Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2016

A. Wang
China Telecom
Z. Wang
Y. Zhuang
Huawei

July 3, 2015

YANG Data Model for Tunnel Management
draft-wwz-netmod-yang-tunnel-cfg-00

Abstract

This document defines a YANG data model for the configuration and management of generic tunnels. The data model includes configuration data and state data. And it can serve as a base model which is augmented with technology-specific details in other, more specific tunnel models.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 4, 2016.

Table of Contents

1. Introduction

Tunneling mechanisms provide isolated communication from one VPN edge device to another in the VPN solutions. Available tunneling mechanisms include (but are not limited to): GRE [RFC2784] [RFC2890], IP-in-IP encapsulation [RFC2003] [RFC2473], IPsec [RFC2401] [RFC2402], and MPLS [RFC3031] [RFC3035], VXLAN ([RFC7348]), VXLAN-GPE ([VXLAN-GPE]), NVGRE ([NVGRE]), GTP [GTP-U], and MPLS-in-GRE ([RFC2784], [RFC2890], [RFC4023]).

[RFC4110] discusses some common characteristics shared by all forms of tunneling, and some common problems to which tunnels provide a solution from the following perspectives:

- Multiplexing
- QoS/SLA
- Tunnel setup and maintenance
- Security

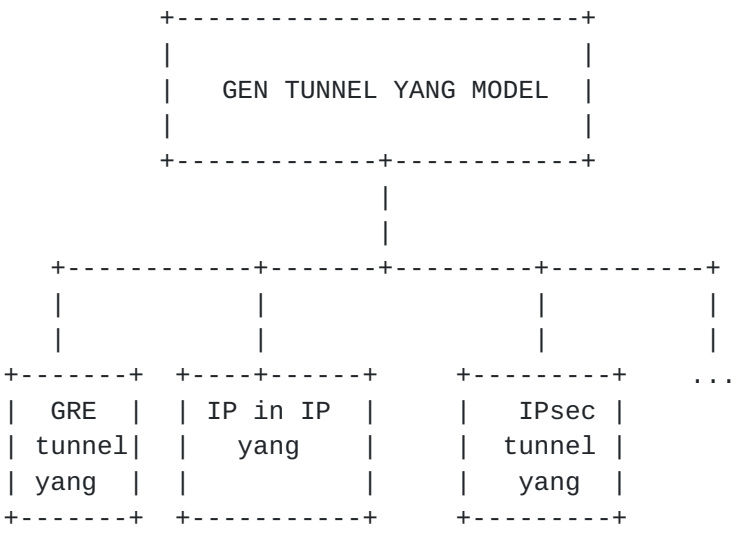
This document defines a yang data model[RFC6020] to represent common building block for tunnels configuration data and state data. This model can be augmented with technology specifics of particular types of tunnel.

2. Conventions and Terminology

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

3. Architecture of Generic YANG Model for tunnel

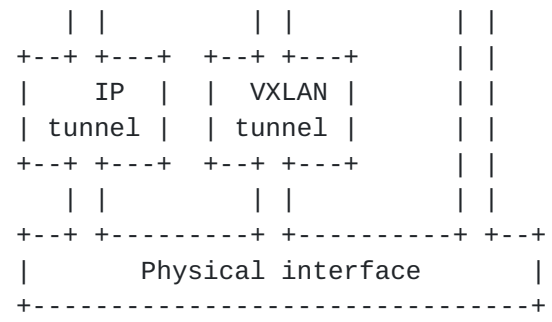
In this document we define a common core tunnel model. The YANG model defined here is generic such that other technologies can extend it for technology specific needs. The Generic Tunnel YANG model acts as the root for other Tunnel YANG models. This allows users to span across Tunnel of different technologies through a uniform API set. Figure 1 depicts the relationship of different Tunnel YANG models to the Generic Tunnel YANG Model. Some technologies may have different sub-technologies. As an example, consider Network IP-in-IP Tunnel. These could employ IPv4-in-IPv4, IPv6-in-IPv4, and other methods as encapsulation protocol. Figure 1 depicts relationship of different YANG modules.



