Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on September 5, 2020.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
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. This document defines YANG modules for test traffic generator, analyzer and internal interface loopback.

1.1. Terminology

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 document, 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.

![Diagram of a tester with two interfaces (named e0 and e1) connected in a loop with single DUT]

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). 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) connected 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 counter statistics and timing measurement data is presented in Appendix A. All example cases present the configuration and state data from a single test trial. The search algorithm logic that operates to control the trial configuration is outside the scope of this document. One of the examples demonstrates the use of the [RFC2544] defined testframe packet.

3. Traffic Generator Module Tree Diagram

```xml
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 vlan {ethernet-vlan,ethernet}?
    |     |     +--rw id uint16
    |     |     +--rw tpid? uint16
    |     |     +--rw pcp? uint8
    |     |     +--rw cfi? uint8
    |     +--:(multi-stream)
    |         +--rw streams
```
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. Traffic Generator Module YANG

<CODE BEGINS> file "ietf-traffic-generator@2020-03-05.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;
    reference "RFC 8343: A YANG Data Model For Interface Management";
  }
  import ietf-yang-types { 
    prefix yang;
    reference "RFC 6991: Common YANG Data Types";
  }
  import iana-if-type { 
    prefix ianaift;
    reference "RFC 7224: IANA Interface Type YANG Module";
  }

  organization "IETF Benchmarking Methodology Working Group";
  contact "WG Web:  <http://tools.ietf.org/wg/bmwg/> 
  WG List:    <mailto:bmwg@ietf.org>"
  Editor: Vladimír Vassilev  
  <mailto:vladimir@lightside-instruments.com>";
  description "This module contains a collection of YANG definitions for 
  description and management of network interconnect testers.

  Copyright (c) 2020 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 

  This version of this YANG module is part of RFC XXXX; see 
  the RFC itself for full legal notices.";

Vassilev Expires September 5, 2020
revision 2020-03-05 {
  description "Initial revision.";
  reference "RFC XXXX: A YANG Data Model for Network Interconnect Tester Management";
}

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 {
  description "Generated traffic burst parameters.";
  leaf frame-size {
    type uint32;
    mandatory true;
    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 delimiter (1 octet)

Minimum standard ethernet frame-size is 64 bytes but
generators might support smaller sizes for validation.

choice frame-data-type {
  description
  "Choice of frame data type generated.";
  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;
  mandatory true;
  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 delimiter (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.


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 {
    description
        "Multi stream traffic generation parameters."
    container streams {
        description
            "Non-presence container holding the configured stream list."
        list stream {
            key "id";
            description
                "Each stream repeats a burst until frames-per-stream
                count is reached followed by interstream-gap delay."
            leaf id {
                type uint32;
                description
                    "Number specifying the order of the stream."
            }
            uses traffic-generator-burst-data;
            leaf frames-per-stream {
                type uint32;
                mandatory true;
                description
                    "The count of frames to be generated before
                    generation of the next stream is started."
            }
            leaf interstream-gap {
                type uint32;
                mandatory true;
                description
                    "Idle period after the last frame of the last burst."
            }
        }
    }
}
grouping ethernet-data {
  description "Ethernet frame data specific parameters.";
  reference "IEEE 802-2014 Clause 9.2";
  leaf src-mac-address {
    type yang:mac-address;
    description "Source Address field of the generated Ethernet packet.";
  }
  leaf dst-mac-address {
    type yang:mac-address;
    description "Destination Address field of the generated Ethernet packet.";
  }
  leaf ether-type {
    type uint16;
    description "Length/Type field of the generated Ethernet packet.";
  }
  container vlan {
    if-feature "ethernet-vlan";
    description "VLAN tag fields..";
    leaf id {
      type uint16 {
        range "0..4095";
      }
      mandatory true;
      description "VLAN id.";
    }
    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 with 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";
    }
  }
}
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" {
  description
    "Traffic generator augmentations of ietf-interfaces.";
  container traffic-generator {
    if-feature "egress-direction";
    description
      "Traffic generator for egress direction.";
    choice type {
      description
        "Choice of the type of the data model of the generator.
         Single or multi stream.";
      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";
    description
      "Traffic generator for ingress direction.";
    choice type {
      description
        "Choice of the type of the data model of the generator.
         Single or multi stream.";
      ...
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.";
    }
}

augment "/if:interfaces/if:interface/tg:traffic-generator/tg:type/
    + "tg:single-stream" {
    when "derived-from-or-self(../if:type, 'ianaift:ethernetCsmacd')" {
        description
        "Ethernet interface type.";
    }
if-feature "ethernet";
description
"Ethernet specific augmentation for egress single stream generator type."
uses ethernet-data;
}
augment "/if:interfaces/if:interface/tg:traffic-generator/"
  + "tg:type/tg:multi-stream/tg:streams/tg:stream" {
  when "derived-from-or-self(../../../if:type,"
    + "'ianaift:ethernetCsmacd')"
  {
    description
    "Ethernet interface type.";
  }
if-feature "ethernet";
description
"Ethernet specific augmentation for egress multi stream generator type."
uses ethernet-data;
}
augment "/if:interfaces/if:interface/tg:traffic-generator-ingress/"
  + "tg:type/tg:single-stream" {
  when "derived-from-or-self(../if:type, 'ianaift:ethernetCsmacd')"
  {
    description
    "Ethernet interface type.";
  }
if-feature "ethernet";
description
"Ethernet specific augmentation for ingress single stream generator type."
uses ethernet-data;
}
6. Traffic Analyzer Module 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;
    reference "RFC 8343: A YANG Data Model For Interface Management";
  }
  import ietf-yang-types {
    prefix yang;
    reference "RFC 6991: Common YANG Data Types";
  }

