Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2019

V. Vassilev
Transpacket

March 9, 2019

A YANG Data Model for Network Interconnect Tester Management
draft-vassilev-bmwg-network-interconnect-tester-00

Abstract

This document introduces new YANG model for use in network interconnect testing containing modules for traffic generator, traffic analyzer and internal interface loopback.

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 10, 2019.

Table of Contents

1. Introduction

There is a need for standard mechanism to allow the specification and implementation of the transactions part of network tests. The mechanism should allow the control and monitoring of the data plane traffic in a transactional way. In addition to that the mechanism should allow the configuration of internal near-end and far-end interface loopbacks. This document defines YANG modules for test traffic generator, analyzer and internal interface loopback.

1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

1.1.1. Definitions and Acronyms

DUT: Device Under Test

TA: Traffic Analyzer

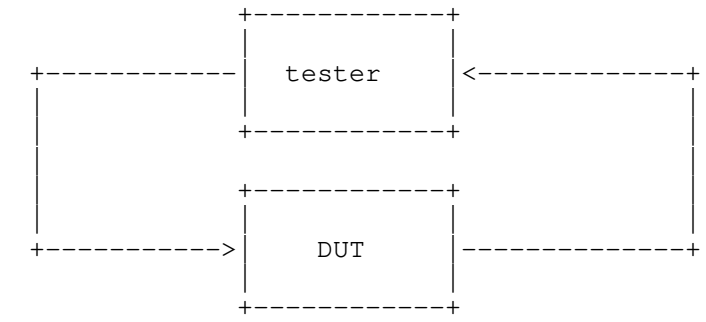
TG: Traffic Generator

1.1.2. Tree Diagram

For a reference to the annotations used in tree diagrams included in this draft, please see YANG Tree Diagrams [RFC8340].

1.2. Problem Statement

Network interconnect tests require active network elements part of the tested network that generate test traffic and network elements that analyze the test traffic at one or more points of its path. A Network interconnect tester is a device that can either generate test traffic, analyze test traffic or both. Here is a figure borrowed from [RFC2544] representing the horseshoe test setup topology consisting of a single tester and a single DUT connected in a network interconnect loop.

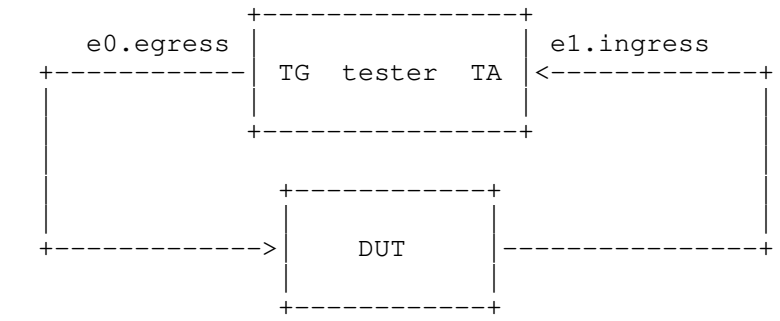


This document attempts to address the problem of defining YANG model of a network interconnect tester that can be used for development of vendor independent network interconnect tests and utilize the advantages of transactional management using standard protocols like NETCONF.

1.3. Solution

The proposed model splits the design into 3 modules - 1) Traffic Generator module (TG), 2) Traffic Analyzer module (TA) and adds an additional module 3) Interface loopback module (LB) addressing

configuration of internal interface loopback mode that is a common requirement for many network interconnect tests. The modules are implemented as augmentations of the ietf-interfaces module adding configuration and state data that models the functionality of a tester. The TA and TG modules concept is illustrated with the following diagram of a tester with two interfaces (named e0 and e1) conected in a loop with single DUT:



2. Using the network interconnect tester model

Basic example of how the model can be used in transactional network test API to control the testers part of a network and report countrer statistics and timing measurement data is presented in Appendix A. One of the examples demonstrates the use of the [RFC2544] defined testframe packet.