Relationship of technology specific TUNNEL YANG model to generic (base) YANG model

3.1. Relationship to interface yang data model

This section clarifies the relationship of this yang module to the interfaces yang [RFC7223]. Tunnels are handled by creating a logical interface for each tunnel. Each logical interface (physical or virtual) need to map to interface yang model [RFC7223]. To do so, in this draft, we augment interface yang model with tunnel interface specifics, which is binding tunnel interface with a physical interface [RFC7223].



4. Design of GENERIC TUNNEL Modules

This document defines the YANG module "gen-tunnel", which augments the "interface" and "interface-state" lists defined in the "ietf-interfaces" module [RFC7223] with tunnel specific data nodes, and also adds tunnel specific state data.

4.1. tunnel configuration model

The data model has the following structure for tunnel configuration per interface:

```
module: gen-tunnel
      augment /if:interfaces/if:interface:
      +--rw tunnel-configuration
         +--rw base-interface?         if:interface-ref
         +--rw tunnel-name?            string
         +--rw ip-address?             yang:phys-address
         +--rw technology?             identityref
         +--rw encapsulation-method?   identityref
         +--rw (signaling-protocol-type)?
         |  +--:(signaling-protocol-null)
         |     +--rw signaling-null?         empty
         +--rw tunnel-source
         |  +--rw (tunnel-source)?
         |     +--:(interface)
         |     |  +--rw interface-type?   leafref
         |     |  +--rw interface?        if:interface-ref
         |     +--:(ipv4-address)
         |     |  +--rw ipv4-address?     inet:ipv4-address
         |     +--:(ipv6-address)
         |        +--rw ipv6-address?     inet:ipv6-address
         +--rw tunnel-destination
         |  +--rw (tunnel-destination)?
         |     +--:(interface)
         |     |  +--rw interface-type?   leafref
         |     |  +--rw interface?        if:interface-ref
         |     +--:(ipv4-address)
         |     |  +--rw ipv4-address?     inet:ipv4-address
         |     +--:(ipv6-address)
         |        +--rw ipv6-address?     inet:ipv6-address
         +--rw MTU?                    uint32
         +--rw ttl?                    uint8
         +--rw priority?               uint8
         +--rw (Multiplexing)?
         |  +--:(multiplexing-null)
         |     +--rw multiplexing-null?      empty
         +--rw (QoS)?
         |  +--:(Qos-null)
         |     +--rw QoS-null?               empty
         +--rw (Security)?
         |  +--:(Security-null)
         |     +--rw Security-null?          empty
         +--rw resource
               ......
```

o The base-interface which with type leafref is used pointing to the corresponding interface-name node in the interface yang [RFC7223]. As describe in section 3.1, each tunnel need to binding to a

physical interface[RFC7223]. In the gen-tunnel model, the base-interface leaf is playing this role.

- The tunnel-name is the identification of the tunnel. Different tunnel is distinguished via the leaf tunnel-name.

- The technology leaf indicate the technology of the tunnel such as IP, MPLS, VXLAN, etc. The default value is IP. And this allows easy extension of the YANG model by other technologies.

- The encapsulation-method leaf indicate the encapsulation method of the tunnel such as Minimal encapsulation, L2TP encapsulation, PPTP encapsulation, L2F encapsulation, UDP encapsulation, ATMP encapsulation, MSDP encapsulation, 6to4 encapsulation, 6over4 encapsulation, ISATAP encapsulation, etc. The default value is direct (no intermediate header). And this allows easy extension of the YANG model by other technologies.

- The signaling-protocol-type indicate the signaling protocol type of the tunnel such as rsvp, crldp, etc. And the default value is empty, which allows easy extension of the YANG model by other technologies.

- The multiplexing might be one which was explicitly designed for multiplexing, or one that wasn't originally designed for this but can be pushed into service as a multiplexing field. For example: the Key field for GRE, the SPI field for IPsec, etc. In gen-tunnel model, the default value of multiplexing is empty, which allows easy extension of the yang model by other technologies.

