
Workgroup: RIFT
Internet-Draft: draft-head-rift-auto-fr-01
Published: 27 June 2022
Intended Status: Standards Track
Expires: 29 December 2022
Authors: J. Head, Ed. T. Przygienda C. Barth
Juniper Networks Juniper Networks Juniper Networks

RIFT Auto-Flood Reflection

Abstract

This document specifies procedures where RIFT can automatically provision IS-IS Flood Reflection topologies by leveraging its native no-touch ZTP architecture.

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 29 December 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction
 - 1.1. Requirements Language
2. Design Considerations
3. Auto-FR Device Roles
 - 3.1. All Participating Nodes
 - 3.2. Flood Reflectors
 - 3.3. Flood Reflectors Clients
4. Auto-FR Variable Derivation
 - 4.1. RIFT System ID
 - 4.2. Auto-FR Version
 - 4.3. Flood Reflection Cluster ID
 - 4.4. Flood Reflection Preference
 - 4.5. IS-IS System ID
 - 4.6. IS-IS NET Address
 - 4.7. Loopback Address
 - 4.7.1. Leaf Nodes as Flood Reflector Clients
 - 4.7.2. ToF Nodes as Flood Reflectors
 - 4.7.2.1. Flood Reflector Election Procedures
5. RIFT Requirements
 - 5.1. RIFT FSM / LIE Validation Requirements
 - 5.2. RIFT Node-TIE Advertisements
6. Operational Considerations
 - 6.1. RIFT Underlay and IS-IS Flood Reflection Topology
 - 6.2. Auto-FR Analytics
 - 6.2.1. Auto-FR Analytics (Global) Key/Value Pair
7. IANA Considerations
8. Security Considerations
9. Acknowledgements

10. References

10.1. Normative References

Appendix A. Thrift Models

A.1. common.thrift

A.2. encoding.thrift

A.3. auto_flood_reflection_kv.thrift

Appendix B. Auto-FR Variable Derivation

Authors' Addresses

1. Introduction

[RIFT] is a protocol that focuses heavily on operational simplicity. It natively supports Zero Touch Provisioning (ZTP) functionality that allows each node to automatically derive its place in the topology and configure itself accordingly when properly cabled as a Clos, Fat-Tree, or other similarly structured variant.

[IS-IS Flood Reflection \[IS-IS-FR\]](#) is a mechanism that enables flat single-area Level 2 IS-IS topologies to scale well beyond their typical properties when deployed in similar topological structures by:

1. Reducing the number of required links and adjacencies.
2. Reducing the size of the Link-State Database.
3. Reducing the amount of flooding.
4. Reducing the number of SPF computations.
5. Reducing the maximum SPF computation time.

RIFT Auto-Flood Reflection (Auto-FR) combines these technologies by using RIFT's ZTP functionality in order to automatically provision IS-IS Flood Reflection topologies in a completely distributed fashion.

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. Design Considerations

IS-IS Flood Reflection operates using Flood Reflectors at the top of the fabric and Flood Reflector Clients at the bottom of the fabric. Any nodes in the middle are not required to support Flood Reflection functionality, nor do they need to support Auto-FR.

Nodes in a Flood Reflection topology require specific variables for deployment. For example, a Cluster ID that is unique to the particular fabric or loopback addresses that are unique to a particular node. RIFT has enough topological information to derive these variables with the appropriate scope in a distributed fashion automatically.

Once the Flood Reflection topology is built, RIFT Key-Value TIEs can be used to distribute operational state information to allow for cluster-wide validation without any additional tooling.

3. Auto-FR Device Roles

Auto-FR requires that each node understands its given role within the scope of the Flood Reflection deployment, so each node derives the necessary variables and resulting configuration.

3.1. All Participating Nodes

Not all nodes have to participate in Auto-FR, however, if a node does assume an Auto-FR role, it MUST derive the following variables:

Flood Reflection Cluster ID

The Flood Reflection Cluster ID is used to distinguish reflection domains (similar to the Cluster ID use in BGP Route Reflection).

IPv6 Loopback Address

Unique IPv6 loopback address.

IS-IS System ID

The IS-IS System Identifier used in deriving the IS-IS NET Address.

IS-IS NET Address

The IS-IS NET Address used to uniquely identify an IS-IS node.

3.2. Flood Reflectors

This section defines an Auto-FR role whereby some ToF (Top-of-Fabric) nodes act as IS-IS Flood Reflectors. It is expected that Flood Reflectors will establish Level 2 IS-IS adjacencies with Flood Reflector Clients in the same area, in the same fabric. The typical Flood Reflector requirements do not change, however, determining which specific values to use requires further consideration.