3. Traffic Generator Module Tree Diagram

```
module: ietf-traffic-generator
  augment /if:interfaces/if:interface:
    +--rw traffic-generator {egress-direction}?
    |  +--rw (type)?
    |  |  +--:(single-stream)
    |  |  |  +--rw frame-size          uint32
    |  |  |  +--rw (frame-data-type)?
    |  |  |  |  +--:(raw-frame-data)
    |  |  |  |     +--rw frame-data?   string
    |  |  |  +--rw interframe-gap      uint32
    |  |  |  +--rw interburst-gap?     uint32
    |  |  |  +--rw frames-per-burst?   uint32
    |  |  |  +--rw src-mac-address?    yang:mac-address {ethernet}?
    |  |  |  +--rw dst-mac-address?    yang:mac-address {ethernet}?
    |  |  |  +--rw ether-type?         uint16 {ethernet}?
    |  |  |  +--rw (encapsulation)? {ethernet}?
    |  |  |     +--:(vlan)
```

```
   |  |  |        +--rw vlan {ethernet-vlan}?
   |  |  |           +--rw id      uint16
   |  |  |           +--rw tpid?   uint16
   |  |  |           +--rw pcp?    uint8
   |  |  |           +--rw cfi?    uint8
   |  |  +--:(multi-stream)
   |  |     +--rw streams
   |  |        +--rw stream* [id]
   |  |           +--rw id                    uint32
   |  |           +--rw frame-size            uint32
   |  |           +--rw (frame-data-type)?
   |  |           |  +--:(raw-frame-data)
   |  |           |     +--rw frame-data?     string
   |  |           +--rw interframe-gap        uint32
   |  |           +--rw interburst-gap?       uint32
   |  |           +--rw frames-per-burst?     uint32
   |  |           +--rw frames-per-stream     uint32
   |  |           +--rw interstream-gap       uint32
   |  |           +--rw src-mac-address?      yang:mac-address {ethernet}?
   |  |           +--rw dst-mac-address?      yang:mac-address {ethernet}?
   |  |           +--rw ether-type?           uint16 {ethernet}?
   |  |           +--rw (encapsulation)? {ethernet}?
   |  |              +--:(vlan)
   |  |                 +--rw vlan {ethernet-vlan}?
   |  |                    +--rw id      uint16
   |  |                    +--rw tpid?   uint16
   |  |                    +--rw pcp?    uint8
   |  |                    +--rw cfi?    uint8
   |  +--rw total-frames?             uint64
   +--rw traffic-generator-ingress {ingress-direction}?
      +--rw (type)?
         |  +--:(single-stream)
         |  |  +--rw frame-size          uint32
         |  |  +--rw (frame-data-type)?
         |  |  |  +--:(raw-frame-data)
         |  |  |     +--rw frame-data?   string
         |  |  +--rw interframe-gap      uint32
         |  |  +--rw interburst-gap?     uint32
         |  |  +--rw frames-per-burst?   uint32
         |  |  +--rw src-mac-address?    yang:mac-address {ethernet}?
         |  |  +--rw dst-mac-address?    yang:mac-address {ethernet}?
         |  |  +--rw ether-type?         uint16 {ethernet}?
         |  |  +--rw (encapsulation)? {ethernet}?
         |  |     +--:(vlan)
         |  |        +--rw vlan {ethernet-vlan}?
         |  |           +--rw id      uint16
         |  |           +--rw tpid?   uint16
         |  |           +--rw pcp?    uint8
```

```
       |  |           +--rw cfi?    uint8
       |  +--:(multi-stream)
       |     +--rw streams
       |        +--rw stream* [id]
       |           +--rw id                    uint32
       |           +--rw frame-size            uint32
       |           +--rw (frame-data-type)?
       |           |  +--:(raw-frame-data)
       |           |     +--rw frame-data?     string
       |           +--rw interframe-gap        uint32
       |           +--rw interburst-gap?       uint32
       |           +--rw frames-per-burst?     uint32
       |           +--rw frames-per-stream     uint32
       |           +--rw interstream-gap       uint32
       |           +--rw src-mac-address?      yang:mac-address {ethernet}?
       |           +--rw dst-mac-address?      yang:mac-address {ethernet}?
       |           +--rw ether-type?           uint16 {ethernet}?
       |           +--rw (encapsulation)? {ethernet}?
       |              +--:(vlan)
       |                 +--rw vlan {ethernet-vlan}?
       |                    +--rw id      uint16
       |                    +--rw tpid?   uint16
       |                    +--rw pcp?    uint8
       |                    +--rw cfi?    uint8
       |
       +--rw total-frames?             uint64
  augment /if:interfaces-state/if:interface/if:statistics:
    +--ro generated-pkts?              yang:counter64
    +--ro generated-octets?            yang:counter64
    +--ro generated-ingress-pkts?      yang:counter64 {ingress-direction}?
    +--ro generated-ingress-octets?    yang:counter64 {ingress-direction}?
```

