Abstract

Currently new services create new opportunities for both network providers and service providers. Shared Unified Policy Automation (SUPA) can provide application-based policies and means to model and program the abstract view of network infrastructure and service function interdependencies in order to support and feed network management and controlling. Such network management and controlling services that provide the required configuration and application programming interfaces may need a set of specified YANG models to achieve the aforementioned goal. This document defines a YANG data model for SUPA configuration.

Status of this Memo

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

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

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/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html
1. Introduction .................................................. 2
2. Conventions used in this document........................ 3
3. Network Configuration Model Overview ..................... 3
4. Network Configuration Modules .............................. 4
   4.1. L3VPN Configuration YANG Module.................... 4
   4.1.1. L3VPN Configuration YANG Model............... 5
   4.2. Service Flow Configuration .......................... 9
   4.2.1. Service Flow Configuration Yang Module....... 11
   4.3. IP TE Configuration YANG Module ................. 15
   4.3.1. IP TE Data Model Structure .................. 16
   4.3.2. IP TE YANG Module ........................ 18
   4.4. Unified Tunnel Configuration YANG Module ...... 23
   4.4.1. Unified Tunnel Model Structure .......... 24
   4.4.2. Service Configuration YANG Module .......... 25
5. Security Considerations ................................... 30
6. IANA Considerations ........................................ 30
7. Acknowledgments ........................................... 31
8. References .................................................. 31
   8.1. Normative References ................................ 31
   8.2. Informative References ............................ 31

1. Introduction

Currently new services bring new challenges and opportunities for both network providers and service providers. Meanwhile, legacy services such as L3VPN [RFC4110], Service Flow and IP TE (Traffic Engineering)[RFC3272] also need specialized management and
controlling capability from the network management systems to improve the experiences for fast deployment and dynamic configuration.

This document introduces Shared Unified Policy Automation (SUPA) [APONF-architecture] which provides application-based policies and means to model and program the abstract view of network infrastructure and service function interdependencies in order to support and feed network management and control by enabling the streaming transfer of bulk-variable/data of the up-to-date Service Function Path (SFP) based network configuration and network topology models, and mapping the SFP based network configuration and network topology models into specific device-level configuration models.

This document introduces YANG [RFC6020] [RFC6021] data models for SUPA configuration. Such a set of models can facilitate the standardization for the interface of SUPA, as they are compatible to a variety of protocols such as NETCONF [RFC6241] and [RESTCONF]. Please note that in the context of SUPA, the term "application" refers to a management application employed, and possibly implemented, by an operator.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying [RFC2119] significance.

3. Network Configuration Model Overview

Figure 1 illustrates the network configuration model which contains several modules for specific services such as L3VPN, Service Flow, IP TE (Traffic Engineering) and Unified Tunnel.
4. Network Configuration Modules

In this section, several specific network configuration models are described based on a set of specific network services and the architecture of SUPA [SUPA-architecture].

4.1. L3VPN Configuration YANG Module

A Layer 3 Virtual Private Network (L3VPN) interconnects sets of hosts and routers based on Layer 3 addresses and forwarding. L3VPN can be based on MPLS or IP technologies. L3VPN is a PE-based VPN managed by operators. L3VPN is widely used in carrier metro networks to provide VPN service for enterprise users.

A L3VPN model is a collection of L3VPN instances. A L3VPN instance contains a set of access interfaces to network devices as well as other attributes, such as routing protocol, address family, topology, and so on.

To configure a L3VPN instance, the administrator needs to specify which port(s) of a network device belongs to a L3VPN instance. Those ports and network device information can be derived from a network topology model in a network management system. The administrator also needs to specify what routing protocol needs to be configured for a L3VPN instance.

