Abstract

This document defines a YANG data model for the configuration and management of RIFT Protocol.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 5, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

The Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction .............................................. 2
2. Design of the Data Model ................................. 2
3. RIFT configuration ....................................... 5
4. RIFT State ................................................. 5
5. RPC ............................................................ 6
6. Notifications ............................................... 6
7. RIFT YANG model .......................................... 6
8. Security Considerations .................................. 20
9. IANA Considerations ...................................... 20
10. Contributors .............................................. 21
11. Normative References .................................... 21
Authors’ Addresses ........................................... 22

1. Introduction

[I-D.przygienda-rift] introduces the protocol definition of RIFT. This document defines a YANG data model that can be used to configure and manage the RIFT protocol.

2. Design of the Data Model

This model imports and augments ietf-routing YANG model defined in [I-D.ietf-netmod-routing-cfg]. Both configuration branch and state branch of [I-D.ietf-netmod-routing-cfg] are augmented. The configuration branch covers node base and policy configuration. The neighbor state will be added in later version. The container "rift" is the top level container in this data model. The presence of this container is expected to enable RIFT protocol functionality.

module: ietf-rift
augment /rt:routing/rt:control-plane-protocols/rt:control-plane-protocol:
  +--rw rift!
    | +--rw node-info
    |       | +--rw systemid  systemid
    |       | +--rw address?  inet:ip-address
    |       | +--rw level-flag?  enumeration
    |       | +--rw level-value?  level-value
    |       | +--rw name?  string
    |       | +--rw pod?  uint16
    |       | +--rw hal?  level-value
    |       | +--rw ol?  boolean {overload}?  
    |       | +--rw ztp?  boolean {ztp-mode}?  
    |       | +--rw bfd?  boolean {bfd}?  

++-rw default-originate? boolean {default-route-originate}?
++-rw flood-reducing? boolean {flood-reducing}?
++-rw interfaces* [local-id]
  |  +--rw local-id uint32
  |  +--rw name? if:interface-ref
  |  +--rw intf-type
  |     +--rw link-type? enumeration
++-rw community
  |  +--rw type? enumeration
  |  +--rw value? uint64
  |  +--rw action? enumeration
++-rw policy-info {policy}?
  |  +--rw pg-type? enumeration
  |  +--rw pg-common
  |     +--rw originator? systemid
  |     +--rw direction
  |        |  +--rw direction-type? enumeration
  |     |  +--rw type
  |        |  +--rw tie-type? enumeration
  |     |  +--rw link-type
  |        +--rw link-type? enumeration
  |  +--rw pg-node
  |     +--rw layer? level-value
  |     +--rw nbr-info
  |        |  +--rw nbr-addr? inet:ip-address
  |        |  +--rw distance? uint32
  |        |  +--rw nbr-layer? level-value
  |        |  +--rw cost? uint32
  |        +--rw link-set* [local-id]
  |             +--rw local-id uint32
  |  +--rw pg-prefix
  |     +--rw prefix? inet:ip-prefix
  |     +--rw cost? uint32
++-ro neighbor
  |  +--ro nbrs* [systemid remote-id]
  |      +--ro systemid systemid
  |      +--ro address? inet:ip-address
  |      +--ro level-flag? enumeration
  |      +--ro level-value? level-value
  |      +--ro name? string
  |      +--ro pod? uint16
  |      +--ro remote-id uint32
  |      +--ro distance? uint32
  |      +--ro local-id? uint32
++-ro database
  |  +--ro ties* [tie-index]
  |      +--ro tie-index uint32
  |      +--ro database-tie