4. Traffic Analyzer Module Tree Diagram

```
module: ietf-traffic-analyzer
  augment /if:interfaces/if:interface:
    +--rw traffic-analyzer! {ingress-direction}?
    |  +--rw filter! {filter}?
    |  |  +--rw type          identityref
    |  |  +--rw ether-type?   uint16
    |  +--ro state
    |     +--ro pkts?              yang:counter64
    |     +--ro errors?            yang:counter64
    |     +--ro testframe-stats
    |     |  +--ro testframe-pkts?    yang:counter64
    |     |  +--ro sequence-errors?   yang:counter64
    |     |  +--ro payload-errors?    yang:counter64
    |     |  +--ro latency
    |     |     +--ro samples?   uint64
```

```
     |      |      +--ro min?       uint64
     |      |      +--ro max?       uint64
     |      |      +--ro average?   uint64
     |      |      +--ro latest?    uint64
     |      +--ro capture {capture}?
     |         +--ro frame* [sequence-number]
     |            +--ro sequence-number             uint64
     |            +--ro timestamp?                  yang:date-and-time
     |            +--ro length?                     uint32
     |            +--ro preceding-interframe-gap?   uint32
     |            +--ro data?                       string
     +--rw traffic-analyzer-egress! {egress-direction}?
        +--rw filter! {filter}?
        |  +--rw type    identityref
        +--ro state
           +--ro pkts?             yang:counter64
           +--ro errors?           yang:counter64
           +--ro testframe-stats
           |  +--ro testframe-pkts?    yang:counter64
           |  +--ro sequence-errors?   yang:counter64
           |  +--ro payload-errors?    yang:counter64
           |  +--ro latency
           |     +--ro samples?   uint64
           |     +--ro min?       uint64
           |     +--ro max?       uint64
           |     +--ro average?   uint64
           |     +--ro latest?    uint64
           +--ro capture {capture}?
              +--ro frame* [sequence-number]
                 +--ro sequence-number             uint64
                 +--ro timestamp?                  yang:date-and-time
                 +--ro length?                     uint32
                 +--ro preceding-interframe-gap?   uint32
                 +--ro data?                       string
  augment /if:interfaces-state/if:interface/if:statistics:
    +--ro testframe-pkts?              yang:counter64 {ingress-direction}?
    +--ro testframe-sequence-errors?   yang:counter64 {ingress-direction}?
    +--ro testframe-payload-errors?    yang:counter64 {ingress-direction}?
  augment /if:interfaces-state/if:interface/if:statistics:
    +--ro testframe-egress-pkts?              yang:counter64 {egress-direction}?
    +--ro testframe-egress-sequence-errors?   yang:counter64 {egress-direction}?
    +--ro testframe-egress-payload-errors?    yang:counter64 {egress-direction}?
```