The following describes the information model for L3VPN, based on which programmers can develop applications to configure L3VPN instances.
module: SUPA-netl3vpn

+-rw netl3vpnInstance* [instanceName]
  +-rw instanceName        string
  +-rw servicType?         enumeration
  +-rw afType?             enumeration
  +-rw acIfs
    +-rw acIf* [vncAcIfId]
      +-rw acIfId         string
      +-rw acIfAddr?      inet:ipv4-address
      +-rw acIfMask?      unsignedByte
      +-rw role?          enumeration
      +-rw phyNodeId?     string
      +-rw physAcIfId?    string
      +-rw protocol*
        +-rw protocolType enumeration
        +-rw igpAttr*
          +-rw protocolId  uint32
        +-rw bgpAttr*
          +-rw remoteAsNumber string
          +-rw remotePeerAddr string

4.1.1. L3VPN Configuration YANG Model

<CODE BEGINS>
module SUPA-netl3vpn {
  namespace "http://www.huawei.com/netconf/vrp";
  prefix "nc";
  organization "Huawei Technologies Ltd";
  description "";
  revision "2014-08-13";

  list netl3vpnInstance {
    key "instanceName";
    max-elements "unbounded";
    min-elements "0";
    description ".";

    leaf instanceName {
      description ".";
      config "true";
      type string {
        length "1..64";
        pattern "^[^?]*";
      }
    }

    leaf servicType {
      description ".";
      config "true";
    }

    ...
  }

...
default "full-mesh";
type enumeration {
  enum full-mesh {
    value "0";
    description "full-mesh";
  }
  enum hub-spoke {
    value "1";
    description "hub-spoke";
  }
}

leaf afType {
  description ".";
  config "true";
  default "ipv4uni";
type enumeration {
  enum ipv4uni {
    value "0";
    description "ipv4uni";
  }
  enum ipv6uni {
    value "1";
    description "ipv6uni";
  }
}
}

list acIf {
  key "acIfId";
  max-elements "unbounded";
  min-elements "0";
  description ".";
  leaf acIfId {
    description ".";
    config "true";
type string {
      length "1..64";
      pattern "([^?]*)";
    }
  }
  leaf acIfAddr {
    description ".";
    config "true";
  }
}
type string {
    pattern "([\^?]*);"
}

leaf acIfMask {
    description ".";
    config "true";
    type uint8 {
        range "0..128";
    }
}

leaf role {
    description ".";
    config "true";
    type enumeration {
        enum edge-if {
            value "0";
            description "edge-if:";
        }
        enum center-if {
            value "1";
            description "center:";
        }
    }
}

leaf phyNodeId {
description ".";
config "true";
type string {
    length "1..64";
    pattern "([^?]*)";
}

leaf phyAcIfId {
    description ".";
    config "true";
type string {
    length "1..64";
    pattern "([^?]*)";
}
}

container protocol {
    description ".";

    leaf protocolType {
        description ".";
        config "true";
default "ospf";
type enumeration {
    enum bgp {
        value "0";
        description "bgp";
    }
    enum ospf {
        value "1";
        description "ospf";
    }
    enum isis {
        value "2";
        description "isis";
    }
    }
}

container igpAttr {
    description ".";

    leaf protocolId {
        description ".";
        config "true";
default "0";
    }
}
4.2. Service Flow Configuration

Service Flow represents a flow and policy rule definition which enables users to granularly control the traffic so that dynamic and software-defined traffic management is possible. This section provides an overview of the YANG-based configuration specific model of the service flow application. There are two basic elements of the service flow model:

- **Flow** is the data traffic, which can be identified by certain field values such as source IP address, destination IP address, and etc, between computers or devices or between nodes in a network.

- **Flow Policy** is the control of flow which determines the in_port/gress of the flow.
The structure of the SUPA service flow data model, as later defined in the YANG module "SUPA-service flow", is depicted in the following diagram. Brackets enclose list keys, "rw" means configuration data, and '?' designates optional nodes. The figure does not depict all definitions; it is solely intended to illustrate the overall structure.

module: SUPA-serviceflow
  +-rw flows
    +-rw flow* [flowName]
      |  +-rw flowName        string
      |  |  +-rw flowFilter* [flowFilterID]
      |  |    |  +-rw flowFilterID  string
      |  |    |  |  +-rw sourceIP?     inet:ipv4-address
      |  |    |  |  +-rw destinationIP? inet:ipv4-address
      |  |    |  |  +-rw sourcePrefix?  inet:ipv4-address
      |  |    |  |  +-rw destinationPrefix? inet:ipv4-address
      |  |    |  |  +-rw prefix?        inet:ipv4-address
      |  |    |  |  +-rw sourcePort?    inet:port-number
      |  |    |  |  +-rw destinationPort? inet:port-number
      |  |    |  |  +-rw inIf?          string
      |  |    |  |  +-rw outIf?         string
      |  |    |  |  +-rw protocolId?   string
    |  +-rw flowPolicies
      |    +-rw flowPolicy* [policyName]
      |      |  |  +-rw policyName  string
      |      |  |  +-rw flowName?   string
      |      |  |  +-rw nodeKeyType? enumeration
      |      |  |  +-rw nodeId?     string
      |      |  |  +-rw tpType?     enumeration
      |      |  |  +-rw tpId?       string
4.2.1. Service Flow Configuration Yang Module

