SHOULD NOT be used, as data center fabrics are built using high speed links on which padded hellos will have little performance impact. Further modifications to the neighbor formation process are considered in the following sections.

2.1. Level 2 Adjacencies Only

Openfabric is designed to work in a single flooding domain over a single data center fabric at the scale of thousands of routers with hundreds of thousands of routes (so a moderate scale in router and route count terms). Because of the way Openfabric optimizes operation in this environment, it is not necessary nor desirable to build multiple flooding domains. For instance, the flooding optimizations described later this document require a full view of the topology, as does any proposed overlay to inject policy into the forwarding plane. In light of this, the following changes SHOULD BE to IS-IS implemetations to support Openfabric:

- IIH PDU 17 (level 2 point-to-point circuit hello) should be the only IIH PDU type transmitted (see section 9.7 of ISO 10589)
- In IIH PDU 17 (level 2 point-to-point circuit hello), the Circuit Type field should be set to 2 (see section 9.7 of ISO 10589)
- Support for IIH PDU 15 (level 1 broadcast hello) should be removed (see section 9.5 of ISO 10589)
- Support for IIH PDU 16 (level 2 broadcast hello) should be removed (see section 9.6 of ISO 10589)

2.2. Point-to-point Adjacencies

Data center network fabrics only contain point-to-point links; because of this, there is no reason to support any broadcast link types, nor to support the Designated Intermediate System processing, including pseudonode creation. In light ot his, processing related to sections 7.2.3 (broadcast networks), 7.3.8 (generation of level 1 pseudonode LSPs), 7.3.10 (generation of level 2 pseudonode LSPs), and

section 8.4.5 (LAN designated intermediate systems) in [ISO10589] SHOULD BE removed.

2.3. Three Way Handshake Support

It is important that two way connectivity be established before synchronizing the link state database, or routing through a link in a data center fabric. To reject optical failures that cause a one way connection between two routers, fabricDC must support the three way handshake mechanism described in [RFC5303].

2.4. Adjacency Formation Optimization

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

- 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.
- If synchronization is not achieved within twice the dead timer on the local interface, the newly connected IS will repeat this process with the second neighbor with which it forms a three-way adjacency.

This process allows each IS newly added to the fabric to exchange a full table once; a very minimal amount of information will be transferred with the remaining neighbors to reach full synchronization.

Any such optimization is bound to present a tradeoff between several factors; the mechanism described here increases the amount of time required to form adjacencies slightly in order to reduce the total state carried across the network. An alternative mechanism could provide a better balance of the amount of information carried across the network for initial synchronization and the time required to synchronize a new IS. For instance, an IS could choose to

synchronize its database with two or three adjacent intermediate systems, which could speed the synchronization process up at the cost of carrying additional data on the network. A locally determined balance between the speed of synchronization and the amount of data carried on the network can be acheived by adjusting the number of adjacent intermediate systems the newly attached IS synchronizes with.

3. Advertisement of Reachability Information

IS-IS describes the topology in two different sets of TLVs; the first describes the set of neighbors connected to an IS, the second describes the set of reachable destination connected to an IS. There are two different forms of both of these descriptions, one of which carries what are widely called narrow metrics, the other of which carries what are widely called wide metrics. In a tightly controlled data center fabric implementation, such as the ones Openfabric is designed to support, no IS that supports narrow metrics will ever be deployed or supported; hence there is no reason to support any metric type other than wide metrics.

- The Level 2 Link State PDU (type 20 in section 9.9 of [ISO10589]) and the scoped flooding PDU (type 10 in section 3.1 of [RFC7356]) SHOULD BE the only PDU types used to carry link state information in a Openfabric implementation

- Processing related to the Level 1 Link State PDU (type 18) MAY BE removed from Openfabric implementations (see section 9.8 of [ISO10589])

- Neighbor reachability MUST BE carried in TLV type 22 (see section 3 of [RFC5305])

- IPv4 reachability SHOULD BE carried in TLV type 135 (see section 4 of [RFC5305]), or TLV type 235 for multitopology implementations (see [RFC5120])

- IPv6 reachability SHOULD BE carried in TLV type 236 (see [RFC5308]), or TLV type 237 for multitopology implemenations (see [RFC5120])

- Processing related to the neighbor reachability TLV (type 2, see sections 9.8 and 9.9 of [ISO10589]) SHOULD BE removed

- Processing related to the narrow metric IP reachability TLV (types 128 and 130) SHOULD BE removed

In order to support segment routing, Openfabric needs to be able to support the advertisement of a Prefix-SID tied to a local loopback address assigned to the IS. The configuration of the label to advertise MAY BE manually configured for the moment or determined through autoconfiguration. A Prefix-SID SHOULD BE advertised if a local label is configured using the Prefix Segment Identifier sub-TLV (see section 2.1 of [I-D.ietf-isis-segment-routing-extensions]).