5. Loopback Module Tree Diagram

```
   module: ietf-loopback
     augment /if:interfaces/if:interface:
       +--rw loopback?   identityref
```

6. Traffic Generator Module YANG

<CODE BEGINS> file "ietf-traffic-generator@2019-03-09.yang"

```
module ietf-traffic-generator {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-traffic-generator";
  prefix tg;

  import ietf-interfaces {
    prefix if;
  }
  import ietf-yang-types {
    prefix yang;
  }
  import iana-if-type {
    prefix ianaift;
  }

  organization
    "IETF Benchmarking Methodology Working Group";
  contact
    "WG Web:   <http://tools.ietf.org/wg/bmwg/>
     WG List:  <mailto:bmwg@ietf.org>

     Editor:   Vladimir Vassilev
               <mailto:vladimir@transpacket.com>";
  description
    "This module contains a collection of YANG definitions for
     description and management of network interconnect testers.

     Copyright (c) 2019 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 XXXX; see
     the RFC itself for full legal notices.";

  revision 2019-03-09 {
    description
      "Initial revision.";
    reference "RFC XXXX: Network Interconnect Tester";
```

```
  }

  feature egress-direction {
    description
      "The device can generate traffic in the egress direction.";
  }

  feature ingress-direction {
    description
      "The device can generate traffic in the ingress direction.";
  }

  feature multi-stream {
    description
      "The device can generate multi-stream traffic.";
  }

  feature ethernet {
    description
      "The device can generate ethernet traffic.";
  }

  feature ethernet-vlan {
    if-feature "ethernet";
    description
      "The device can generate vlan tagged ethernet traffic.";
  }

  grouping traffic-generator-burst-data {
    leaf frame-size {
      type uint32;
      description
        "Size of the frames generated. For example for
         ethernet interfaces the following definition
         applies:

         Ethernet frame-size in octets includes:
         * Destination Address (6 octets),
         * Source Address (6 octets),
         * Frame Type (2 octets),
         * Data (min 46 octets or 42 octets + 4 octets 802.1Q tag),
         * CRC Checksum (4 octets).

         Ethernet frame-size does not include:
         * Preamble (dependent on MAC configuration
                     by default 7 octets),
         * Start of frame delimeter (1 octet)
```

```
         Minimum standard ethernet frame-size is 64 bytes but
         generators might support smaller sizes for validation.";
      mandatory true;
    }
    choice frame-data-type {
      case raw-frame-data {
        leaf frame-data {
          type string {
            pattern '([0-9A-F]{2})*';
          }
          must 'string-length(.)<=(../frame-size*2)';
          description
            "The raw frame data specified as hexadecimal string.
             The specified data can be shorter then the ../frame-size
             value specifying only the header or the header and the
             payload without for example the 4 byte CRC Checksum
             in the case of a Ethernet frame.";
        }
      }
    }
    leaf interframe-gap {
      type uint32;
      description
        "Length of the idle period between generated frames.
         For example for ethernet interfaces the following
         definition applies:

         Ethernet interframe-gap between transmission of frames
         known as the interframe gap (IFG). A brief recovery time
         between frames allows devices to prepare for
         reception of the next frame. The minimum
         interframe gap is 96 bit times (12 octet times) (the time it
         takes to transmit 96 bits (12 octets) of raw data on the
         medium). However the preamble (7 octets) and start of
         frame delimeter (1 octet) are considered a constant gap that
         should be included in the interframe-gap. Thus the minimum
         value for standard ethernet transmission should be considered
         20 octets.";
      mandatory true;
    }
    leaf interburst-gap {
      type uint32;
      description
        "Similar to the interframe-gap but takes place between
         any two bursts of the stream.";
    }
    leaf frames-per-burst {
      type uint32;
```

```
      description
        "Number of frames contained in a burst";
    }
  }

  grouping traffic-generator-multi-stream-data {
    container streams {
      list stream {
        key "id";
        leaf id {
          type uint32;
          description
            "Number specifying the order of the stream.";
        }
        uses traffic-generator-burst-data;
        leaf frames-per-stream {
          type uint32;
          description
            "The count of frames to be generated before
             generation of the next stream is started.";
          mandatory true;
