Workgroup:	RIFT		
Internet-Draft:	draft-head-rift-auto-fr-00		
Published:	29 December 2021		
Intended Status:	Standards Track		
Expires:	2 July 2022		
Authors:	J. Head, Ed.	T. Przygienda	C. Barth
	Juniper Networks	*Juniper Networks*	*Juniper Networks*

RIFT Auto-FR

Abstract

This document specifies procedures that allow IS-IS Flood Reflection topologies to be fully and automatically provisioned when using RIFT by leveraging RIFT's 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 2 July 2022.

Copyright Notice

Table of Contents

1. Introduction

IS-IS Flood Reflection [IS-IS-FR] is a mechanism that enables large single-area Level 2 IS-IS networks to scale well beyond their typical properties when deployed in Clos/Fat Tree topologies.

[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.

This sense of topological hierarchy makes RIFT well-suited to automatically provision IS-IS Flood Reflection with no additional external interaction using its ZTP functionality.

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 taking part in flood reflection 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 basic validation without 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 Reflector Cluster ID
: The Flood Reflector Cluster ID us to distinguish reflection domains, similar to that of a BGP Cluster ID for route reflection.

IPv6 Loopback Address
: Unique IPv6 loopback address.

ISO System ID
: The ISO System Identifier used in IS-IS.

ISO NET
: The ISO Network Entity Title used in IS-IS.

3.2. Flood Reflectors

This section defines an Auto-FR role whereby some 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 (e.g. a transit spine node may not need to derive any or participate in 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 necessary.

It is also important to mention that all variable derivation is in some way based on 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 foundational code elements are also mentioned there.

4.1. 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 packet and Node-TIE 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.

This section also describes an extension to the Node Capbilities schema indicating whether a node supports Auto-FR.

Auto-FR Version MUST be considered in existing RIFT adjacency FSM rules so that nodes that support Auto-FR can inter-operate with nodes that do not. The LIE validation is extended with the following clause:

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

Miscabling should be declared if this clause is not met.

The appendix (Appendix A) details necessary changes to the RIFT LIE, Node-TIE, and Node Capabilities thrift schema.

4.3. Flood Reflector Cluster ID

This section describes extensions to both the RIFT LIE packet and Node-TIE 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.

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 RIFT LIE and Node-TIE thrift schema.

4.4. 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 System ID. Derivation of the System ID varies slightly depending upon the node's location/role in the fabric and will be described in subsequent sections.

4.4.1. Leaf Nodes as Flood Reflector Clients

Calculation is done using the 6-bytes of reserved ULA space, the 4-byte Cluster ID, and the node's 8-byte System ID.

In order for leaf nodes to derive IPv6 loopbacks, the following algorithms are required - auto_fr_cidsidv6loopback (Figure 11) and auto_fr_v6prefixcidsid2loopback (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_cidsid2v4loopback (Figure 10) algorithm to support IPv4 loopback derivation.

4.4.2. 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 System ID of each elected route reflector.

In order for ToF nodes to derive IPv6 loopbacks, the following algorithms are required - auto_fr_cidsidv6loopback (Figure 11), auto_fr_v6prefixcidsid2loopback (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_cidsid2v4loopback (Figure 10) algorithm to support IPv4 loopback derivation.

A topology MUST elect at least 1 Top-of-Fabric node as an IS-IS flood reflector, but SHOULD elect 3.

4.4.2.1. Flood Reflector Election Procedures

Each ToF performs the election independently based on system IDs of other ToF nodes in the fabric obtained via southbound reflection. The route reflector election procedures are defined as follows:

1. ToF node with the highest System ID.
2. ToF node with the lowest System ID.
3. ToF node with the 2nd highest System ID.
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 two nodes, ToF01 and ToF02 with System IDs 002c6af5a281c000 and 002c6bf5788fc000 respectively, ToF02 would be elected due to it having the highest System ID of the ToFs (002c6bf5788fc000). If a ToF determines that it is elected as flood reflector, it uses the knowledge of its position in the list to derive flood reflector IPv6 loopback address.

The algorithm shown in "auto_fr_sids2frs" (Figure 12) is required to accomplish this.

5. 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.