4. Determining and Advertising Location on the Fabric

The tier to which a IS is connected is useful to enable autoconfiguration of intermediate systems connected to the fabric and to reduce flooding. Once the tier of an intermediate system within the fabric has been determined, it MUST be advertised using the 4 bit Tier field described in section 3.3 of [I-D.shen-isis-spine-leaf-ext]. This section describes two mechanisms for determining the tier at which a IS is connected in the fabric in several steps.

4.1. Calculating Tier Number with a Fixed T0

The first method begins with one of the T0 intermediate systems advertising its location in the fabric. This information can either be obtained through:

- A single T0 intermediate system is manually configured to advertise 0x00 in their IS reachability tier sub-TLV, indicating they are at the edge of the fabric (a ToR IS).

- The T0 intermediate systems detect they are T0 through the presence connected hosts (i.e. through a request for address assignment or some other means). If such detection is used, and the IS determines it is located at T0, it should advertise 0x00 in its IS reachability tier sub-TLV.

The second method above SHOULD be used with care, as it may not be secure, and it may not work in all data center environments. For instance, if a host is mistakenly (or intentionally, as a form of attack) attached to a spine IS, or a request for address assignment is transmitted to a spine IS during the bootup phase of the device or fabric, it is possible to cause a spine IS to advertise itself as a T0. Unless the autodetection of the T0 devices is secured, the manual mechanism SHOULD BE used (configuring at least one T0 device manually).

Given at least one T0 device is advertising its tier number, the remaining intermediate systems calculate their tier number as follows:

- The local IS calculates an SPT (using SPF) setting the cost of every link to 1; this effectively calculates a topology only view of the network, without considering any configured link costs

- Find the closest IS advertising a tier number of 0 in the Spine Leaf extension sub-TLV; call this node A, and set FD to this cost

- Calculate an SPT (using SPF) from the perspective of A (above), and setting the cost of every link to 1; the maximum cost to any node should be 2 for a 3 stage fabric, 4 for a 5 stage fabric, etc.

- Choose any node that is a maximum metric from A (above); call this IS B

- Find the cost to B on the locally calculated SPT from the first step; call this TD

- Calculate the tier number of the local node by subtracting FD from TD

In the example network, assume 5A is manually configured as a T0, and is advertising its tier number. From here:

- From 1A the path to 5A is 4 hops; this is FD

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

- The maximum path length is 4; 1F is one such node; set this node to B, and set TD to 4

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

This process will work for any spine and leaf fabric without "cross links."

4.2. Calculating the Tier Number in a Five Stage Spine and Leaf

In some fabrics, it is possible to calculate which intermediate systems are at T0 using a modified Shortest Path First (SPF) calculation. Specifically, if the fabric is configured in five stages, as shown in the example network, and is not some form of butterfly, Benes, or a three stage fabric, it is possible to calcualte if an IS is at T0 using the following process:

- 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

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

For the remaining intermediate systems to determine which tier they are situated on, they perform the following calculation:

- 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

The IS 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

5. Flooding Optimization

Flooding is perhaps the most challenging scaling issue for a link state protocol running on a dense, large scale fabric. To reduce the flooding of link state information in the form of Link State Protocol Data Units (LSPs), Openfabric takes advantage of information already available in the link state protocol, the list of the local intermediate system's neighbor's neighbors, and the fabric locality computed above. The following tables are required to compute a set of reflooders:

- Neighbor List (NL) list: The set of neighbors
- Neighbor's Neighbors (NN) list: The set of neighbor's neighbors; this can be calculated by running SPF truncated to two hops
- Do Not Reflood (DNR) list: The set of neighbors who should have LSPs (or fragments) who should not reflood LSPs
- Reflood (RF) list: The set of neighbors who should flood LSPs (or fragments) to their adjacent neighbors to ensure synchronization

NL is set to contain all neighbors, and sorted deterministically (for instance, from the highest IS identifier to the lowest). All intermediate systems within a single fabric SHOULD use the same mechanism for sorting the NL list. NN is set to contain all neighbor's neighbors, or all intermediate systems that are two hops away, as determined by performing a truncated SPF. The DNR and RF tables are initially empty. To begin, the following steps are taken to reduce the size of NN and NL:

- 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

  o 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, MUST be transmitted using a circuit scope PDU as described in [RFC7356].