        }
        leaf interstream-gap {
          type uint32;
          description
            "Idle period after the last frame of the last burst.";
          mandatory true;
        }
      }
    }
  }

  augment "/if:interfaces/if:interface" {
    container traffic-generator {
      if-feature "egress-direction";
      choice type {
        case single-stream {
          uses traffic-generator-burst-data;
        }
        case multi-stream {
          uses traffic-generator-multi-stream-data;
        }
      }
      leaf total-frames {
        type uint64;
        description
          "If this leaf is present the stream generation will stop
           after the specified number of frames are generated.";
```

```
      }
    }
    container traffic-generator-ingress {
      if-feature "ingress-direction";
      choice type {
        case single-stream {
          uses traffic-generator-burst-data;
        }
        case multi-stream {
          uses traffic-generator-multi-stream-data;
        }
      }
      leaf total-frames {
        type uint64;
        description
          "If this leaf is present the stream generation will stop
           after the specified number of frames are generated.";
      }
    }
  }
  augment "/if:interfaces-state/if:interface/if:statistics" {
    description
      "Counters of generated traffic octets and packets.";
    leaf generated-pkts {
      type yang:counter64;
      description
        "Traffic generator packets sent.";
    }
    leaf generated-octets {
      type yang:counter64;
      description
        "Traffic generator octets sent.";
    }
    leaf generated-ingress-pkts {
      if-feature "ingress-direction";
      type yang:counter64;
      description
        "Traffic generator packets generated in ingress mode.";
    }
    leaf generated-ingress-octets {
      if-feature "ingress-direction";
      type yang:counter64;
      description
        "Traffic generator octets generated in ingress mode.";
    }
  }

  grouping ethernet-data {
```

```
    leaf src-mac-address {
      type yang:mac-address;
    }
    leaf dst-mac-address {
      type yang:mac-address;
    }
    leaf ether-type {
      type uint16;
      description
        "The Ethernet Type (or Length) value
         defined by IEEE 802.";
      reference "IEEE 802-2014 Clause 9.2";
    }
    choice encapsulation {
      case vlan {
        container vlan {
          if-feature "ethernet-vlan";
          leaf id {
            type uint16 {
              range "0..4095";
            }
            mandatory true;
          }
          leaf tpid {
            type uint16;
            default "33024";
            description
              "Configures the Tag Protocol Identifier (TPID)
               of the 802.1q VLAN tag sent. This value is used
               together whith the vlan id for filtering incoming
               vlan tagged packets.";
          }
          leaf pcp {
            type uint8 {
              range "0..7";
            }
            default "0";
            description
              "Configures the IEEE 802.1p Priority Code Point (PCP) value
               of the transmitted 802.1q VLAN tag.";
          }
          leaf cfi {
            type uint8 {
              range "0..1";
            }
            default "0";
            description
              "Configures the Canonical Format Identifier (CFI) field
```

```
                (shall be 0 for Ethernet switches) of the transmitted
                802.1q VLAN tag.";
            }
          }
        }
      }
    }

    augment "/if:interfaces/if:interface/tg:traffic-generator/tg:type/"
          + "tg:single-stream" {
      if-feature "ethernet";
      when "derived-from-or-self(../if:type, 'ianaift:ethernetCsmacd')" {
        description
          "Ethernet interface type.";
      }
      uses ethernet-data;
    }
    augment "/if:interfaces/if:interface/tg:traffic-generator/tg:type/"
          + "tg:multi-stream/tg:streams/tg:stream" {
      if-feature "ethernet";
      when "derived-from-or-self(../../../if:type, 'ianaift:ethernetCsmacd')" {
        description
          "Ethernet interface type.";
      }
      uses ethernet-data;
    }
    augment "/if:interfaces/if:interface/tg:traffic-generator-ingress/tg:type/"
          + "tg:single-stream" {
      if-feature "ethernet";
      when "derived-from-or-self(../if:type, 'ianaift:ethernetCsmacd')" {
        description
          "Ethernet interface type.";
      }
      uses ethernet-data;
    }
    augment "/if:interfaces/if:interface/tg:traffic-generator-ingress/tg:type/"
          + "tg:multi-stream/tg:streams/tg:stream" {
      if-feature "ethernet";
      when "derived-from-or-self(../../../if:type, 'ianaift:ethernetCsmacd')" {
        description
          "Ethernet interface type.";
      }
      uses ethernet-data;
    }
}
```