ToF nodes performing Flood Reflector functionality MUST derive the following variables:

IPv6 Flood Reflector Loopback Address

Unique IPv6 loopback address.

3.3. Flood Reflectors Clients

Although no specific variables for Flood Reflector Clients are described at this time, the generic role is specified as a placeholder for future enhancements.

Future Consideration

Future Consideration

4. Auto-FR Variable Derivation

As previously mentioned, not all nodes are required to derive all variables in a network. For example, spine nodes may only be required for transit traffic and not need to support Auto-FR at all. All variables are derived from RIFT's FSM or ZTP mechanism, so no additional flooding other than RIFT's typical flooding is required.

It is also important to mention that all variable derivation is in some way based on the RIFT System

ID and/or Cluster ID and MUST comply precisely with calculation methods specified in the [Auto-FR Variable Derivation](#) section to allow interoperability between different implementations. All necessary foundational code elements are also mentioned there.

4.1. RIFT System ID

The 64-bit RIFT System ID that uniquely identifies a node as defined in [\[RIFT\]](#). This not derived specifically for Auto-FR, but for all RIFT nodes and is used in the derivation procedures described in this section.

4.2. Auto-FR Version

This section describes extensions to both the RIFT LIE and Node-TIE packet schemas in the form of a 16-bit value that identifies the Auto-FR Version. Auto-FR capable nodes MUST support this extension, but MAY choose not to advertise it in LIEs and Node-TIEs when Auto-FR is not being utilized.

The complete encoding.thrift schema documented in [\[RIFT\]](#) describes both major and minor protocol/schema versions. Auto-FR Version calculation is done by multiplying a static value of 256 by the major version then adding the minor version, that is to say - $256 * \text{MAJOR} + \text{MINOR}$.

This section also describes an extension to the NodeCapabilities schema indicating whether or not a node supports Auto-FR.

The [appendix \(Appendix A\)](#) details necessary changes to the LIEPacket, NodeTIEElement, and NodeCapabilities Thrift schemas.

4.3. Flood Reflection Cluster ID

This section describes extensions to both the RIFT LIE and Node-TIE packet schemas in the form of a 32-bit value that identifies the Auto-FR Cluster ID. Auto-FR capable nodes MUST support this extension, but MAY choose not to advertise it in LIEs and Node-TIEs when Auto-FR is not being utilized.

Deployments using more than one Auto-FR cluster MUST use different Cluster IDs. Failure to do so may cause sub-optimal routing as L1/L2 nodes from different clusters would belong to the same subnet.

A Cluster ID with a value of 0 is considered invalid and MUST NOT be used for any purpose.

The [appendix \(Appendix A\)](#) details necessary changes to the LIEPacket and NodeTIEElement Thrift schemas.

4.4. Flood Reflection Preference

This section describes extensions to the Node-TIE packet schema in the form of a 32-bit value that indicates a Flood Reflection Preference value to be used during Flood Reflector election procedures with the higher value being preferred. Auto-FR capable nodes MUST support this extension.

The [appendix \(Appendix A\)](#) details necessary changes to the NodeTIEElement Thrift schemas.

4.5. IS-IS System ID

Auto-FR nodes MUST derive a unique 8-byte IS-IS System ID for use in deriving the IS-IS NET Address. Calculation is done using the 8-byte RIFT System ID and 4-byte Cluster ID.

In order for nodes to derive an IS-IS System ID, the following algorithms are required - [auto_fr_cidsid2isissid \(Figure 9\)](#) and [auto_fr_v6hash \(Figure 14\)](#).

4.6. IS-IS NET Address

Auto-FR nodes MUST derive a unique 10-byte IS-IS NET (Network Entity Title) Address to uniquely identify itself within the Flood Reflection topology. The 1st byte (which indicates the AFI) MUST have a value of "49". The last byte (i.e. the NSEL) MUST have a value of 0. Remaining calculation is done using the 8-byte RIFT System ID and 4-byte Cluster ID.

In order for nodes to derive an IS-IS NET Address, the following algorithms are required - [auto_fr_cidsid2isisnet \(Figure 8\)](#) and [auto_fr_cidsid2isissid \(Figure 9\)](#).

4.7. Loopback Address

Auto-FR nodes MUST derive a ULA-scoped IPv6 loopback address to be used in IS-IS. Calculation is done using the 6-bytes of reserved ULA space, the 4-byte Cluster ID, and the node's 8-byte RIFT System ID. Derivation of the IS-IS System ID varies slightly depending upon the node's location/role in the fabric and will be described in subsequent sections.