5.1. Flooding Failures

It is possible in some failure modes for flooding to be incomplete because of the flooding optimizations outlined. Specifically, if a reflooder fails, or is somehow disconnected from all the links across which it should be reflooding, it is possible an LSP is only partially flooded through the fabric. To prevent such situations, any IS receiving an LSP transmitted using DNR SHOULD:

- 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

6. Other Optimizations

6.1. Transit Link Reachability

In order to reduce the amount of control plane state carried on large scale spine and leaf fabrics, openfabric implementations SHOULD NOT advertise reachability for transit links. These links MAY remain unnumbered, as IS-IS does not require layer 3 IP addresses to operate. Each IS SHOULD be configured with a single loopback address, which is assigned an IPv6 address, to provide reachability to intermediate systems which make up the fabric.

[RFC3277] SHOULD be supported on devices supporting openfabric with unnumbered interface in order to support traceability and network management.

6.2. Transiting T0 Intermediate Systems

In data center fabrics, ToR intermediate systems SHOULD NOT be used to transit between two T1 (or above) spine intermediate systems. The simplest way to prevent this is to set the overload bit [RFC3277] for all the LSPs originated from T0 intermediate systems. However, this solution would have the unfortunate side effect of causing all reachability beyond any T0 IS to have the same metric, and many implementations treat a set overload bit as a metric of 0xFFFF in calculating the Shortest Path Tree (SPT). This document proposes an alternate solution which preserves the leaf node metric, while still avoiding transiting T0 intermediate systems.

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

7. Openfabric and Route Aggregation

While schemes may be designed so reachability information can be aggregated in Openfabric deployments, this is not a recommended configuraiton.

8. Security Considerations

This document outlines modifications to the IS-IS protocol for operation on large scale data center fabrics. While it does add new TLVs, and some local processing changes, it does not add any new security vulnerabilities to the operation of IS-IS. However, openfabric implementations SHOULD implement IS-IS cryptographic authentication, as described in [RFC5304], and should enable other security measures in accordance with best common practices for the IS-IS protocol.

If T0 intermediate systems are auto-detected using information outside Openfabric, it is possible to attack the calucations used for flooding reduction and auto-configuration of intermediate systems. For instance, if a request for an address pool is used as an indicator of an attached host, and hence receiving such a request causes an intermediate system to advertise itself as T0, it is possible for an attacker (or a simple mistake) to cause auto-configuration to fail. Any such auto-detection mechanims SHOULD BE

secured using appropriate techniques, as described by any protocols or mechanisms used.

9. References

9.1. Normative References

[I-D.shen-isis-spine-leaf-ext]
Shen, N., Ginsberg, L., and S. Thyamagundalu, "IS-IS Routing for Spine-Leaf Topology", draft-shen-isis-spine-leaf-ext-05 (work in progress), January 2018.

[ISO10589]
International Organization for Standardization, "Intermediate system to Intermediate system intra-domain routeing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode Network Service (ISO 8473)", ISO/IEC 10589:2002, Second Edition, Nov 2002.

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

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

[RFC5120] Przygienda, T., Shen, N., and N. Sheth, "M-ISIS: Multi Topology (MT) Routing in Intermediate System to Intermediate Systems (IS-ISs)", RFC 5120, DOI 10.17487/RFC5120, February 2008, <https://www.rfc-editor.org/info/rfc5120>.

[RFC5301] McPherson, D. and N. Shen, "Dynamic Hostname Exchange Mechanism for IS-IS", RFC 5301, DOI 10.17487/RFC5301, October 2008, <https://www.rfc-editor.org/info/rfc5301>.

[RFC5303] Katz, D., Saluja, R., and D. Eastlake 3rd, "Three-Way Handshake for IS-IS Point-to-Point Adjacencies", RFC 5303, DOI 10.17487/RFC5303, October 2008, <https://www.rfc-editor.org/info/rfc5303>.

[RFC5305] Li, T. and H. Smit, "IS-IS Extensions for Traffic Engineering", RFC 5305, DOI 10.17487/RFC5305, October 2008, <https://www.rfc-editor.org/info/rfc5305>.