<CODE ENDS>

7. Traffic Analyzer Module YANG

<CODE BEGINS> file "ietf-traffic-analyzer@2019-03-09.yang"

```
 module ietf-traffic-analyzer {
   yang-version 1.1;
   namespace "urn:ietf:params:xml:ns:yang:ietf-traffic-analyzer";
   prefix ta;

   import ietf-interfaces {
     prefix if;
   }
   import ietf-yang-types {
     prefix yang;
   }

   organization
     "IETF Benchmarking Methodology Working Group";
   contact
     "WG Web:   <http://tools.ietf.org/wg/bmwg/>
      WG List:  <mailto:bmwg@ietf.org>

      Editor:   Vladimir Vassilev
                <mailto:vladimir@transpacket.com>";
   description
     "This module contains a collection of YANG definitions for
      description and management of network interconnect testers.

      Copyright (c) 2019 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 XXXX; see
      the RFC itself for full legal notices.";

   revision 2019-03-09 {
     description
       "Initial revision.";
     reference "RFC XXXX: Network Interconnect Tester";
   }

   feature egress-direction {
```

```
    description
      "The device can analyze traffic from the egress direction.";
  }

  feature ingress-direction {
    description
      "The device can generate traffic from the ingress direction.";
  }

  feature filter {
    description
      "This feature indicates that the device implements
       filter that can specify a subset of packets to be
       analyzed when filtering is enabled.";
  }

  feature capture {
    description
      "This feature indicates that the device implements
       packet capture functionality.";
  }

  identity filter {
    description
      "Base filter identity.";
  }

  identity ethernet {
    base ta:filter;
  }

  grouping statistics-data {
    leaf pkts {
      type yang:counter64;
    }
    leaf errors {
      type yang:counter64;
    }
    container testframe-stats {
      leaf testframe-pkts {
        type yang:counter64;
      }
      leaf sequence-errors {
        type yang:counter64;
      }
      leaf payload-errors {
        type yang:counter64;
      }
```

```
      container latency {
        leaf samples {
          type uint64;
        }
        leaf min {
          units "nanoseconds";
          type uint64;
        }
        leaf max {
          units "nanoseconds";
          type uint64;
        }
        leaf average {
          description
            "The sum of all sampled latencies divided
             by the number of samples.";
          units "nanoseconds";
          type uint64;
        }
        leaf latest {
          units "nanoseconds";
          type uint64;
        }
      }
    }
  }

  grouping capture-data {
    container capture {
      if-feature "capture";
      list frame {
        key "sequence-number";
        leaf sequence-number {
          type uint64;
        }
        leaf timestamp {
          type yang:date-and-time;
        }
        leaf length {
          type uint32;
        }
        leaf preceding-interframe-gap {
          type uint32;
        }
        leaf data {
          type string {
            pattern '([0-9A-F]{2})*';
          }
```

```
        }
      }
    }
  }

  grouping filter-data {
    container filter {
      presence
        "When present ingress packets are
         filtered before analyzed according
         to the filter type";
      if-feature "filter";
      leaf type {
        mandatory true;
        type identityref {
          base ta:filter;
        }
      }
    }
  }

  augment "/if:interfaces/if:interface" {
    container traffic-analyzer {
      if-feature "ingress-direction";
      presence "Enables the traffic analyzer for ingress traffic.";
      uses filter-data;
      container state {
        config false;
        uses statistics-data;
        uses capture-data;
      }
    }
    container traffic-analyzer-egress {
      if-feature "egress-direction";
      presence "Enables the traffic analyzer for egress traffic.";
      uses filter-data;
      container state {
        config false;
        uses statistics-data;
        uses capture-data;
      }
    }
  }
  augment "/if:interfaces/if:interface/ta:traffic-analyzer/ta:filter" {
    when "ta:type = 'ta:ethernet'";
    leaf ether-type {
      type uint16;
      description
```

```
        "The Ethernet Type (or Length) value
         defined by IEEE 802.";
      reference "IEEE 802-2014 Clause 9.2";
    }
  }
  augment "/if:interfaces-state/if:interface/if:statistics" {
    if-feature "ingress-direction";
    description
      "Counters implemented by ports with analyzers.";
    leaf testframe-pkts {
      type yang:counter64;
      description
        "Testframe packets recognized by the traffic analyzer.";
    }
    leaf testframe-sequence-errors {
      type yang:counter64;
      description
        "Testframe packets part of the recognized total
         but with unexpected sequence number.";
    }
    leaf testframe-payload-errors {
      type yang:counter64;
      description
        "Testframe packets part of the recognized total
         but with payload errors.";
    }
  }
  augment "/if:interfaces-state/if:interface/if:statistics" {
    if-feature "egress-direction";
    description
      "Counters implemented by ports with egress analyzers.";
    leaf testframe-egress-pkts {
      type yang:counter64;
      description
        "Testframe egress packets recognized by the traffic analyzer.";
    }
    leaf testframe-egress-sequence-errors {
      type yang:counter64;
      description
        "Testframe egress packets part of the recognized total
         but with unexpected sequence number.";
    }
    leaf testframe-egress-payload-errors {
      type yang:counter64;
      description
        "Testframe egress packets part of the recognized total
         but with payload errors.";
    }
```

```
   }
 }
```