4.7.1. Leaf Nodes as Flood Reflector Clients

Leaf Nodes acting as Flood Reflector Clients MUST derive their loopback address according to the specific section describing the algorithm. Calculation is done using the 6-bytes of reserved ULA space, the 4-byte Cluster ID, and the 8-byte RIFT System ID.

In order for leaf nodes to derive IPv6 loopbacks, the following algorithms are required - [auto_fr_cidsv6loopback](#) (Figure 11) and [auto_fr_v6prefixcidsv2loopback](#) (Figure 15).

IPv4 addresses MAY be supported, but it should be noted that they have a higher likelihood of collision. The appendix contains the required [auto_fr_cidsv4loopback](#) (Figure 10) algorithm to support IPv4 loopback derivation.

4.7.2. ToF Nodes as Flood Reflectors

ToF nodes acting as Flood Reflectors MUST derive their loopback address according to the specific section describing the algorithm. Calculation is done using the 6-bytes of reserved ULA space, the 4-byte Cluster ID, and the 8-byte RIFT System ID.

In order for ToF nodes to derive IPv6 loopbacks, the following algorithms are required - [auto_fr_cidsv6loopback](#) (Figure 11), [auto_fr_v6prefixcidsv2loopback](#) (Figure 15), and [auto_fr_cidfrpref2frloopback](#) (Figure 7).

IPv4 addresses MAY be supported, but it should be noted that they have a higher likelihood of collision. The appendix contains the required [auto_fr_cidsv4loopback](#) (Figure 10) algorithm to support IPv4 loopback derivation.

4.7.2.1. Flood Reflector Election Procedures

Each ToF performs the election independently based on the RIFT System IDs and a Flood Reflection preference value of other ToF nodes in the fabric obtained via southbound reflection. The Flood Reflector election procedures are defined as follows:

1. Highest System ID with the highest preference.
2. Lowest System ID with the highest preference.
3. 2nd highest System ID with the 2nd highest preference.
4. etc.

This ordering is necessary to prevent a single node with either the highest or lowest System ID from triggering changes to flood reflector loopback addresses as it would result in all IS-IS adjacencies flapping.

For example, if ToF01 (System ID: 002c6af5a281c000 / FR Preference: 100) and ToF02 (System ID: 002c6bf5788fc000 / FR Preference: 1) went through the election process, ToF02 would be elected due to it having the highest System ID. If a ToF determines that it is elected as a Flood Reflector, it uses the knowledge of its position in the list to derive Flood Reflector IPv6 loopback address.

A topology MUST elect at least 1 ToF node as an IS-IS Flood Reflector, but SHOULD elect 3. The election process varies depending upon whether or not the topology is comprised of a single plane or multiple planes. The multiplane election procedure will be described in a future version of this document.

The algorithm shown in "auto_fr_sids2frs" (Figure 12) is required to perform the Flood Reflector election procedures.

5. RIFT Requirements

5.1. RIFT FSM / LIE Validation Requirements

RIFT FSM adjacency rules MUST consider [Auto-FR Version \(Section 4.2\)](#) and [Auto-FR Cluster ID \(Section 4.3\)](#) values so that nodes that do not support Auto-FR can interoperate with nodes that do. The LIE validation is extended with the following clause and if it is not met, miscabbling should be declared:

```
(if auto_flood_reflection_version is not advertised by either node OR
  if auto_flood_reflection_version is identical on both nodes)
  AND
(auto_flood_reflection_cluster_id is not advertised by either node OR
  auto_flood_reflection_cluster_id is identical on both nodes)
```

5.2. RIFT Node-TIE Advertisements

All nodes utilizing Auto-FR MUST advertise their [Auto-FR Version \(Section 4.2\)](#), [Flood Reflection Cluster ID \(Section 4.3\)](#), and [Flood Reflection Preference \(Section 4.4\)](#) values in at least one Node-TIE in each direction (i.e. North and South).

6. Operational Considerations

To fully realize the benefits of Auto-FR, it may help to describe the high-level method. Simply put, RIFT automatically provisions the underlay and Auto-FR provisions the Flood Reflection topology. The goal of this section is to draw simple lines between general fabric concepts, RIFT, and Auto-FR and how they fit into current network designs and practices.

This section also describes a set of optional [Key-Value TIEs \[RIFT-KV\]](#) that leverages the variables that have already been derived to provide further operational enhancement to the operator.