- A tunnel may not have intrinsic QoS/SLA capabilities, but it inherits whatever mechanisms exist for IP. Other mechanisms such as RSVP extensions [RFC2764] or DiffServ extensions [RFC2983], may be used with it. In gen-tunnel model, the default value of QoS/SLA capability is empty, which allows easy extension of the yang model by other technologies.

- A tunnel may provide significant security services. For example: When IPsec tunneling is used in conjunction with IPsec's cryptographic capabilities, excellent authentication and integrity functions can be provided. In gen-tunnel model, the default value of security is empty, which allows easy extension of the yang model by other technologies.

4.1.1. Resource container

The resource container is used to indicate the resources required for a tunnel. Which contains a MaxRate leaf, a MaxBurstSize leaf, an ExBurstSize leaf, a Frequency leaf and a weight leaf.

```
module: gen-tunnel
augment /if:interfaces/if:interface:
   +--rw base-interface?         if:interface-ref
   ......
   +--rw resource
      +--rw bandwidth?      uint32
      +--rw MaxRate?        uint32
      +--rw MaxBurstSize?   uint32
      +--rw ExBurstSize?    uint32
      +--rw Frequency?      uint32
      +--rw weight?         uint32
```

- The bandwidth indicate the bandwidth required of this tunnel.
- The MaxRate indicate the maximum rate of this tunnel.
- The MaxBurstSize indicate the maximum burst size of this tunnel.
- The ExBurstSize indicate the excess burst size of this tunnel.
- The Frequency indicate the granularity of the availability of committed rate.
- The weight indicate the relative weight for using excess bandwidth above its committed rate.

4.2. tunnel state model

The tunnel state model augments the "interface-state" lists defined in the "ietf-interfaces" module [RFC7223] with tunnel specific state data. On the top of the tunnel state model, we defines a tunnel list, each tunnel within it corresponding to a tunnel instance. Within each tunnel, we present an admin-status, an oper-status , MTU, packets input/output, input/output error, input/output utility rate and the number of bytes.

```
module: gen-tunnel
augment /if:interfaces-state:
   +--ro tunnel-state
      +--ro tunnel* [tunnel-name]
         +--ro tunnel-name            leafref
         +--ro admin-status?          enumeration
         +--ro oper-status?           enumeration
         +--ro MTU?                   uint32
         +--ro packets-input?         yang:counter64
         +--ro input-errors?          yang:counter64
         +--ro packets-output?        yang:counter64
         +--ro output-errors?         yang:counter64
         +--ro input-utility-rate?    yang:counter64
         +--ro output-utility-rate?   yang:counter64
         +--ro transmitted-bytes?                yang:counter64
```

5. Gen-tunnel yang data hierarchy

The complete data hierarchy related to the Tunnel YANG model is presented below. The following notations are used within the data tree and carry the meaning as below.

Each node is printed as:

```
 <status> <flags> <name> <opts> <type>
```

```
        <status> is one of:
             +  for current
             x  for deprecated
             o  for obsolete


        <flags> is one of:

            rw for configuration data
            ro for non-configuration data
            -x for rpcs
            -n for notifications


        <name> is the name of the node
```