<CODE ENDS>

8. Loopback Module YANG

<CODE BEGINS> file "ietf-loopback@2019-03-09.yang"

```
module ietf-loopback {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-loopback";
  prefix loopback;

  import ietf-interfaces {
    prefix if;
  }

  organization
    "IETF Benchmarking Methodology Working Group";
  contact
    "WG Web:   <http://tools.ietf.org/wg/bmwg/>
     WG List:  <mailto:bmwg@ietf.org>

     Editor:   Vladimir Vassilev
               <mailto:vladimir@transpacket.com>";
  description
    "This module contains a collection of YANG definitions for
     description and management of network interconnect testers.

     Copyright (c) 2019 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 XXXX; see
     the RFC itself for full legal notices.";

  revision 2019-03-09 {
    description
      "Initial revision.";
    reference "RFC XXXX: Network Interconnect Tester";
  }
```

```
    identity loopback {
      description
        "Base loopback identity.";
    }

    identity near-end {
      base loopback;
      description
        "Identifies loopback mode where all local egress
         packets are looped back as ingress packets.";
    }

    identity far-end {
      base loopback;
      description
        "Identifies loopback mode where all remote ingress
         packets are looped back as egress packets.";
    }

    augment "/if:interfaces/if:interface" {
      leaf loopback {
        type identityref {
          base loopback;
        }
      }
    }
  }
```

<CODE ENDS>

9. IANA Considerations

This document registers three URIs and three YANG modules.

9.1. URI Registration

This document registers three URIs 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-traffic-generator
URI: urn:ietf:params:xml:ns:yang:ietf-traffic-analyzer
URI: urn:ietf:params:xml:ns:yang:ietf-loopback

Registrant Contact: The IESG.