notifications:
  +---n error-set
    +---ro tie-level-error
        |  +---ro originator?  systemid
        |  +---ro direction
        |     |  +---ro direction-type?  enumeration
        |  +---ro type
        |     |  +---ro tie-type?  enumeration
        |  +---ro link-type
        |     |  +---ro link-type?  enumeration
        |  +---ro seq?  uint32
        |  +---ro lifetime?  uint16
        +---ro tie-node
            |  +---ro layer?  level-value
            |  +---ro nbr-info
            |     |  +---ro nbr-addr?  inet:ip-address
            |     |  +---ro distance?  uint32
            |     |  +---ro nbr-layer?  level-value
            |     |  +---ro cost?  uint32
            |     +---ro link-set* [local-id]
            |         +---ro local-id  uint32
            +---ro tie-prefix
                +---ro prefix?  inet:ip-prefix
                +---ro cost?  uint32
            +---ro nbr-error
                +---ro nbrs* [systemid remote-id]
                    +---ro systemid  systemid
                    +---ro address?  inet:ip-address
                    +---ro level-flag?  enumeration
                    +---ro level-value?  level-value
                    +---ro name?  string
                    +---ro pod?  uint16
                    +---ro remote-id  uint32
                    +---ro distance?  uint32
                    +---ro local-id?  uint32

3. RIFT configuration

RIFT configurations require node base information configurations. Some features can be used to enhance protocol, such as BFD, flooding-reducing, policy guide information and community attribute.

4. RIFT State

RIFT states are composed of RIFT node state, neighbor state, database and kv-store. Neighbor’s state will be added in later version. Statistics information may be added in later version.
5. RPC

TBD.

6. Notifications

Unexpected TIE and neighbor’s layer error should be notified.

7. RIFT YANG model

<CODE BEGINS> file "ietf-rift@2018-03-04.yang"
module ietf-rift {

  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-rift";
  prefix rift;

  import ietf-inet-types {
    prefix "inet";
    reference "RFC6991";
  }

  import ietf-routing {
    prefix "rt";
    reference "RFC8022";
  }

  import ietf-interfaces {
    prefix "if";
    reference "RFC7223";
  }

  organization
    "IETF RIFT(Routing In Fat Trees) Working Group";

  contact
    "WG Web:  <http://tools.ietf.org/wg/rift/>"
    "WG List:  <mailto:rift@ietf.org>"
    Editor:  Zheng Zhang
              <mailto:zhang.zheng@zte.com.cn>
    Editor:  Yuehua Wei
              <mailto:wei.yuehua@zte.com.cn>
    Editor:  Shaowen Ma
              <mailto:mashao@juniper.net>
description
"The module defines the YANG definitions for RIFT.

Copyright (c) 2018 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 3618; see
the RFC itself for full legal notices.

revision 2018-03-04 {
    description "Initial revision.";
    reference
        "RFC XXXX: A YANG Data Model for RIFT.
        draft-przygienda-rift-04: RIFT: Routing in Fat Trees."
}

/*
  * Features
  */
feature ztp-mode {
    description "Support Zero Touch Provisioning (ZTP) mode.";
}

feature default-route-originate {
    description "Support the orgination of default route. The default route
generation must be careful to avoid balck-holing.";
}

feature overload {
    description "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.";
}

feature bfd {
    description "Support BFD (RFC5881) function to react quickly to link failures.";
}

feature flood-reducing {
typedef systemid {
    type string {
        pattern '^[0-9A-Fa-f]{4}\.[0-9A-Fa-f]{4}\.[0-9A-Fa-f]{4}\.[0-9A-Fa-f]{4}$';
    }
    description "This type defines RIFT system id using pattern, system id looks like : 0143.0438.0100.AeF0";
}

typedef level-value {
    type uint8 {
        range "0 .. 64";
    }
    description "The value of node level. The max value is 64."
}

/*
 * Identity
 */
identity rift {
    base rt:routing-protocol;
    description "Identity for the RIFT routing protocol."
}

/*
 * Groupings
 */
grouping level-flag {
    leaf level-flag {
        type enumeration {
            enum "auto" {
                description "Node derive level automatically. Only applied if 'configured' is undefined."
            }
            enum "undefined" {
                description "The level has not been determined and has not been configured."
            }
            enum "leaf-only" {
                description "The node will never leave the 'bottom of the
hierarchy’. ‘Superspine’ and ‘configured’ value cannot be defined at the same time as this flag. It implies configured value of 0.”;

enum "configured" {
  description "This means a level value provided manually. When this is defined a node is not participating in ZTP. ‘superspine’ ‘must’ not be set when this value is defined; ‘leaf-only’ can be set only if this value is undefined or set to 0.”;
}

enum "drived" {
  description "This means the level value computed via automatic level derivation when ‘configured’ has not been set to something else than ‘undefined’.”;
}

enum "l2l" {
  description "This means leaf to leaf. ‘superspine’ cannot be defined at the same time as this. It implies ‘leaf-only’ and the according restrictions.”;
}

enum "superspine" {
  description "‘leaf-only’ and ‘configured’ cannot be defined at the same time as this. It implies ‘configured’ value of 64.”;
}


description "This node level information.”;

description "This node level information.”;