6.1. RIFT Underlay and IS-IS Flood Reflection Topology

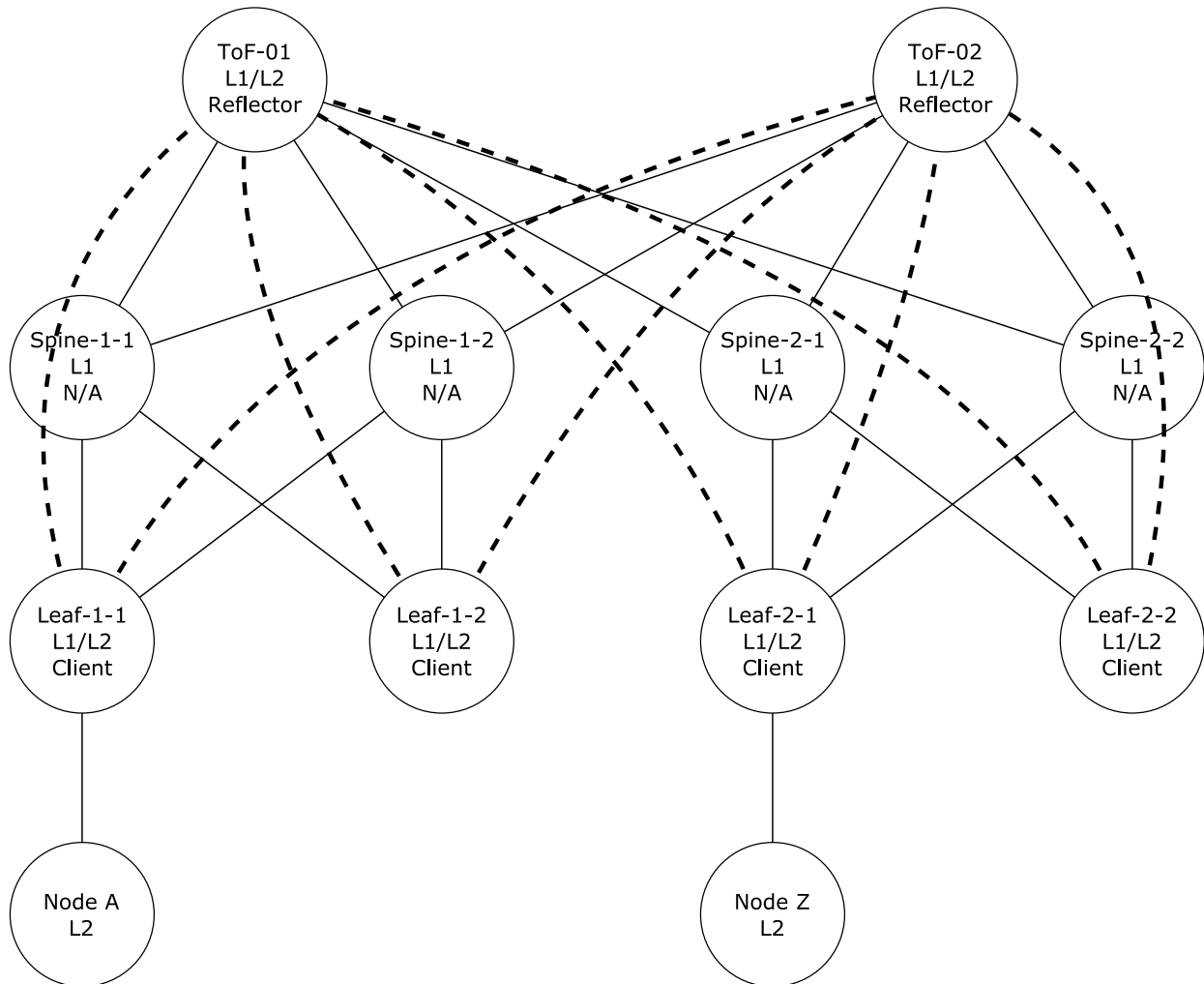


Figure 1: Auto-FR Example Topology

Figure 1 illustrates a typical 5-stage Clos IP fabric. Each node is named and labelled in such a way that conveys:

1. The node's generic placement within the context of the RIFT underlay
2. The node's level(s) within the IS-IS area.
3. The node's role within the IS-IS Flood Reflection topology.

Table 1 should help further align these concepts.

RIFT Placement	IS-IS Level	IS-IS FR Role
ToF Nodes	L1/L2	Flood Reflector

RIFT Placement	IS-IS Level	IS-IS FR Role
Spine Nodes	L1	N/A
Leaf Nodes	L1/L2	Flood Reflector Client

Table 1: Role Associations

Connections between various nodes can be understood in two different ways:

1. Lines between ToF and leaf nodes are Level 2 IS-IS Flood Reflection adjacencies.
2. Lines between spine and leaf are part of the physically connected underlay.
3. Lines between ToF and spine are part of the physically connected underlay.