<CODE BEGINS>
module SUPA-serviceflow {
    namespace "urn:TBD:params:xml:ns:yang:serviceflow";
    // replace with IANA namespace when assigned
    prefix "nc";
    import ietf-inet-types { prefix inet;}
    organization "TBD";
    contact "WILL-BE-DEFINED-LATER";
    description "This module defines a model for service flow";
    revision "2014-08-13";
}

container flows {
    list flow {
        key flowName;
        max-elements "unbounded";
        min-elements "0";
        description "Flow";
        leaf flowName {
            description "Flow Name";
            config "true";
            type string {
                length "0..31";
            }
        }
        list flowFilter {
            key flowFilterID;
            max-elements "unbounded";
            min-elements "0";
            description "Flow Filter";
            leaf flowFilterID {
                description "Flow Filter";
                config "true";
                type string {
                    length "0..64";
                }
            }
        }  
        leaf sourceIP {
            description "source IP";
            config "true";
            default "0.0.0.0";
        }
    }
}</CODE>
type inet:ipv4-address;
)
leaf destinationIP {
    description "destination IP";
    config "true";
    default "0.0.0.0";
    type inet:ipv4-address;
}
leaf sourcePrefix {
    description "source Prefix";
    config "true";
    default "0.0.0.0";
    type inet:ipv4-address;
}
leaf destinationPrefix {
    description "destination Prefix";
    config "true";
    default "0.0.0.0";
    type inet:ipv4-address;
}
leaf prefix {
    description "Prefix";
    config "true";
    default "0.0.0.0";
    type inet:ipv4-address;
}
leaf sourcePort {
    description "Source Port";
    config "true";
    type inet:port-number{
        range "0..65535";
    }
}
leaf destinationPort {
    description "Destination Port";
    config "true";
    type inet:port-number{
        range "0..65535";
    }
}
leaf inIf {
    description "In Intreface Name";
    config "true";
    type string {
        length "0..64";
    }
}
leaf outIf {
    description "Out Interface Name";
    config "true";
    type string {
        length "0..64";
    }
}

leaf protocolId {
    description "Protocol ID";
    config "true";
    type string {
        length "0..64";
    }
}

container flowPolicies {
    list flowPolicy {
        key "policyName";
        max-elements "unbounded";
        min-elements "0";
        description "Flow Policy";

        leaf policyName {
            description "Policy Name";
            config "true";
            type string {
                length "0..64";
            }
        }

        leaf flowName {
            description "Flow Name";
            config "true";
            type string {
                length "0..64";
            }
        }

        leaf nodeKeyType {
            description "Node Key Type";
            config "true";
            default "lsr-id";
            type enumeration {

enum lsr-id {
  value "0";
  description "lsr-id:"
}

enum invalid {
  value "1";
  description "invalid:";
}

enum system-id {
  value "2";
  description "system-id:"
}

enum router-id {
  value "3";
  description "router-id:"
}

enum fp-id {
  value "4";
  description "fp-id:"
}

enum mac {
  value "5";
  description "mac:"
}

leaf nodeId {
  description "Node Id";
  config "true";
  default "_leftNode_";
  type string {
    length "0..64";
  }
}

leaf tpType {
  description "Terminal Point Key";
  config "true";
  default "ip";
  type enumeration {
    enum ip {
      value "0";
      description "ip:"
    }
    enum invalid {
      value "1";
      description "invalid:";
    }
  }
}
enum interface {
  value "2";
  description "interface:";
}

leaf tpId {
  description "Terminal Point Id";
  config "true";
  default ".Tp_.";
  type string {
    length "0..64";
  }
}

4.3. IP TE Configuration YANG Module

The network connection between data centers is usually leased and its bandwidth is very expensive. The traditional shortest path algorithm is based on static cost, in which the path calculation cannot be dynamically adjusted based on real-time bandwidth usage. This can often cause bandwidth waste in practice. An IP path application can add constraints on the paths to solve this issue.