}


grouping level-info {
  uses level-flag;

  leaf level-value {
    type level-value;
    description "The node’s level value.”;
  }

description "The ‘level’ of node. It indicates the position of node in the Clos and Fat Tree network.”;

}


grouping node-key {
  leaf systemid {
    type systemid;
    mandatory true;
    description "Each node is identified via a SystemID which is 64 bits wide.”;
  }


leaf address {
    type inet:ip-address;
    description "The loopback address of node. It can be used to indicate the node with system-id.";
}

description "The key information used to distinguish a node."
}

grouping base-node-info {
    uses node-key;
    uses level-info;

    leaf name {
        type string;
        description "The name of this node. It won’t be used as the key of node, just used for description.";
    }

    leaf pod {
        type uint16;
        description "Point of Delivery. The self-contained vertical slice of a Clos or Fat Tree network containing normally only level 0 and level 1 nodes. It communicates with nodes in other PoDs via the spine. We number PoDs to distinguish them and use PoD #0 to denote ‘undefined’ PoD."
    }

    description "The base information of a node."
}

grouping neighbor {
    leaf remote-id {
        type uint32;
        description "The remote-id to reach this neighbor."
    }

    leaf distance {
        type uint32;
        description "The cost value to arrive this neighbor."
    }

    leaf local-id {
        type uint32;
        description "The local-id of link connect to this neighbor."
    }

description "The neighbor information except for systemid."
}

grouping policy {
    description "The policy information. When the tie-type of pgp/kv is set to 'node', node-element is making sense. When the tie-type
of pgp/kv is set to other values except 'node', prefix-info is making sense.

uses pg-info;
container pg-node {
    uses node-element;
    description "The node element information used as policy guide information.";
}
container pg-prefix {
    uses prefix-info;
    description "The prefix information used as policy guide information.";
}
}

grouping node {
    uses base-node-info;
    leaf hal {
        type level-value;
        description "The highest defined level value seen from all valid level offers received.";
    }
    leaf ol {
        if-feature overload;
        type boolean;
        description "If the overload bit in TIEs should be set.";
    }
    leaf ztp {
        if-feature ztp-mode;
        type boolean;
        description "If the node is in ztp mode. If this value is set node should work in ztp mode.";
    }
    leaf bfd {
        if-feature bfd;
        type boolean;
        description "If BFD function is enabled to react link failures after neighbor’s detection.";
    }
    leaf default-originate {
        if-feature default-route-originate;
        type boolean;
        description "If the node can originate southbound default route. This function should be used carefully to avoid black-holing.";
    }
    leaf flood-reducing {
        if-feature flood-reducing;
        type boolean;
description "If the node support flood reducing function defined in section 4.2.3.8.";

description "The information of local node. Includes base information, configurable parameters and features.";

grouping link-type {
  leaf link-type {
    type enumeration {
      enum northbound {
        description "A link to a node one level up.";
      }
      enum southbound {
        description "A link to a node one level down.";
      }
      enum east-west {
        description "A link between two nodes in the same level.";
      }
      enum l2l {
        description "Leaf to leaf. East-west links at leaf level. It is different from East-west links at other levels.";
      }
    }
    description "The type of a link.";
  }
  description "The type of a link.";
}

grouping interface {
  list interfaces {
    key "local-id";
    leaf local-id {
      type uint32;
      mandatory true;
      description "The local id of this interface.";
    }
    leaf name {
      type if:interface-ref;
      description "The interface’s name.";
    }
    container intf-type {
      uses link-type;
      description "The link type of local interface.";
    }
    description "The interface information on this node.";
  }
  description "The interface information.";
}
grouping community-action {
  description "The actions apply on a matching of community.";
  leaf action {
    type enumeration {
      enum prune-nh-required {
        description "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.";
      }
      enum prune-nh-avoid {
        description "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.";
      }
      enum drop-if {
        description "If node X has community M, discard P.";
      }
      enum drop-if-not {
        description "If node X does not have the community M, discard P.";
      }
      enum add-if-node-match {
        description "If the node X has community S, then add community T to P.community_list.";
      }
    }
  }
  description "The actions apply on a matching of community.";
}