It is important to remember that Auto-FR is not altering the way in which IS-IS Flood Reflection operates in any way, it simply takes existing deployment scenarios and simplifies the provisioning process.

6.2. Auto-FR Analytics

Leaf nodes MAY optionally advertise analytics information about the Auto-FR fabric to ToF nodes using [RIFT Key-Value TIEs \[RIFT-KV\]](#). This may be helpful in that validation and troubleshooting activities can be performed on the ToF nodes rather than manually verifying the state separately on multiple leaf nodes.

6.2.1. Auto-FR Analytics (Global) Key/Value Pair

This Key/Value pair describes node level information within the context of the Flood Reflection topology. The RIFT System ID of the advertising leaf node MUST be used to differentiate the node among other nodes in the fabric.

The Auto-FR Analytics (Global) Key/Value pair MUST be advertised with the 3rd and 4th bytes of the Key Identifier consisting of all 0s.

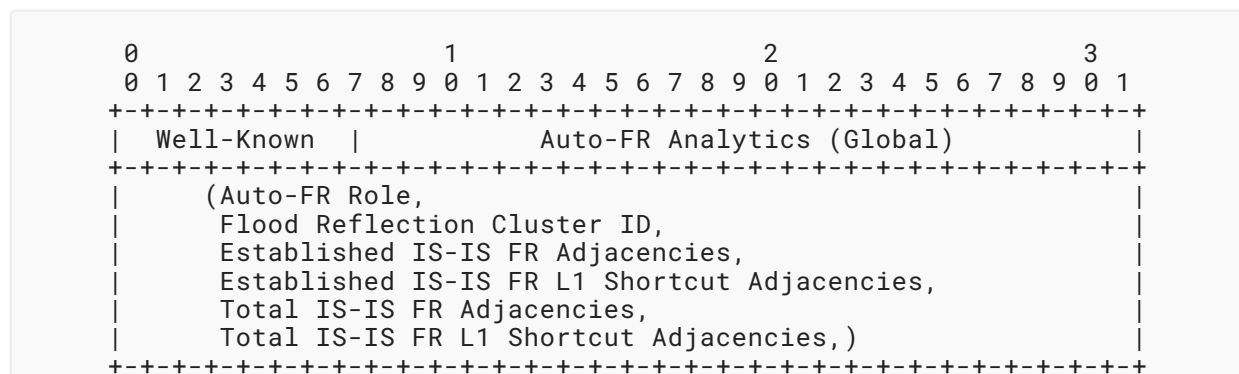


Figure 2: Auto-FR Global Key/Value Pair

where:**Auto-FR Role:**

A REQUIRED value indicating the node's Auto-FR role within the fabric.

0: Illegal value, MUST NOT be used.

1: Auto-FR Flood Reflector Client

2: Auto-FR Flood Reflector

Auto-FR Cluster ID

A REQUIRED 32-bit integer indicating the Auto-FR Cluster ID of the local node.

Established IS-IS Flood Reflector Adjacency Count:

A RECOMMENDED 16-bit integer indicating the number of IS-IS Level 2 Flood Reflector adjacencies in the "Up" state on the local node.

Functional IS-IS Level 1 Shortcut Count

A RECOMMENDED 16-bit integer indicating the number of functional IS-IS Level 1 "shortcuts" on the local node.

Total IS-IS Flood Reflector Adjacency Count:

A RECOMMENDED 16-bit integer indicating the total number of IS-IS Level 2 Flood Reflector adjacencies on the local node regardless of state.

Total IS-IS Level 1 Shortcut Count

A RECOMMENDED 16-bit integer indicating the total number of IS-IS Level 1 "shortcuts" the local node regardless of state.

Implementations leveraging Thrift for Key-Value functionality SHOULD refer to the [auto_flood_reflection_kv.thrift](#) (Appendix A.3) schema in the appendix.

7. IANA Considerations

This section requests the following suggested values from the RIFT Well-Known Key-Type Registry.

Value	Key-Identifier	Description	Status/ Reference
5	Auto-FR Analytics (Global)	Key/Value pair containing operational state of a Flood Reflector Client node.	This document.

Table 2: Auto-FR Suggested Value(s) for RIFT Well-Known Key-Type Registry

8. Security Considerations

This document introduces no new security concerns to RIFT or other specifications referenced in this document as RIFT natively secures LIE and TIE packets as described in [RIFT].

9. Acknowledgements

This section will be used to acknowledge major contributors.