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

[RFC5309] Shen, N., Ed. and A. Zinin, Ed., "Point-to-Point Operation over LAN in Link State Routing Protocols", RFC 5309, DOI 10.17487/RFC5309, October 2008, <https://www.rfc-editor.org/info/rfc5309>.

[RFC5311] McPherson, D., Ed., Ginsberg, L., Previdi, S., and M. Shand, "Simplified Extension of Link State PDU (LSP) Space for IS-IS", RFC 5311, DOI 10.17487/RFC5311, February 2009, <https://www.rfc-editor.org/info/rfc5311>.

[RFC5316] Chen, M., Zhang, R., and X. Duan, "ISIS Extensions in Support of Inter-Autonomous System (AS) MPLS and GMPLS Traffic Engineering", RFC 5316, DOI 10.17487/RFC5316, December 2008, <https://www.rfc-editor.org/info/rfc5316>.

[RFC7356] Ginsberg, L., Previdi, S., and Y. Yang, "IS-IS Flooding Scope Link State PDUs (LSPs)", RFC 7356, DOI 10.17487/RFC7356, September 2014, <https://www.rfc-editor.org/info/rfc7356>.

[RFC7981] Ginsberg, L., Previdi, S., and M. Chen, "IS-IS Extensions for Advertising Router Information", RFC 7981, DOI 10.17487/RFC7981, October 2016, <https://www.rfc-editor.org/info/rfc7981>.

9.2. Informative References

[I-D.ietf-isis-segment-routing-extensions] Previdi, S., Ginsberg, L., Filsfils, C., Bashandy, A., Gredler, H., Litkowski, S., Decraene, B., and J. Tantsura, "IS-IS Extensions for Segment Routing", draft-ietf-isis-segment-routing-extensions-15 (work in progress), December 2017.

[I-D.ietf-spring-segment-routing] Filsfils, C., Previdi, S., Ginsberg, L., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", draft-ietf-spring-segment-routing-14 (work in progress), December 2017.

[RFC3277] McPherson, D., "Intermediate System to Intermediate System (IS-IS) Transient Blackhole Avoidance", RFC 3277, DOI 10.17487/RFC3277, April 2002, <https://www.rfc-editor.org/info/rfc3277>.

[RFC3719] Parker, J., Ed., "Recommendations for Interoperable Networks using Intermediate System to Intermediate System (IS-IS)", RFC 3719, DOI 10.17487/RFC3719, February 2004, <https://www.rfc-editor.org/info/rfc3719>.

[RFC4271] Rekhter, Y., Ed., Li, T., Ed., and S. Hares, Ed., "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, DOI 10.17487/RFC4271, January 2006, <https://www.rfc-editor.org/info/rfc4271>.

[RFC5304] Li, T. and R. Atkinson, "IS-IS Cryptographic Authentication", RFC 5304, DOI 10.17487/RFC5304, October 2008, <https://www.rfc-editor.org/info/rfc5304>.

[RFC5440] Vasseur, JP., Ed. and JL. Le Roux, Ed., "Path Computation Element (PCE) Communication Protocol (PCEP)", RFC 5440, DOI 10.17487/RFC5440, March 2009, <https://www.rfc-editor.org/info/rfc5440>.

[RFC5449] Baccelli, E., Jacquet, P., Nguyen, D., and T. Clausen, "OSPF Multipoint Relay (MPR) Extension for Ad Hoc Networks", RFC 5449, DOI 10.17487/RFC5449, February 2009, <https://www.rfc-editor.org/info/rfc5449>.

[RFC5614] Ogier, R. and P. Spagnolo, "Mobile Ad Hoc Network (MANET) Extension of OSPF Using Connected Dominating Set (CDS) Flooding", RFC 5614, DOI 10.17487/RFC5614, August 2009, <https://www.rfc-editor.org/info/rfc5614>.

[RFC5837] Atlas, A., Ed., Bonica, R., Ed., Pignataro, C., Ed., Shen, N., and JR. Rivers, "Extending ICMP for Interface and Next-Hop Identification", RFC 5837, DOI 10.17487/RFC5837, April 2010, <https://www.rfc-editor.org/info/rfc5837>.

[RFC6232] Wei, F., Qin, Y., Li, Z., Li, T., and J. Dong, "Purge Originator Identification TLV for IS-IS", RFC 6232, DOI 10.17487/RFC6232, May 2011, <https://www.rfc-editor.org/info/rfc6232>.