grouping community {
  leaf type {
    type enumeration {
      enum top {
        description "The direction type of community is toward to top.";
      }
      enum bottom {
        description "The direction type of community is toward to bottom.";
      }
    }
  }
  description "community direction type.";
  leaf value {
    type uint64;
    description "community value.";
  }
  uses community-action;
  description "Community Attribute.";
}
grouping direction-type {
    leaf direction-type {
        type enumeration {
            enum northbound {
                description "It means moving towards the top of the Clos or Fat Tree network.";
            }
            enum southbound {
                description "It means moving towards the bottom of the Clos or Fat Tree network.";
            }
        }
    }
    description "The direction type.";
}

grouping node-element {
    leaf layer {
        type level-value;
        description "The layer of this element.";
    }
    container nbr-info {
        leaf nbr-addr {
            type inet:ip-address;
            description "The information used to distinguish a neighbor.";
        }
        leaf distance {
            type uint32;
            description "The cost value to arrive this neighbor.";
        }
        leaf nbr-layer {
            type level-value;
            description "The layer of this neighbor.";
        }
        leaf cost {
            type uint32;
            description "The cost of this neighbor.";
        }
        description "The information of this neighbor.";
    }
    list link-set {
        key "local-id";
        leaf local-id {
            type uint32;
            description "The local id of link.";
        }
    }
}
grouping prefix-info {
    leaf prefix {
        type inet:ip-prefix;
        description "The prefix information.";
    }
    leaf cost {
        type uint32;
        description "The cost of this prefix.";
    }
    description "The detail information of prefix.";
}

grouping tie-type {
    leaf tie-type {
        type enumeration {
            enum node {
                description "The node topology information elements.";
            }
            enum prefix {
                description "The prefix topology information elements.";
            }
            enum pgp-prefix {
                description "The PGP prefix information elements.";
            }
            enum kv {
                description "The Key Value information elements.";
            }
        }
        description "The possible TIE’s types.";
    }
    description "The possible TIE’s types.";
}

grouping tie-id {
    leaf originator {
        type systemid;
        description "The originator’s systemid of this TIE.";
    }
    container direction {
        uses direction-type;
        description "The direction type of this TIE.";
    }
    container type {
uses tie-type;
  description "The type of this TIE.");
}
container link-type {
  uses link-type;
  description "The type of link in this TIE.");
}
description "TIE is the acronym for 'Topology Information Element'.
  TIEs are exchanged between RIFT nodes to describe parts
  of a network such as links and address prefixes. This is
  the TIE identification information.";
}

grouping tie-info {
  leaf seq {
    type uint32;
    description "The sequence number of a TIE.";
  }
  leaf lifetime {
    type uint16 {
      range "1 .. 65535";
    }
    description "The lifetime of a TIE.";
  }
  container tie-node {
    uses node-element;
    description "The node element information in this TIE.";
  }
  container tie-prefix {
    uses prefix-info;
    description "The prefix information in this TIE.";
  }
}
description "TIE is the acronym for 'Topology Information Element'.
  TIEs are exchanged between RIFT nodes to describe parts
  of a network such as links and address prefixes. This TIE
  info is used to indicate the state of this TIE. When the
  type of this TIE is set to 'node', the node-element is
  making sense. When the type of this TIE is set to other
  types except for 'node', the prefix-info is making sense.";
}

grouping pg-info {
  description "Information that is passed in either southbound direction
  or north-bound direction by the means of diffusion and can
  be filtered via policies. Policy-Guidedn Prefixes and KV
  Ties are examples of Policy-Guided Information.";
  leaf pg-type {
    type enumeration {
enum pgp {
    description "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.";
}

enum kv-tie {
    description "A S-TIE that is carrying a set of key value pairs. It can be used to distribute information in the southbound direction within the protocol.";
}