If the node is augmented into the tree from another module, its name is printed as <prefix>:<name>.

```
            <opts> is one of:

                 ?  for an optional leaf or choice
                 !  for a presence container
                 *  for a leaf-list or list
                 [<keys>] for a list's keys

            <type> is the name of the type for leafs and leaf-lists

   module: gen-tunnel
   augment /if:interfaces/if:interface:
            +--rw tunnel-configuration
               +--rw base-interface?         if:interface-ref
               +--rw tunnel-name?            string
               +--rw ip-address?             yang:phys-address
               +--rw technology?             identityref
               +--rw encapsulation-method?   identityref
               +--rw (signaling-protocol-type)?
               |  +--:(signaling-protocol-null)
               |     +--rw signaling-null?         empty
               +--rw tunnel-source
               |  +--rw (tunnel-source)?
               |     +--:(interface)
               |     |  +--rw interface-type?   leafref
               |     |  +--rw interface?        if:interface-ref
               |     +--:(ipv4-address)
               |     |  +--rw ipv4-address?     inet:ipv4-address
               |     +--:(ipv6-address)
               |        +--rw ipv6-address?     inet:ipv6-address
               +--rw tunnel-destination
               |  +--rw (tunnel-destination)?
               |     +--:(interface)
               |     |  +--rw interface-type?   leafref
               |     |  +--rw interface?        if:interface-ref
               |     +--:(ipv4-address)
               |     |  +--rw ipv4-address?     inet:ipv4-address
               |     +--:(ipv6-address)
               |        +--rw ipv6-address?     inet:ipv6-address
               +--rw MTU?                    uint32
               +--rw ttl?                    uint8
               +--rw priority?               uint8
               +--rw (Multiplexing)?
               |  +--:(multiplexing-null)
               |     +--rw multiplexing-null?      empty
               +--rw (QoS)?
               |  +--:(Qos-null)
               |     +--rw QoS-null?               empty
               +--rw (Security)?
```

```
               |  +--:(Security-null)
               |     +--rw Security-null?          empty
               +--rw resource
                  +--rw bandwidth?      uint32
                  +--rw MaxRate?        uint32
                  +--rw MaxBurstSize?   uint32
                  +--rw ExBurstSize?    uint32
                  +--rw Frequency?      uint32
                  +--rw weight?         uint32
         augment /if:interfaces-state:
            +--ro tunnel-state
               +--ro tunnel* [tunnel-name]
                  +--ro tunnel-name            leafref
                  +--ro admin-status?          enumeration
                  +--ro oper-status?           enumeration
                  +--ro MTU?                   uint32
                  +--ro packets-input?         yang:counter64
                  +--ro input-errors?          yang:counter64
                  +--ro packets-output?        yang:counter64
                  +--ro output-errors?         yang:counter64
                  +--ro input-utility-rate?    yang:counter64
                  +--ro output-utility-rate?   yang:counter64
                  +--ro transmitted-bytes?     yang:counter64
```