[RFC7182] Herberg, U., Clausen, T., and C. Dearlove, "Integrity Check Value and Timestamp TLV Definitions for Mobile Ad Hoc Networks (MANETs)", RFC 7182, DOI 10.17487/RFC7182, April 2014, <https://www.rfc-editor.org/info/rfc7182>.

[RFC7921] Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", RFC 7921, DOI 10.17487/RFC7921, June 2016, <https://www.rfc-editor.org/info/rfc7921>.

Authors' Addresses

Russ White (editor)
LinkedIn

Email: russ@riw.us

Shawn Zandi (editor)
LinkedIn

Email: szandi@linkedin.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 14, 2017

Y. Zheng
China Unicom
S. Xu
D. Dhody
Huawei Technologies
March 13, 2017

Usecases for Network Artificial Intelligence (NAI)
draft-zheng-opsawg-network-ai-usecases-00

Abstract

This document discusses the scope of Network Artificial Intelligence (NAI), and the possible use cases that are able to demonstrate the advantage of applying NAI.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 14, 2017.

Copyright Notice

Table of Contents

1. Introduction 2
2. NAI Architecture 3
3. NAI Use Cases 3
3.1. Traffic Predication and Re-Optimization/Adjustment . . . 3
3.2. Route Monitoring and Analytics 4
3.3. Multilayer Fault Detection In NFV Framework 5
3.4. Data Center Network Use Cases 7
3.4.1. Service Function Chaining 7
4. Contributors 8
5. Security Considerations 8
6. IANA Considerations 8
7. Acknowledgement 8
8. References 8
8.1. Normative References 9
8.2. Informative References 9
Authors' Addresses 9

1. Introduction

Current networks have become much more dynamic and complex, and pose new challenges for network management and optimization. For example, network management/optimization should be automated to avoid human intervention (and thus to minimize the operational expense). Artificial Intelligence (AI) and Machine Learning (ML) is a promising approach to realize such automation, and can even do better than human beings. Furthermore, the population of Software-Defined Networks (SDN) paradigm makes the application of Artificial Intelligence in networks possible, since the SDN controller has the complete knowledge of the network status and can control behavior of network nodes to implement AI decisions.

AI and ML technologies can learn from historical data, and make predictions or decisions, rather than following strictly static program instructions. They can dynamically adapt to a changing situation and enhance their own intelligence with by learning from new data. It can learn and complete complicated tasks. It also has potential in the network technology area especially with SDN and Network Function Virtualization (NFV).

This document presents the concept of Network Artificial Intelligence. It first discusses the scope of Network Artificial Intelligence (NAI). And then Some use cases are discussed to demonstrate the advantage of applying NAI.

2. NAI Architecture

The definition of the architecture of NAI could be refer to [I-D.li-rtgwg-network-ai-arch]. In the architecture of NAI, central controller is the core part of Network Artificial Intelligence which can be called as 'Network Brain'. The Network Telemetry and Analytics (NTA) engines can be introduced acompanying with the central controller. The Network Telemetry and Analytics (NTA) engine inclues data collector, analytics framework, data persistence, and NAI applications.