description "The types of Policy-Guided Information.";
}

container pg-common {
    uses tie-id;
    description "The brief key information of this policy. It applies to node-element and prefix as well.";
}

grouping algorithm {
    choice algorithm-type {
        case spf {
            description "The algorithm is SPF.";
        }
        description "The possible algorithm types.";
    }
    description "The computation algorithm types.";
}

grouping nh-set {
    list next-hop-set {
        key "local-id";
        leaf local-id {
            type uint32;
            description "The local id of interface.";
            description "The next-hop set.";
        }
        description "The next-hop set.";
    }
    description "The next-hop set.";
}

/* Data nodes
 augment "/rt:routing/rt:control-plane-protocols/rt:control-plane-protocol" {
when "derived-from(rt:type, 'rift:rift')" {
  description "This augment is only valid for a routing protocol instance of RIFT.";
}

description "RIFT (Routing in Fat Trees) YANG model.

container rift {
  presence "Container for RIFT protocol.");
  description "RIFT configuration data.");

  container node-info {
    description "The node information about RIFT.";
    uses node;
    uses interface;
    container community {
      uses community;
      description "The community attribute and information.";
    }
    container policy-info {
      if-feature policy;
      uses policy;
      description "The policy information.";
    }
  }

  container neighbor {
    config false;
    list nbrs {
      key "systemid remote-id";
      uses base-node-info;
      uses neighbor;
      /*uses nbr-status;*/ /*The status will be added in later version.*/
      description "The information of a neighbor.";
    }
    description "The neighbor’s information.";
  }

  container database {
    config false;
    list ties {
      key "tie-index";
      leaf tie-index {
        type uint32;
        description "The index of a TIE.";
      }
      container database-tie {
        uses tie-id;
        uses tie-info;
        description "The TIEs in the database.";
      }
      description "The detail information of a TIE.";
    }
  }
}
uses algorithm;
description "The TIEs information in database.";
} // database

container kv-store {
  config false;
  list kvs {
    key "kvs-index";
    leaf kvs-index {
      type uint32;
      description "The index of a kv pair.";
    }
  }
  container kvs-tie {
    uses tie-id;
    uses tie-info;
    description "The TIEs in the kv-store.";
  }
  description "The information used to distinguish a Key/Value pair. When the type of kv is set to 'node', node-element is making sense. When the type of kv is set to other values except 'node', prefix-info is making sense.";
}
description "The Key/Value store information.";
} // kv-store
} // rift
} // augment

/*
 * RPCs
 */

/*
 * Notifications
 */
notification error-set {
  description "The errors notification of RIFT.";
  container tie-level-error {
    uses tie-id;
    uses tie-info;
    description "The level is undefined in the LIEs.";
  }
  container nbr-error {
    list nbrs {
      key "systemid remote-id";
    }
  }
} // error-set

uses base-node-info;
uses neighbor;

description "The information of a neighbor."
{

description "The neighbor errors set."
}


code ends

8. Security Considerations

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

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

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations.

The RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations.

9. IANA Considerations

The IANA is requested to assign two new URIs from the IETF XML registry ([RFC3688]). Authors are suggesting the following URI:


Registrant Contact: RIFT WG

XML: N/A, the requested URI is an XML namespace
This document also requests one new YANG module name in the YANG Module Names registry ([RFC6020]) with the following suggestion:

name: ietf-rift
prefix: rift
reference: RFC XXXX

10. Contributors

The authors would like to thank Benchong Xu (xu.benchong@zte.com.cn), for his valuable contributions.

11. Normative References

[I-D.ietf-netmod-rfc6087bis]
Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", draft-ietf-netmod-rfc6087bis-18 (work in progress), February 2018.

[I-D.ietf-netmod-routing-cfg]

[I-D.przygienda-rift]


Authors' Addresses
Zheng Zhang
ZTE Corporation

Email: zhang.zheng@zte.com.cn

Yuehua Wei
ZTE Corporation

Email: wei.yuehua@zte.com.cn

Shaowen Ma
Juniper

Email: mashao@juniper.net

Xufeng Liu
Jabil

Email: Xufeng_Liu@jabil.com