10. References

10.1. Normative References

- [IS-IS-FR] Przygienda, A., Bowers, C., Lee, Y., Sharma, A., and R. White, "IS-IS Flood Reflection", Work in Progress, draft-ietf-lsr-isis-flood-reflection-07, November 2021.
- [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>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RIFT] Przygienda, T., Sharma, A., Thubert, P., Rijsman, B., and D. Afanasiev, "RIFT: Routing in Fat Trees", Work in Progress, draft-ietf-rift-rift-15, December 2021.
- [RIFT-KV] Head, J. and T. Przygienda, "RIFT Key/Value Structure and Registry", Work in Progress, draft-ietf-rift-kv-registry-01, June 2022.

Appendix A. Thrift Models

This section contains the normative Thrift models required to support Auto-FR. Per the main [RIFT] specification, all signed values MUST be interpreted as unsigned values.

A.1. common.thrift

This section specifies extensions to RIFT common.thrift model.

These extensions are REQUIRED in order to support Auto-FR.

```
...
enum AutoFRModel {
    TunnelMode      = 0,
    NoTunnelMode    = 1,
}

const AutoFRModel default_autofr_model = AutoFRModel.TunnelMode

typedef i32          FloodReflectionClusterIDType

const FloodReflectionClusterIDType  IllegalClusterID = 0
const FloodReflectionClusterIDType  DefaultClusterID = 1

/// preference to become FR, higher is better
typedef i32          FloodReflectionPreferenceType

const FloodReflectionPreferenceType MinFloodReflectionPreference = 0

...
```

Figure 3: RIFT Auto-FR: *common.thrift*

A.2. encoding.thrift

This section specifies extensions to RIFT encoding.thrift model.

These extensions are REQUIRED in order to support Auto-FR.

```
struct NodeCapabilities {
  ...
  /** indicates whether auto-flood-reflection feature is implemented on
  this node (but not necessarily enabled). */
  20: optional bool
  auto_flood_reflection_support = false;
  ...
}

struct LIEPacket {
  ...
  /** It provides optional version of FR ZTP as 256 * MAJOR + MINOR,
  indicates support for auto FR */
  40: optional i16
  auto_flood_reflection_version;

  41: optional common.FloodReflectionClusterIDType
  auto_flood_reflection_cluster_id;
  ...
}

struct NodeTIEElement {
  ...
  /** All Auto FR elements MUST be present in at least one TIE in each
  direction if auto FR is running. */
  /** It provides optional version of FR ZTP as 256 * MAJOR + MINOR,
  indicates support for auto FR */
  30: optional i16
  auto_flood_reflection_version;
  /** cluster ID of Auto FR */
  31: optional common.FloodReflectionClusterIDType
  auto_flood_reflection_cluster_id;
  /** preference to become FR */
  32: optional common.FloodReflectionPreferenceType
  auto_flood_reflection_preference;
  ...
}
```

Figure 4: RIFT Auto-FR: *encoding.thrift*

A.3. auto_flood_reflection_kv.thrift

This section defines `auto_flood_reflection_kv.thrift` as a method of supporting Auto-FR analytics functionality.

```
include "common.thrift"

namespace py auto_flood_reflection_kv
namespace rs models

const i8          AutoFRWellKnownKeyType = 2
typedef i16       AutoFRCounterType
typedef i32       AutoFRLongCounterType

const i8          GlobalAutoFRTelemetryKV = 5

/** We don't need the full role structure, only an indication of the
node's basic role */
enum AutoFRRole {
    ILLEGAL          = 0,
    auto_fr_leaf     = 1,
    auto_fr_reflector = 2,
}

/** Per the according RIFT draft the key comes from the well known space.
Part of the key is used as Fabric-ID.

    1st   byte  MUST be = "Well-Known"
    2nd   byte  MUST be = "Auto-FR Analytics (Global) KV",
    3rd/4th bytes MUST be = all 0s
*/
struct AutoFRTelemetryGlobalKV {
    /** Only values that the ToF cannot derive itself should be flooded.
    */
    1: required  set<AutoFRRole>
    auto_fr_roles,

    2: required  common.FloodReflectionClusterIDType      cluster_id,

    3: optional  AutoFRCounterType
    established_isis_fr_adjacencies_count,

    4: optional  AutoFRCounterType
    established_isis_l1_shortcut_adjacencies_count,

    5: optional  AutoFRCounterType
    total_isis_fr_adjacencies_count,

    6: optional  AutoFRCounterType
    total_isis_l1_shortcut_adjacencies_count,
}
```