Figure 2 illustrates a simple example topology. There are two paths from DC A to DC B, for example, A-->B (path 1) and A-->C-->B (path 2). When the bandwidth between A and B is not sufficient, A will automatically transmit the traffic via C. The network management applications will configure a threshold T (e.g., 80%) as a constraint for the path and apply it to the IP path. When an application request is received, A will detect the bandwidth use of both paths. When the bandwidth use ratio of path 1 (T1) has exceeded value T (e.g., 90%), while the bandwidth use ratio of path 2 (T2) is much less than T (e.g., 10%), it will transmit the traffic to B via C, even though P1 is the shortest path between A and B. Here the constraint about the path routing has to be A-->C-->B.

In this case, the bandwidth use efficiency between A and B will be improved, and risks of congestion between the datacenters will be alleviated.
4.3.1. IP TE Data Model Structure

There are multiple use cases for such a configuration specific data model, which is service-oriented and device-independent. A network controller can then use the instantiated data to map the specific service to the network elements that it controls. Alternatively, nodes within the network could also abstract their state of the network and share this state either among themselves or with the controller.

This section provides an overview of the YANG based configuration specific model of the IP TE application. The main elements of the IP TE model are as follows:

- An "ipte" is a set of traffic engineered IP paths; it consists of multiple ipteFlows and iptePathResults.

- An ipteFlow is an IP flow with path constraints, including both bandwidth and resource requirements. ipteFlows can be distinguished via ipteFlowName which unique within the management domain.
An iptePathResult is a computed path of a requested ipteFlow. An iptePathResult consists of a set of nodes that belong to the computed path. An iptePathResult can be distinguished via ipteFlowName and pathName.

The structure of the ipte data model, as defined in the YANG module "SUPA-ipte", is described as follows. Brackets denote list keys, "rw" denotes configuration data, "ro" denotes operational state data, "*" denotes the parameter that can have multiple instances, and "?" denotes optional parameters. The figure is, again, solely intended to provide view of the overall structure of the ipte data model.
4.3.2. IP TE YANG Module

<CODE BEGINS>
module huawei-ipte {

    prefix "nc";

description "V8R7 schema";
revision "2014-08-13";

    container ipteFlows {

        list ipteFlow {

            key ipteFlowName;
            max-elements unbounded;
            min-elements 0;
            description "IP flow intends to be adjusted.";

            leaf ipteFlowName {

description "String name of the IP flow";
config true;
type string {
    length "0..64";
    pattern "([^"]*)";
}
}
container pathPrefixs {
    list pathPrefix {
        key prefix;
        max-elements unbounded;
        min-elements 0;
        description "IP address prefix to specify the
destination IP address of the flow.";

        leaf prefix {
            description "prefix";
            config true;
type string {
                length "0..64";
                pattern "([^"]*)";
            }
        }
        leaf maskLength {
            description "mask length";
            config true;
type uint32 {
                range "0..128";
            }
        }
    }
}

leaf bandwidth {
    description "Minimum available bandwidth required in
kbps";
    config true;
type uint32 {
        range "0..128";
    }
}
}
container paths {
    description "Constrained path of the flow";
config true;
list path {
    key pathName;
    max-elements unbounded;
    min-elements 0;
    description "constraint path";
}

leaf pathName {
    description "String name of the constrained path";
    config true;
    type string {
        length "0..64";
        pattern "([^?]*")";
    }
}

leaf pathType {
    description "Constrained type of the path";
    config true;
    default "auto";
    type enumeration {
        enum strict {
            value 0;
            description "strict";
        }
        enum auto {
            value 1;
            description "auto";
        }
    }
}

container pathNodes {
    list pathNode {
        key nodeId;
        max-elements unbounded;
        min-elements 0;
        description ".";

        leaf nodeId {
            description "constraint path node";
            config true;
            type string {
                length "0..64";
            }
        }
    }
}
pattern "([?]+)";
}
}
leaf nodeRole {

description "The role of the node";
config true;

type enumeration {
    enum ingress {
        value 0;
        description "ingress node";
    }
    enum egress {
        value 1;
        description "egress node";
    }
    enum transit {
        value 2;
        description "transit node";
    }
}
}

leaf sequence {
    description "constraint path node sequence";
    config true;
    default 1;
    type uint32 {
        range "0..128";
    }
}