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

9.2. YANG Module Name Registration

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

```
name: ietf-traffic-generator
namespace: urn:ietf:params:xml:ns:yang:ietf-traffic-generator
prefix: tg
reference: RFC XXXX

name: ietf-traffic-analyzer
namespace: urn:ietf:params:xml:ns:yang:ietf-traffic-analyzer
prefix: ta
reference: RFC XXXX

name: ietf-loopback
namespace: urn:ietf:params:xml:ns:yang:ietf-loopback
prefix: loopback
reference: RFC XXXX
```

10. Security Considerations

This document does not introduce any new security concerns in addition to those specified in [RFC7950], section 15.

11. References

11.1. Normative References

[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>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

11.2. Informative References

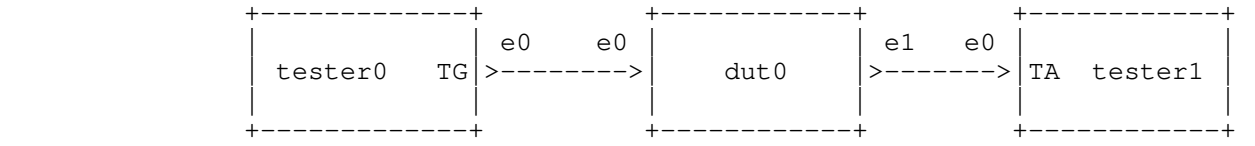
[RFC2544] Bradner, S. and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices", RFC 2544, DOI 10.17487/RFC2544, March 1999, <https://www.rfc-editor.org/info/rfc2544>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

Appendix A. Examples

The following topology will be used for the examples in this section:



A.1. Basic Test Program

This program based on transactional network test API shows how the modules can be used:

```
#Connect to network
net=tntapi.connect("topology.xml")

# Configure DUTs and enable traffic-analyzers
net.node("dut0").edit( \
  "create /interfaces/interface[name='e0'] -- type=ethernetCsmacd")
net.node("dut0").edit(
  "create /interfaces/interface[name='e1'] -- type=ethernetCsmacd")
net.node("dut0").edit(
  "create /flows/flow[id='t0'] -- match/in-port=e0 "
  "actions/action[order='0']/output-action/out-port=e0]")

net.node("tester1").edit(
  "create /interfaces/interface[name='e0']/traffic-analyzer")
net.commit()

#Get network state - before
before=net.get()
```

```
# Start traffic
net.node("tester0).edit(
  "create /interfaces/interface[name='e0']/traffic-generator -- "
  "frame-size=64 interframe-gap=20")

net.commit()

time.sleep(60)

# Stop traffic
net.node("tester1").edit("delete /interfaces/interface[name='e0']/
                         "traffic-generator")
net.commit()

#Get network state - after
after=net.get()

#Report
sent_pkts=delta("tester0",before,after,
  "/interfaces/interface[name='e0']/statistics/out-unicast-pkts")

received_pkts=delta("tester1",before,after,
  "/interfaces/interface[name='e0']/statistics/in-unicast-pkts")

latency_max=absolute(after,
  "/interfaces/interface[name='e0']/traffic-analyzer/state/"
  "testframe-stats/latency/max")

#Cleanup
net.node("tester1").edit(
   "delete /interfaces/interface/traffic-analyzer")
net.node("dut0").edit("delete /flows")
net.node("dut0").edit("delete /interfaces")
net.commit()
```

A.2. Generating RFC2544 Testframes

In sec. C.2.6.4 Test Frames a detailed format is specified. The frame-data leaf allows full control over the generated frames payload.

```
...
net.node("tester1").edit(
  "merge /interfaces/interface[name='e0']/"
  "traffic-generator -- frame-data="
  "6CA96F0000026CA96F00000108004500"
  "002ED4A500000A115816C0000201C000"
  "0202C0200007001A0000010203040506"
  "0708090A0B0C0D0E0F101112")
...
```

Author's Address

Vladimir Vassilev
Transpacket

Email: vladimir@transpacket.com