Figure 5: RIFT Auto-FR: *auto_flood_reflection_kv.thrift*

Appendix B. Auto-FR Variable Derivation

This section contains the normative variable derivation algorithms that are REQUIRED to support Auto-FR.

```
/// indicates how many FRs we're computing in AUTO FR
pub const MAX_AUTO_FR_FRS: usize = 3;

/// indicates the cluster has no ID, used in computations to omit effects
of cluster ID
pub const NO_CLUSTER_ID: FloodReflectionClusterIDType = 0;

/// unique v6 prefix for all nodes starts with this
pub fn auto_fr_v6pref(cid: FloodReflectionClusterIDType) -> String {
    format!("FD00::{:04X}:B1", cid)
}

/// how many bytes in a v6pref for different purposes
pub const AUTO_FR_V6PREFLEN: usize = 8 * 5;

/// unique v6 prefix for flood reflector purposes starts like this
pub fn auto_fr_v6frpref(cid: FloodReflectionClusterIDType) -> String {
    format!("FD00::{:04X}:B2", cid)
}

/// unique v4 prefix for IRB purposes
pub const AUTO_FR_V4LOOPBACKNET: u8 = 10;
pub const AUTO_FR_V4LOOPBACKMASK : usize = 8;
```

Figure 6: RIFT Auto-FR: *auto_fr_const_structs_types*

```
/// auto FR V6 loopback for FRs
pub fn auto_fr_cidfrpref2frloopback(cid: FloodReflectionClusterIDType,
                                     preference: u8) -> Result<Ipv6Addr,
                                     ServiceErrorType> {
    auto_fr_v6prefixcidsid2loopback(&auto_fr_v6frpref(cid), cid, (1 +
    preference) as _)
}
```

Figure 7: RIFT Auto-FR: *auto_fr_cidfrpref2frloopback*


```
pub fn auto_fr_cidssid2isisnet(cid: FloodReflectionClusterIDType, sid:
UnsignedSystemID) -> Vec<u8> {
    let mut r = vec![0x49]; // magic AFI

    // area ID derived from cluster ID
    r.extend(&cid.to_ne_bytes().iter().fold(0x77u16,
|prev, val| (prev ^
(val.rotate_right(4) as u16))).to_ne_bytes());
    // ISIS ID derived from system ID + cid/sid
    r.extend(auto_fr_cidssid2isissid(cid, sid).into_iter());
    // selector non v-node
    r.push(0);

    r
}
```

Figure 8: RIFT Auto-FR: *auto_fr_cidssid2isisnet*

```
/// ISIS system ID derivation
pub fn auto_fr_cidssid2isissid(
cid: FloodReflectionClusterIDType,
sid: UnsignedSystemID) -> Vec<u8> {

    let sb = auto_fr_v6hash(cid, sid);

    vec![sb[0],
sb[1],
sb[2],
sb[3],
sb[4] ^ sb[5].rotate_right(4),
sb[6] ^ sb[7].rotate_right(4),
]
}
```

Figure 9: RIFT Auto-FR: *auto_fr_cidssid2isissid*

```
/// v4 loopback address derivation for every node in auto-fr, returns
address and
/// subnet mask length.
pub fn auto_fr_cidssid2v4loopback(cid: FloodReflectionClusterIDType, sid:
UnsignedSystemID) -> (IPv4Address, u8) {
    let mut derived = sid.to_ne_bytes().iter()
        .fold(0 as IPv4Address, |p, e| (p << 4) ^ (*e as IPv4Address));
    derived ^= cid as IPv4Address;
    // use the byte we loose for entropy
    derived ^= derived >> (32 - AUTO_FR_V4LOOPBACKMASK);
    // and sanitize for loopback range, we nuke 8 bits out
    derived &= (!U32MASKS[AUTO_FR_V4LOOPBACKMASK]) as IPv4Address;

    let m = ((AUTO_FR_V4LOOPBACKNET as IPv4Address) << (32 -
AUTO_FR_V4LOOPBACKMASK)) | derived;
    (m as _, AUTO_FR_V4LOOPBACKMASK as _)
}
```

Figure 10: RIFT Auto-FR: *auto_fr_cidssid2v4loopback*

```
/// V6 loopback derivation for every node in auto fr
pub fn auto_fr_cidssidv6loopback(cid: FloodReflectionClusterIDType,
                                sid: UnsignedSystemID) ->
Result<Ipv6Addr, ServiceErrorType> {
    auto_fr_v6prefixcidssid2loopback(&auto_fr_v6pref(cid), cid, sid)
}
```