}`

container ipтеPathResults {

list iptePathResult {
    config false;
    key pathName;
    max-elements unbounded;
    min-elements 0;

    description "Traffic engineered IP path as a result of IP
    flow adjustment.";

    leaf iptePrefixName {
        description "prefix name";
        config false;
        type string {
            length "0..64";
            pattern "([?]*")";
        }
    }

    leaf pathName {
        description "constraint path name";
        config false;
        type string {
            length "0..64";
            pattern "([?]*")";
        }
    }

    container iptePathResultNodes {
        list iptePathResultNode {
            max-elements unbounded;
            min-elements 0;
            description ".";
            key nodeId;
            leaf nodeId {
                description "constraint path node ID";
                config false;
                type string {
                    length "0..64";
                    pattern "([?]*")";
                }
            }
            leaf nodeRole {
                description "The role of the node";
                config false;
            }
        }
    }
}
type enumeration {
  enum ingress {
    value 0;
    description "ingress node";
  }
  enum egress {
    value 1;
    description "egress node";
  }
  enum transit {
    value 2;
    description "transit node";
  }
}

leaf sequence {
  description "constraint path node sequence";
  config false;
  default 1;
  type uint32 {
    range "0..128";
  }
}

4.4. Unified Tunnel Configuration YANG Module

Unified tunnel (also abbreviated as utunnel) denotes a kind of generic tunnel which is used to carry services from a source node to a destination node while users do not need to care about the details. The process of using such a utunnel when carrying a service can be summarized as follows: a) create a utunnel, b) set the utunnel as the outgoing port of a service flow, c) if the service matches the filter of the service flow, the service will be directed into the utunnel.
With utunnel, operators are able to easily implement a group of tunnels in the following scenarios:

- between two network entities;
- from one network entity to a set of network entities;
- to and from an end-to-end connection via group tunnels between the network entities in the path between two points.

### 4.4.1. Unified Tunnel Model Structure

The universal elements of the unified tunnel model are as follows:

- utunnel, which abstracts the common properties of the various tunnel technologies, such as TE tunnel, GRE tunnel, etc. is proposed to simplify use.
- Each utunnel has a unique tunnelName, which distinguishes it from other utunnels in the list.
- A sourceNodeId and destinationNodeId need to be specified when creating a utunnel. The direction of a utunnel should also be considered, this is to decide whether it needs to be chosen from unidirectional or bidirectional. However, the users of a utunnel may not need to specify tunnelType, if the default tunnelType is acceptable.

The structure of the SUPA unified tunnel data model, as later defined in the YANG module "SUPA-utunnel", is depicted in the following diagram. Brackets enclose list keys, "rw" means configuration data, and "?" designates optional nodes. The figure does not depict all definitions; it is intended to illustrate the overall structure.
module: SUPA-utunnel
 +++rw tunnels
     +--rw tunnel* [tunnelName]
         +--rw tunnelName                string
         +--ro tunnelID?                 string
         +--rw direction?                enumeration
         +--rw tunnelType?               enumeration
         +--rw sourceNodeKeyType?        enumeration
         +--rw sourceNodeId?             string
         +--rw sourceTpType?             enumeration
         +--rw sourceTpId?               string
         +--rw destinationNodeKeyType?   enumeration
         +--rw destinationNodeId?        string
         +--rw destinationTpType?        enumeration
         +--rw destinationTpId?          string
         +--rw adminStatus?              enumeration
         +--ro operStatus?               enumeration