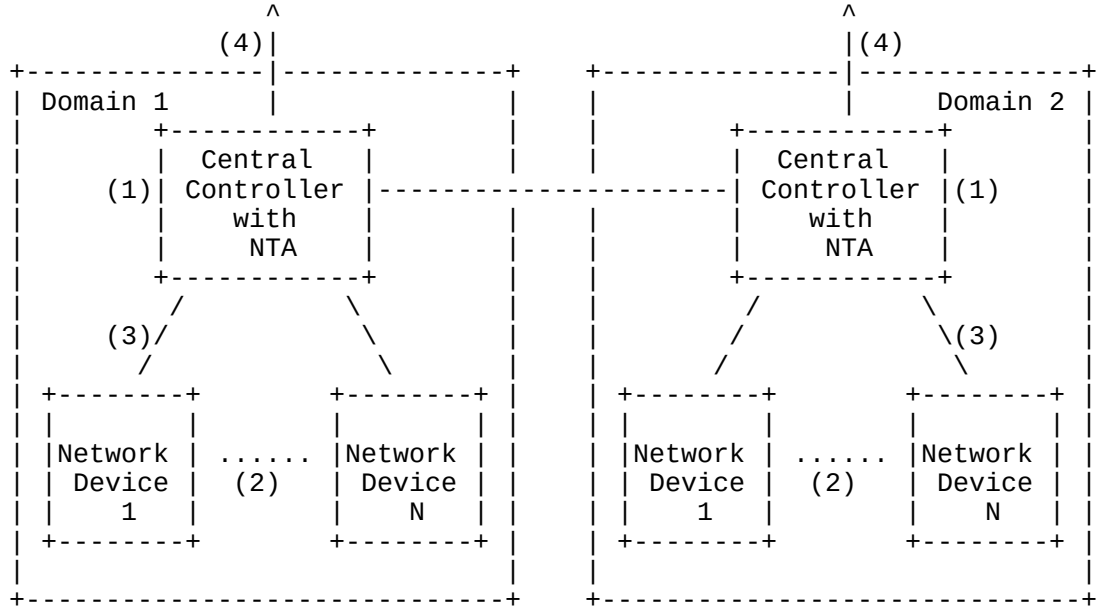


Figure 1: An Architecture of Network Artificial Intelligence(NAI)

3. NAI Use Cases

3.1. Traffic Predication and Re-Optimization/Adjustment

This subsection introduces the Path Computation Element (PCE) [RFC4655] use cases in wide area networks (WAN). In PCE scenario, network data collection is realized through the control plane protocols such as PCE protocol (PCEP) and BGP-LS [RFC7752] protocol and data are passed to the PCE application. PCEP receives the state of Label Switched Path (LSP) from the network, and BGP-LS receives the topology information from the network. If network telemetry is used, traffic information can be received from the network as well directly at the NTA engine using protocols such as gRPC.

PCE application (APP) only maintains the latest information. To enable NAI, history of all LSP and topology changes is stored in external data repository. Further traffic monitoring data could also be collected and stored, if network telemetry is used. There are two usecases in the application scenarios: (1) reroute/re-optimize using the historical trend and predications from AI; (2) traffic congestion avoidance and AI-enabled auto-bandwidth adjustment.

For the usecase (1), the analytics component in NTA (Network Telemetry and Analytics), can use stored data to build models to predict impact of network events and state of the LSPs. For example, it can use historical trends to guide path computation to include/ exclude specific links. Finding correlations between data, finding anomalies and data visualization are also possible.

The analytics component in NTA can also use stored data to detect and predict network events and request PCE to take necessary actions. For example, it can use network bandwidth utilization historical trends to request for re-optimizations.

For the usecase (2), with network telemetry, the NTA can collect per-link and per-LSP traffic flow using gRPC from network. Such network telemetry data includes statistics for tunnels, links, bandwidth reservations, actual usage, delay, jitter, packet loss, etc. Meanwhile, it also collects data regarding network events and its impact on traffic flows. The analytics component can use telemetry data to build traffic models to predict traffic congestion when new years or sporting events are coming. According to the congestion prediction, the PCE app could reroute traffic to avoid congested links. Besides the case, NTA can also perform predication and make necessary changes to network. In particular, the PCE APP performs bandwidth usage prediction (i.e., bandwidth calendaring) by looking at the historical trends of all sampled data instead of the instant sampled data. The collected data are traffic engineering data base (TEDB) and LSP-DB, and can also include scheduling information. In addition, the collected data also include auto-bandwidth related changes under particular network events. Using machine learning algorithm, the analytics component is able to correct such changes with the events, and predicts network events and their impact.

3.2. Route Monitoring and Analytics

This subsection introduces the BGP Monitoring Protocol (BMP) [RFC7854] use case in wide area networks (WAN). The BGP protocol is known for its flexibility and ability to manage a large number of neighbors and routes. It is also the basis for many overlay services such as L3VPN, L2VPN and so on. The BMP protocol can be used by the

controller to monitor BGP protocol neighbor status and routing information on the routers.

According to [RFC7854], BMP client located in the router collects BGP neighbor status, routes for each neighbor, and events defined by the user. And then it passes the informations through the BMP protocol to the management station located on the controller. Based on BMP monitoring of BGP, there are three use cases: (1) BGP Route Leaks Monitoring; (2) BGP Hijacks Monitoring; (3) Traffic Analytics.

Route leaks involve the illegitimate advertisement of prefixes, blocks of IP addresses, which propagate across networks and lead to incorrect or suboptimal routing. For case (1), based on BMP, NAI apps can analyze BGP route leaks.