data hierarchy of TUNNEL

6. Tunnel YANG Module

```
<CODE BEGINS> file "ietf-tunnel.yang"
   module gen-tunnel {
     namespace "http://example.com/gen-tunnel";
     prefix "tunnel";
     import ietf-yang-types { prefix yang;}
 import ietf-inet-types { prefix inet;}
     import ietf-interfaces {
       prefix if;
     }
     import iana-if-type {
       prefix ianaift;
     }
organization "IETF Netmod Working Group";
     contact
       "wangzitao@huawei.com";
     description
       "This module defines ietf tunnel yang data model";

     revision 2015-06-09 {
```

```
        description
         "Initial revision. - 01 version";
         reference "draft-wwz-netmod-yang-tunnel-model-00";
        }
       identity technology-types {
        description
        "this is the base identity of technology types which are
         IP, GRE, MPLS, etc";
       }

       identity ipv4 {
        base technology-types;
        description
         "technology of ipv4";
       }

       identity ipv6 {
        base technology-types;
        description
         "technology of ipv6";
       }

   identity encapsulation-type{
    description
    "The encapsulation method used by a tunnel
    which are direct, GRE, 6to4, 6over4, IPsec, etc.";
   }

   identity direct{
    base encapsulation-type;
    description
    "no intermediate header.";
   }

   identity endpoint-type{
    description
    "this is the base identity of tunnel type which are CE, PE, etc.";
   }
        augment "/if:interfaces/if:interface" {
          when "if:type = 'ianaift:tunnel'";
     description
     "Parameters for configuring tunnel on interfaces.";
         container tunnel-configuration{
      description
      "tunnel configuration model.";
          leaf base-interface {
            type if:interface-ref;
            must "/if:interfaces/if:interface[if:name = current()]" {
```

```
              description
                "The base interface.";
            }
    description
                "The base interface.";
          }

      leaf tunnel-name{
      type string;
      description
      "this name can be used to distinguish different tunnel";
     }

      leaf ip-address{
       type yang:phys-address;
           description
   "ip-address of this tunnel interface";
      }
        leaf technology{
      type identityref{
       base technology-types;
      }
      description
      "this leaf can be used to indicate different technologies
       such as IP, GRE, MPLS, etc";
     }

         leaf encapsulation-method{
      type identityref{
       base encapsulation-type;
      }
      description
      "The encapsulation method used by a tunnel.";
     }


         leaf endpoint-type{
      type identityref{
       base endpoint-type;
      }
      description
      "The endpoint type.";
     }

       choice signaling-protocol-type {
         case signaling-protocol-null {
           description
             "this is a placeholder when no signaling protocol
```

```
     is needed";
           leaf signaling-null {
              type empty;
              description
                "there is no signaling protocol define,
      it will be defined in
      technology specific model.";
           }
         }
      description
      "signaling protocol type";
       }



       container tunnel-source{
    description
    "parameters of source tunnel";
        choice tunnel-source {
         case interface{
     leaf interface-type{
       type leafref{
    path "/if:interfaces/if:interface"
    +"/if:type";
   }
   description
   "interface type";
      }
      leaf interface{
      type if:interface-ref;
      description
       "Interface";
   }
     }
         case ipv4-address {
           leaf ipv4-address {
             type inet:ipv4-address;
             description
               "Ipv4 Address";
           }
           description
             "Ip Address based node Addressing.";
         }
         case ipv6-address {
           leaf ipv6-address {
             type inet:ipv6-address;
             description
               "Ipv6 Address";
```

```
          }
          description
            "ipv6 Address based node Addressing.";
        }
        description
          "node Addressing.";
       }
   }

   container tunnel-destination{
    description
    "Data nodes for the operational state of tunnel on interfaces";
        choice tunnel-destination {
         case interface{
     leaf interface-type{
      type leafref{
    path "/if:interfaces/if:interface"
    +"/if:type";
   }
   description
   "interface type";
      }
      leaf interface{
      type if:interface-ref;
      description
       "Interface";
   }
     }
         case ipv4-address {
           leaf ipv4-address {
             type inet:ipv4-address;
             description
               "Ipv4 Address";
           }
           description
             "Ip Address based node Addressing.";
         }
         case ipv6-address {
           leaf ipv6-address {
             type inet:ipv6-address;
             description
               "Ipv6 Address";
           }
           description
             "ipv6 Address based node Addressing.";
         }
         description
           "node Addressing.";
```

```
        }
       }

    leaf MTU{
    type uint32;
    description
    "Maximum Transmission Unit";
   }

   leaf ttl {
        type uint8;
        default "255";
        description
        "Time to Live";
       }

       leaf priority {
         type uint8;
         description
           "priority";
       }

   choice Multiplexing{
    description
    "multiplexing parameters";
    case multiplexing-null{
     description
         "this is a placeholder when no multiplexing is needed";
     leaf multiplexing-null{
      type empty;
      description
          "there is no multiplexing define, it will be defined
           in technology specific model.";
     }
    }
   }

      choice QoS{
    description
    "QoS Parameters";
    case Qos-null{
     description