Figure 11: RIFT Auto-FR: *auto_fr_cidssidv6loopback*

```

/// Function sorts vector of system IDs first,
/// Followed by a shuffle taking largest/smallest/2nd largest/2nd
/// smallest.
/// Preference is used to take the according subsets to run this
/// algorithm
/// sequentially.
pub(crate) fn auto_fr_sids2frs(mut v: Vec<(FloodReflectionPreferenceType,
                                         UnsignedSystemID)>)
    -> Vec<UnsignedSystemID> {
    v.par_sort_by(|(p1, s1),
                  (p2, s2)|
        match p2.cmp(p1) {
            Ordering::Equal => s2.cmp(s1),
            e @ _ => e
        });

    let mut elected = vec![];

    while elected.len() < MAX_AUTO_FR_FRS && !v.is_empty() {
        let pref = (&v[0]).0;

        let mut splitat = 0;
        while splitat < v.len() && (&v[splitat]).0 == pref {
            splitat += 1;
        }

        let mut so = v.split_off(splitat);
        std::mem::swap(&mut v, &mut so);

        let mut mixed = if so.len() > 2 {
            let mut s = so.split_off(so.len() / 2);
            s.reverse();
            interleave(so.into_iter(), s.into_iter())
                .collect::<Vec<_>>()
        } else {
            so
        };

        elected.extend(mixed.drain(..))
    }

    elected.drain(..).map(|(_, sid)| sid).collect()
}

```

Figure 12: RIFT Auto-FR: *auto_fr_sids2frs*

```

pub(crate) fn auto_fr_v62octets(a: Ipv6Addr) -> Vec<u8> {
    a.octets().iter().cloned().collect()
}

```

Figure 13: RIFT Auto-FR: *auto_fr_v62octets*

```

/// generic bytes derivation used for different purposes
pub fn auto_fr_v6hash(cid: FloodReflectionClusterIDType, sid:
UnsignedSystemID)
    -> [u8; 8] {
    let sub = (cid as UnsignedSystemID) ^ sid.rotate_right(8);

    sub.to_ne_bytes()
}

```

Figure 14: RIFT Auto-FR: *auto_fr_v6hash*

```

/// local address with encoded cluster ID and system ID for collision
free identifiers. Basis
/// for several different prefixes.
pub fn auto_fr_v6prefixcidsid2loopback(v6pref: &str, cid:
FloodReflectionClusterIDType,
    sid: UnsignedSystemID) ->
Result<Ipv6Addr, ServiceErrorType> {
    assert!(cid != ILLEGAL_CLUSTER_I_D);
    let a = format!("{}",
        cid,
        v6pref,
        sid.to_ne_bytes()
            .iter()
            .chunks(2)
            .into_iter()
            .map(|chunk|
                chunk.fold(0u16, |v, n| (v << 8) | *n as
u16))
            .map(|v| format!("{:04X}", v))
            .collect::<Vec<_>>()
            .into_iter()
            .join(":"));
    Ipv6Addr::from_str(&a)
        .map_err(|_| ServiceErrorType::INTERNALRIFTERROR)
}

```

Figure 15: RIFT Auto-FR: *auto_fr_v6prefixcidsid2loopback*

```
/// cluster prefixes derived instead of advertising default on the
cluster to allow
/// for default route on ToF or leaves
pub fn auto_fr_cid2cluster_prefixes(cid: FloodReflectionClusterIDType) ->
Result<Vec<IPPrefixType>, ServiceErrorType> {
    vec![
        (auto_fr_cidsidv6loopback(cid, ILLEGAL_SYSTEM_I_D as _),
AUTO_FR_V6PREFLEN),
        (auto_fr_cidfrpref2frloopback(cid, 0 as _), AUTO_FR_V6PREFLEN),
    ]
    .into_iter()
    .map(|(p, _)|
        match p {
            Ok(_) => Ok(
                IPPrefixType::Ipv6prefix(
                    IPv6PrefixType {
                        address: auto_fr_v62octets(p?),
                        prefixlen: AUTO_FR_V6PREFLEN as _,
                    })),
            Err(e) => Err(e),
        }
    )
    .collect:::<Result<Vec<_>, _>>()
}
```

Figure 16: RIFT Auto-FR: *auto_fr_cid2cluster_prefixes*

Authors' Addresses

Jordan Head (EDITOR)

Juniper Networks
1133 Innovation Way
Sunnyvale, CA
United States of America
Email: jhead@juniper.net

Tony Przygienda

Juniper Networks
1133 Innovation Way
Sunnyvale, CA
United States of America
Email: prz@juniper.net

Colby Barth

Juniper Networks
1133 Innovation Way
Sunnyvale, CA
United States of America
Email: cbarth@juniper.net