5.1. RIFT Underlay and IS-IS Flood Reflection Topology

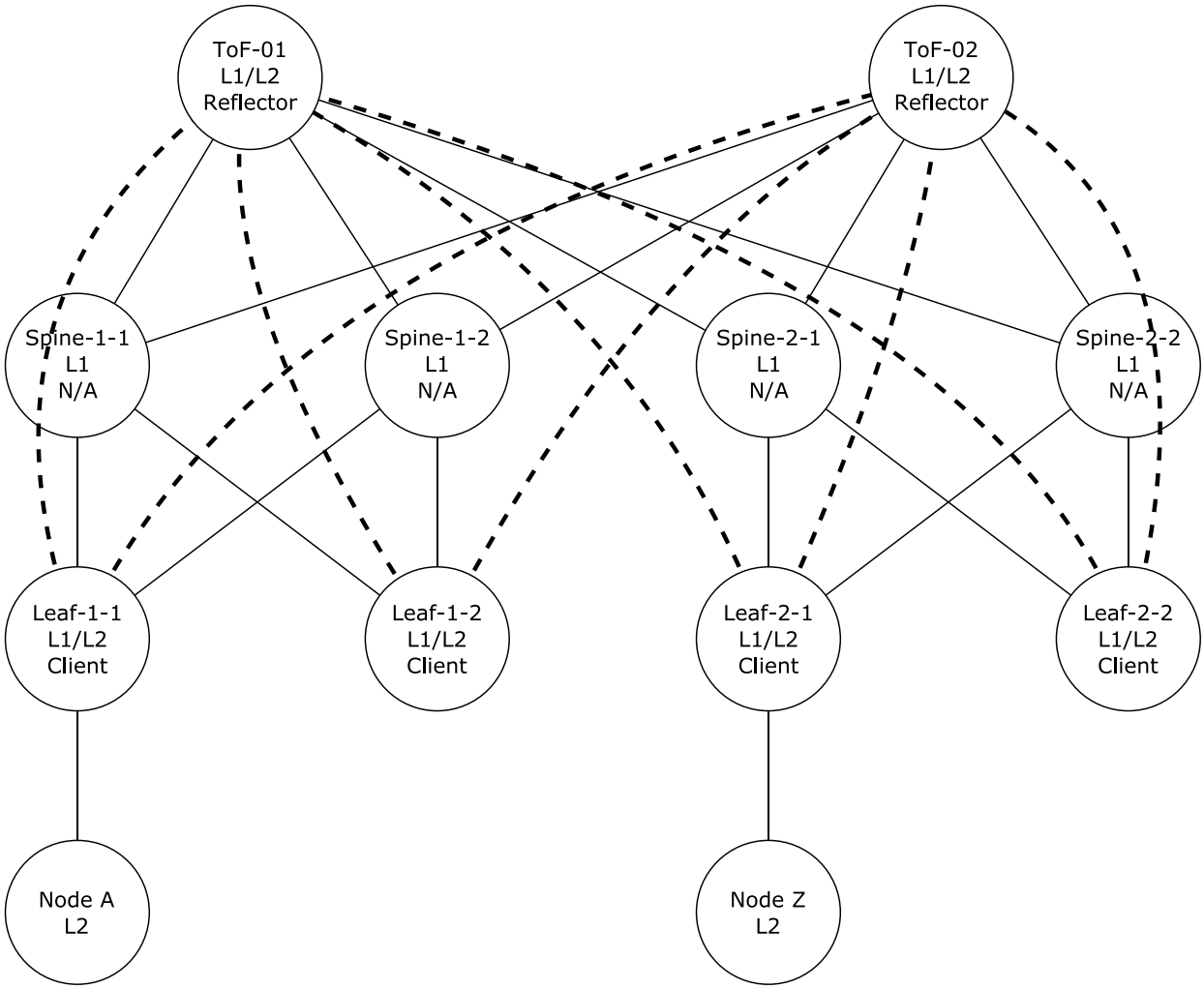

Figure 1: Auto-FR Example Topology

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.

It is important to remember that Auto-FR is not altering anything specific to IS-IS Flood Reflection topologies, it takes existing deployment scenarios and simplies the provisioning process. The topology also illustrates the flood reflection adjacencies between ToF and Leaf nodes.

Table 1 should help further align these concepts.

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

Table 1: Role Associations

5.2. Auto-FR Analytics

Leaf nodes MAY optionally advertise analytics information about the Auto-FR fabric to ToF nodes using RIFT Key-Value TIEs. This may be helpful in that validation and troubleshooting activities can be performed on the ToF nodes.

This section requests suggested values from the RIFT Well-Known Key-Type Registry and describes their use for Auto-FR.

Name	Value	Description
Auto-FR Analytics Global	5	Analytics describing an Auto-FR node within a fabric.

Table 2: Requested RIFT Key Registry Values

The normative Thrift schema can be found in the appendix (Appendix A.3).

5.2.1. Auto-FR Global Analytics Key Type

This Key Type describes node level information within the context of the Auto-FR fabric. The System ID of the advertising leaf node MUST be used to differentiate the node among other nodes in the fabric.

The Auto-FR Global Key Type MUST be advertised with the 3rd and 4th bytes of the Key Identifier consisting of all 0s.