4.4.2. Service Configuration YANG Module

<CODE BEGINS>
module SUPA-utunnel {
    namespace "TBD";
    prefix "nc";
    organization "TBD";
    contact "TBD";
    description "TBD";
    revision "2014-08-13";

    container tunnels {

        list tunnel {

            key "tunnelName";
            max-elements "unbounded";
            min-elements "0";
            description "tunnel";

            leaf tunnelName {
                description "Tunnel Name";
                config "true";
                type string {
                    length "1..31";
                }
            }

            leaf tunnelID {

description "tunnel ID";
config "false";
type string {
   length "1..31";
}
}
leaf direction {
   description "tunnel direction";
   config "true";
type enumeration {
   enum single {
      value "0";
      description "single direction:";
   }
   enum double {
      value "1";
      description "double direction:";
   }
}
}
leaf tunnelType {
   description "tunnel type";
   config "true";
type enumeration {
   enum ldp {
      value "0";
      description "ldp:";
   }
   enum bgp {
      value "1";
      description "bgp:";
   }
   enum te {
      value "2";
      description "te:";
   }
   enum static-lsp {
      value "3";
      description "static-lsp:";
   }
   enum gre {
      value "4";
      description "gre:";
   }
}
}
leaf sourceNodeKeyType {
description "Source Node Key Type";
config "true";
default "lsr-id";
type enumeration {
    enum name {
        value "0";
        description "name:";
    }
    enum invalid {
        value "1";
        description "invalid:";
    }
    enum system-id {
        value "2";
        description "system-id:";
    }
    enum router-id {
        value "3";
        description "router-id:";
    }
    enum lsr-id {
        value "4";
        description "lsr-id:";
    }
    enum fp-id {
        value "5";
        description "fp-id:";
    }
    enum mac {
        value "6";
        description "mac:";
    }
}
leaf sourceNodeId {
    description "Source Node Id";
    config "true";
    default "_sourceNode_";
type string {
    length "1..31";
}
leaf sourceTpType {
    description "Source Terminal Point Key";
    config "true";
    default "ip";
type enumeration {

enum ip {
    value "0";
    description "ip:"
}
enum invalid {
    value "1";
    description "invalid:"
}
enum interface {
    value "2";
    description "interface:"
}

leaf sourceTpId {
    description "Source Terminal Point Id";
    config "true";
    default "_sourceTp_";
    type string {
        length "1..31";
    }
}

leaf destinationNodeKeyType {
    description "Destination Node Key Type";
    config "true";
    default "lsr-id";
    type enumeration {
        enum name {
            value "0";
            description "name:"
        }
        enum invalid {
            value "1";
            description "invalid:"
        }
        enum system-id {
            value "2";
            description "system-id:"
        }
        enum router-id {
            value "3";
            description "router-id:"
        }
        enum lsr-id {
            value "4";
            description "lsr-id:"
        }
    }
}
enum fp-id {
  value "5";
  description "fp-id:";
}
enum mac {
  value "6";
  description "mac:";
}
leaf destinationNodeId {
  description "Destination Node Id";
  config "true";
  default "_destinationNode_";
  type string {
    length "1..31";
  }
}
leaf destinationTpType {
  description "Destination Terminal Point Key Type";
  config "true";
  default "ip";
  type enumeration {
    enum ip {
      value "0";
      description "ip:";
    }
    enum invalid {
      value "1";
      description "invalid:";
    }
    enum interface {
      value "2";
      description "interface:";
    }
  }
}
leaf destinationTpId {
  description "Destination Terminal Point Id";
  config "true";
  default "_destinationTp_";
  type string {
    length "1..31";
  }
}
leaf adminStatus {
  description "AdminState";
config "true";
default "up";
type enumeration {
  enum down {
    value "0";
    description "down:";
  }
  enum up {
    value "1";
    description "up:";
  }
}
leaf operStatus {
  description "operStatus";
  config "false";
type enumeration {
  enum down {
    value "0";
    description "down:";
  }
  enum up {
    value "1";
    description "up:";
  }
}
}

5. Security Considerations

TBD

6. IANA Considerations

This document has no actions for IANA.
7. Acknowledgments

This document has benefited from reviews, suggestions, comments and proposed text provided by the following members, listed in alphabetical order: Jing Huang, Junru Lin, Yiyong Zha, and Cathy Zhou.

8. References

8.1. Normative References


8.2. Informative References


Authors’ Addresses

Adel Zaalouk
EICT GmbH
Torgauer Strasse 12-15
Berlin 10829
Germany
Email: adel.ietf@gmail.com

Kostas Pentikousis
EICT GmbH
Torgauer Strasse 12-15
Berlin 10829
Germany
Email: k.pentikousis@eict.de

Will(Shucheng) Liu
Huawei Technologies
Bantian, Longgang District
Shenzhen 518129
P.R. China
Email: liushucheng@huawei.com