For case (2), by manipulating BGP, data can be rerouted in an attacker’s favor out them to intercept or modify traffic.If the malicious announcement is more specific than the legitimate one, or claims to offer a shorter path, the traffic may be directed to the attacker.By broadcasting false announcements, the compromised router may poison the RIB of its peers.After poisoning one peer, the malicious routing information could propagate to other peers, to other Autonomous Systems, and onto the interactive Internet. Based on monitoring BGP routes, ML algorithms can be trained to determine when a hijack has taken place and take necessary actions.

In case (3), with BMP protocol providing BGP changes, together with Telemetry providing network traffic information, The NAI Apps can analyze traffic trends, predict traffic changes, and do traffic optimizing.

3.3. Multilayer Fault Detection In NFV Framework

The high reliability and high availability required for carrier-class applications is a big challenge in virtualized and software-based environment where failures are normal in a software-based environment. The interdependence between NFV’s abstraction levels and virtual resources is complex as shown in Fig.. The dynamic characteristics of the resources in the cloud environment make it difficult to locate the fault. So multilayer fault detection for NFV networks and cloud environment will be very useful.

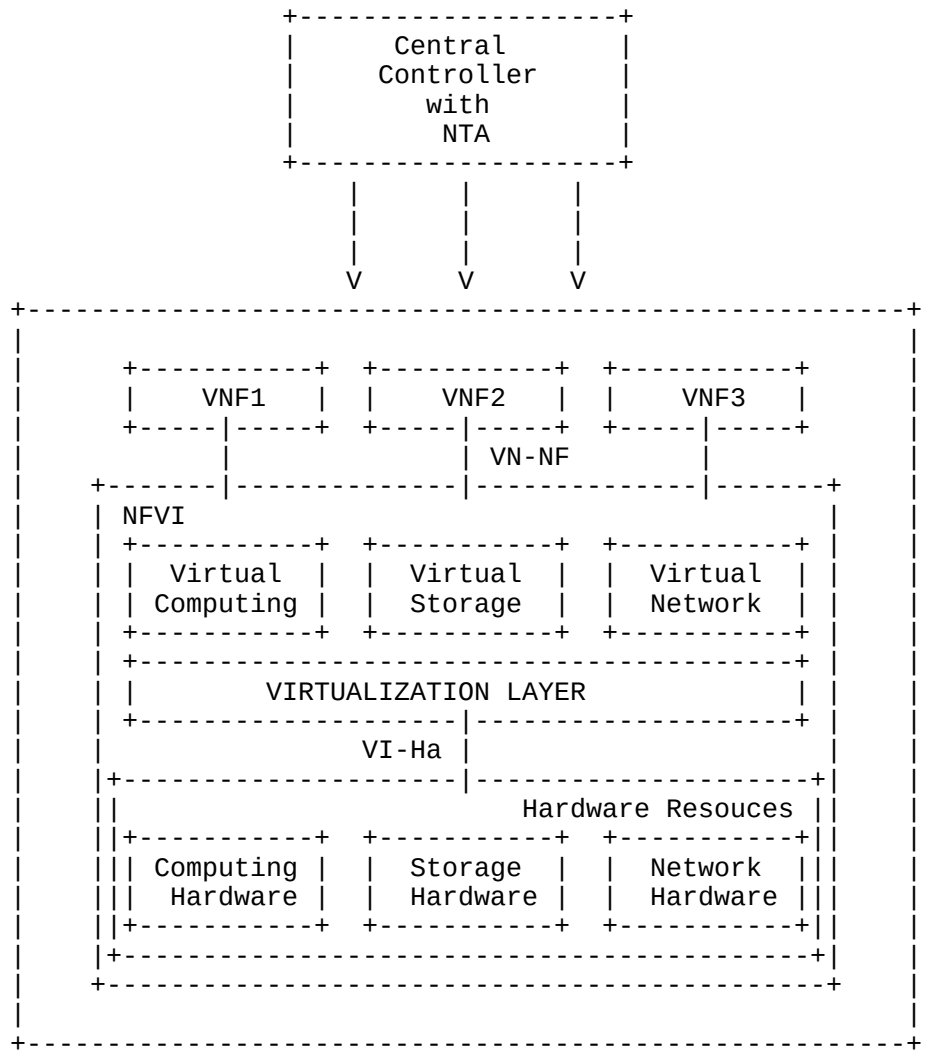


Figure 2 NAI in Multi-layer NFV Framework

For the virtualization layer, CPU performance, memory usage, interface bandwidth and other KPI indicators can be monitored. At the same time resource occupancy and the life cycle of NVF software process can also be monitored. Through the NAI, the relevant statistical data in multiple levels can be analyzed and the models can be setup to locate the root cause for the possible fault in the multi-layer environment.