         "this is a placeholder when no QoS is needed";
     leaf QoS-null{
      type empty;
      description
          "there is no QoS define, it will be defined
           in technology specific model.";
```

```
     }
    }
   }

   choice Security{
    description
    "security parameters";
    case Security-null{
     description
         "this is a placeholder when no Security is needed";
     leaf Security-null{
      type empty;
      description
          "there is no Security define, it will be defined
           in technology specific model.";
     }
    }
   }
   container resource{
    // if-feature resource-support;
     description
     "this container is used to indicate the resources required
          for a tunnel.";

      leaf bandwidth{
       type uint32;
   description
   "the bandwidth of tunnel";
      }

      leaf MaxRate{
       type uint32;
   description
   "The maximum rate in bits/second.";
      }

      leaf MaxBurstSize{
       type uint32;
   description
   "The maximum burst size in bytes.";
      }

      leaf ExBurstSize{
       type uint32;
   description
   "The Excess burst size in bytes.";
      }
```

```
      leaf Frequency{
       type uint32;
   description
   "The granularity of the availability of committed
            rate.";
      }

      leaf weight{
       type uint32;
   description
   "The relative weight for using excess bandwidth above
            its committed rate.";
      }
    }
        }
       }
        augment "/if:interfaces-state" {
          when "if:type = 'ianaift:tunnel'";
     description
     "Data nodes for the operational state of tunnel on interfaces";
     container tunnel-state{
    config false;
    description
    "define the state of this tunnel";
    list tunnel{
     key "tunnel-name";
     description
     "The list of tunnel on the interface";
     leaf tunnel-name{
      type leafref{
       path "/if:interfaces/if:interface"
      +"/tunnel:tunnel-configuration/tunnel:tunnel-name";
     }
      description
      "this leaf can be used to distinguish different tunnels";
     }

          leaf admin-status{
           type enumeration {
            enum up {
             value 1;
             description
             "Ready to pass packets.";
            }
            enum down {
             value 2;
             description
             "Not ready to pass packets and not in some test mode.";
```

```
            }
            enum testing {
             value 3;
             description
             "In some test mode.";
            }
           }
           description
           "The desired state of the tunnel.";
          }

          leaf oper-status {
           type enumeration {
            enum up {
             value 1;
             description
             "Ready to pass packets.";
            }
            enum down {
             value 2;
             description
             "The tunnel does not pass any packets.";
            }
            enum testing {
             value 3;
             description
             "In some test mode. No operational packets can
              be passed.";
            }
            enum unknown {
             value 4;
             description
             "Status cannot be determined for some reason.";
            }
            enum dormant {
             value 5;
             description
             "Waiting for some external event.";
            }
            enum not-present {
             value 6;
             description
             "Some component (typically hardware) is missing.";
            }
            enum lower-layer-down {
             value 7;
             description
             "Down due to state of lower-layer tunnel(s).";
```

```
          }
         }
         description
         "The current operational state of the tunnel.";
        }

      leaf MTU{
       type uint32;
    description
    "Maximum Transmit Uint";
      }

        leaf packets-input{
     type yang:counter64;
     description
     "Number of  packets input.";
    }

    leaf input-errors{
     type yang:counter64;
     description
     "Number of input packets dropped because of errors or for
          other reasons.";
    }

        leaf packets-output{
     type yang:counter64;
     description
     "Number of  packets output.";
    }

    leaf output-errors{
     type yang:counter64;
     description
     "Number of output packets dropped because of errors or for
          other reasons.";
    }

    leaf input-utility-rate{
     type yang:counter64;
     description
     "input utility rate.";
    }

    leaf output-utility-rate{
     type yang:counter64;
     description
     "output utility rate.";
```

```
    }

    leaf bytes{
     type yang:counter64;
     description
     "Number of bytes forwarded by the tunnel.";
    }
   }
  }
  }
    }
```

<CODE ENDS>

7. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol [RFC6241] . The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242] . The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

8. IANA Considerations

This document registers a URI in the IETF XML registry [RFC3688] . Following the format in RFC 3688, the following registration is requested to be made:

URI: urn:ietf:params:xml:ns:yang:ietf-tunnel

Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.

This document registers a YANG module in the YANG Module Names registry [RFC6020].

name: ietf-tunnel

namespace:urn:ietf:params:xml:ns:yang:ietf-tunnel

prefix: itun reference: RFC XXXX

9. Normative References

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

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

Authors' Addresses

Aijun Wang
China Telecom
No.118,Xizhimenneidajie,Xicheng District
Beijing 100035
China

Email: wangaj@ctbri.com.cn

Zitao Wang
Huawei Technologies,Co.,Ltd
101 Software Avenue, Yuhua District
Nanjing 210012
China

Email: wangzitao@huawei.com

Yan Zhuang
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: zhuangyan.zhuang@huawei.com