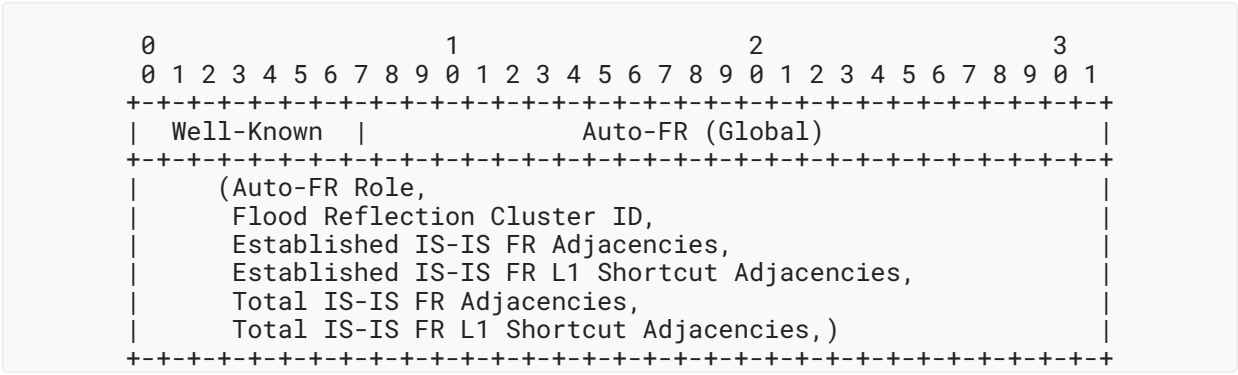

Figure 2: Auto-FR Global Key-Value TIE

where:

Auto-FR Role:
The 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 32-bit integer indicating the Auto-FR Cluster ID of the local node.

Functional IS-IS Flood Reflector Adjacency Count:
A 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 16-bit integer indicating the number of IS-IS Level 1 Shortcut adjacencies in the "Up" state on the local node.

Total IS-IS Flood Reflector Adjacency Count:
A 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 16-bit integer indicating the total number of IS-IS Level 1 Shortcut adjacencies on the local node regardless of state.

6. Acknowledgements

This section will be used to acknowledge major contributors.

7. 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].

8. References

8.1. 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>.

[RIFT] Przygienda, T., Sharma, A., Thubert, P., Rijsman, B., and D. Afanasiev, "RIFT: Routing in Fat Trees", Work in Progress, draft-ietf-rift-rift-14, July 2021.

[RIFT-KV] Head, J. and T. Przygienda, "RIFT Keys Structure and Well-Known Registry in Key Value TIE", Work in Progress, draft-ietf-rift-kv-registry-01, July 2021.

8.2. Informative References

[RIFT-AUTO-EVPN] Head, J. and T. Przygienda, "RIFT Auto-EVPN", Work in Progress, draft-ietf-rift-auto-evpn-01, October 2021.

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 changes to main RIFT common.thrift model.

```
...
enum AutoFRModel {
    /** Full Mesh of L1 tunnel shortcuts, only model supported currently
with auto FR */
    TunnelMode      = 0,
    NoTunnelMode    = 1,
}

const AutoFRModel default_autofr_model = AutoFRModel.TunnelMode

typedef i32          FloodReflectionClusterIDType

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

Figure 3: RIFT Auto-FR: common.thrift

A.2. encoding.thrift

This section specifies changes to main RIFT encoding.thrift model.

```
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 contains the normative Auto-FR Analytics Thrift schema.

```
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 = "Global Auto-FR Telemetry 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 Thrift models required to support Auto-FR. Per the main [RIFT] specification, all signed values MUST be interpreted as unsigned values.

```
/// 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_cidsid2isisnet(cid: FloodReflectionClusterIDType, sid:
UnsignedSystemID) -> Vec<u8> {
    let mut r = vec![0x49];

    r.extend(&cid.to_ne_bytes());
    r.extend(auto_fr_cidsid2isissid(cid, sid).into_iter());
    r.push(0); // magic end

    assert!(r.len() == 10);

    r
}
```

Figure 8: RIFT Auto-FR: auto_fr_cidsid2isisnet

```
/// ISIS system ID derivation
pub fn auto_fr_cidsid2isissid(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],
         sb[6] ^ sb[7],
    ]
}
```

Figure 9: RIFT Auto-FR: auto_fr_cidsid2isissid

```
/// v4 loopback address derivation for every node in auto-fr, returns
address and
/// subnet mask length.
pub fn auto_fr_cidsid2v4loopback(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_cidsid2v4loopback

```
/// V6 loopback derivation for every node in auto fr
pub fn auto_fr_cidsidv6loopback(cid: FloodReflectionClusterIDType,
                                sid: UnsignedSystemID) -> Result<Ipv6Addr,
ServiceErrorType> {
    auto_fr_v6prefixcidsid2loopback(&auto_fr_v6pref(cid), cid, sid)
}
```

Figure 11: RIFT Auto-FR: auto_fr_cidsidv6loopback

```
/// function sorts vector of systemIDs first,
/// followed by a shuffle taking largest/smallest/2nd largest/2nd smallest.
pub(crate) fn auto_fr_sids2frs(mut v: Vec<UnsignedSystemID>)
                               -> Vec<UnsignedSystemID> {
    v.par_sort();
    if v.len() > 2 {
        let mut s = v.split_off(v.len() / 2);
        s.reverse();
        interleave(v.into_iter(), s.into_iter())
            .collect::<Vec<_>>()
    } else {
        v
    }
}
```

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;

    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!("{}00::{}",
                    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