3.4. Data Center Network Use Cases

Traditionally, data center networks have comprised a large number of switches and routers that direct traffic based on the limited view of each device. With help of SDN/NFV the data center networks are more agile and dynamic to changing usage and traffic patterns. The real-time traffic data and usage can be used to make the data center management and operations intelligent.

Various protocols such as sFLOW, IPFIX could be used to get the port statistics as well as traffic sampling. Over time this information can help build the traffic usage models on a per port and per flow basis. With historical data as the base the NTA engine can predict the traffic usage and make necessary instructions to the SDN controller or NFV orchestrator. These instructions could be reroute a flow to avoid a congested port or scale-in another switch to share load based on the predicted traffic demand.

The NTA engine should find correlation between the various network data to build models and predict the impact of network events, congestions, network utilization patters etc. Further NTA could detect anomalies based on the historical patterns and help in root cause analysis. The policy framework can be enhanced to consider the analytics.

NTA engine could also get the usage and health information from the Host (servers). Correlation between this information with the information received from network could help in finding security flows and anomalies when the information does not match.

3.4.1. Service Function Chaining

This sub section introduces how to apply NAI to SFC scenario to intelligently reroute/re-optimize the service chains; increase utilization for both Service Functions(SF) and network; intelligent selection of the Service Function Path (SFP) based on data traffic trends.

As per [RFC7665], Service function chaining (SFC) enables the creation of composite (network), services that consist of an ordered set of SFs that must be applied for specific treatment of received packets and/or frames and/or flows selected as a result of classification The SFs of chain are connected using a service function forwarder (SFF), which is responsible for forwarding traffic to one or more connected SFs according to information carried in the SFC encapsulation, as well as handling traffic coming back from the SF.

The various network telemetry information like delay, jitter, packet loss from the network and the CPU/memory usage utilizations from the SFs, can be collected using sFLOW/gRPC protocol and stored in persistent data repository. The analytics component in NTA can use stored data to build statistics models to predict the impact on various Service Function Paths due to network events, traffic and state of the SFPs and instruct the SDN controller to take necessary actions SDN controller can calculate new paths/reroute the SFC path to avoid congested Ports/SFFs or overloaded SFs. This correlation of application analytics from the SFs and the network analytics from the SFFs could enhance the intelligent management of the service chains for the operators.

The usage and traffic pattern over time can help increase the utilization of SF as well as the underlay network.

4. Contributors

The following people have substantially contributed to the usecases of NAI:

Lizhao You
Huawei
Email: youlizhao@huawei.com

Kalyankumar Asangi
Huawei
Email: kalyana@huawei.com

5. Security Considerations

TBD

6. IANA Considerations

This document has no actions for IANA.

7. Acknowledgement

Thanks to Li Zhenbin and Liu Shucheng for their comments and contribution.

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

[I-D.li-rtgwg-network-ai-arch] Li, Z. and J. Zhang, "An Architecture of Network Artificial Intelligence(NAI)", draft-li-rtgwg-network-ai-arch-00 (work in progress), October 2016.

[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>.

[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>.

[RFC7752] Gredler, H., Ed., Medved, J., Previdi, S., Farrel, A., and S. Ray, "North-Bound Distribution of Link-State and Traffic Engineering (TE) Information Using BGP", RFC 7752, DOI 10.17487/RFC7752, March 2016, <http://www.rfc-editor.org/info/rfc7752>.

[RFC7854] Scudder, J., Ed., Fernando, R., and S. Stuart, "BGP Monitoring Protocol (BMP)", RFC 7854, DOI 10.17487/RFC7854, June 2016, <http://www.rfc-editor.org/info/rfc7854>.

Authors' Addresses

Yi Zheng
China Unicom
No.9, Shouti Nanlu, Haidian District
Beijing 100048
China

Email: zhengyi39@chinaunicom.cn

Xu Shiping
Huawei Technologies
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095
P.R. China

Email: xushiping7@huawei.com

Dhruv Dhody
Huawei Technologies
Divyashree Techno Park, Whitefield
Bangalore, Karnataka 560066
India

Email: dhruv.ietf@gmail.com