  organization "IETF Benchmarking Methodology Working Group";
  contact "WG Web: http://tools.ietf.org/wg/bmwg/"
  contact "WG List: bmwg@ietf.org"
  contact "Editor: Vladimir Vassilev"
  contact "mailto:vladimir@lightside-instruments.com"
  description "This module contains a collection of YANG definitions for description and management of network interconnect testers."

  Copyright (c) 2020 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 2020-03-05 {
  description
   "Initial revision.";
  reference
   "RFC XXXX: A YANG Data Model for
     Network Interconnect Tester Management";
}

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;
  description
   "Ethernet packet fields filter.";
}

grouping statistics-data {
  description
   "Analyzer statistics.";
  leaf pkts {
    type yang:counter64;
    description
"Total number of packets analyzed."

leaf errors {
  type yang:counter64;
  description
  "Count of packets with errors.
   Not counted in the pkts or captured.
   For example packets with CRC error.";
}

container testframe-stats {
  description
  "Statistics for testframe packets containing
   either sequence number, payload checksum,
   timestamp or any combination of these features.";
  leaf testframe-pkts {
    type yang:counter64;
    description
    "Total count of detected testframe packets.";
  }
  leaf sequence-errors {
    type yang:counter64;
    description
    "Total count of testframe packets with
     unexpected sequence number. After each sequence
     error the expected next sequence number is
     updated.";
  }
  leaf payload-errors {
    type yang:counter64;
    description
    "Total count of testframe packets with
     payload errors.";
  }
  container latency {
    description
    "Latency statistics.";
    leaf samples {
      type uint64;
      description
      "Total count of packets used for estimating
       the latency statistics. Ideally
       samples=../testframe-stats.";
    }
    leaf min {
      type uint64;
      units "nanoseconds";
      description
      "Minimum measured latency.";
    }
leaf max {
  type uint64;
  units "nanoseconds";
  description
    "Maximum measured latency.";
}

leaf average {
  type uint64;
  units "nanoseconds";
  description
    "The sum of all sampled latencies divided
     by the number of samples.";
}

leaf latest {
  type uint64;
  units "nanoseconds";
  description
    "Latency of the latest sample.";
}

grouping capture-data {
  description
    "Grouping with statistics and data
     of one or more captured frame.";
  container capture {
    if-feature "capture";
    description
      "Statistics and data of
       one or more captured frames.";
    list frame {
      key "sequence-number";
      description
        "Statistics and data of a captured frame.";
      leaf sequence-number {
        type uint64;
        description
          "Incremental counter of frames captured.";
      }
      leaf timestamp {
        type yang:date-and-time;
        description
          "Timestamp of the moment the frame was captured.";
      }
      leaf length {  

type uint32;
description
"Frame length. Ideally the data captured will be
depending on implementation limitations."
} 
leaf preceding-interframe-gap {
  type uint32;
  units "nanoseconds";
  description
  "Measured delay between the reception of the previous
  frame was completed and the reception of the current
  frame was started.";
}
leaf data {
  type string {
    pattern '([0-9A-F]{2})*';
  }
  description
  "Raw data of the captured frame.";
}

grouping filter-data {
  description
  "Grouping with a filter container specifying the filtering
  rules for processing only a specific subset of the
  frames.";
  container filter {
    if-feature "filter";
    presence "When present packets are
      filtered before analyzed according
to the filter type";
    description
    "Contains the filtering rules for processing only
a specific subset of the frames.";
    leaf type {
      type identityref {
        base ta:filter;
      }
      mandatory true;
      description
      "Type of the applied filter. External modules can
define alternative filter type identities.";
    }
  }
}
augment "/if:interfaces/if:interface" {
  description
  "Traffic analyzer augmentations of ietf-interfaces.";
  container traffic-analyzer {
    if-feature "ingress-direction";
    presence "Enables the traffic analyzer for ingress traffic.";
    description
    "Traffic analyzer for ingress direction.";
    uses filter-data;
    container state {
      config false;
      description
      "State data.";
      uses statistics-data;
      uses capture-data;
    }
  }
  container traffic-analyzer-egress {
    if-feature "egress-direction";
    presence "Enables the traffic analyzer for egress traffic.";
    description
    "Traffic analyzer for egress direction.";
    uses filter-data;
    container state {
      config false;
      description
      "State data.";
      uses statistics-data;
      uses capture-data;
    }
  }
}

augment "/if:interfaces/if:interface/ta:traffic-analyzer/ta:filter" {
  when "ta:type = 'ta:ethernet'";
  description
  "Ethernet frame specific filter type.";
  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.";
  }
}
7. IANA Considerations

This document registers three URIs and three YANG modules.

7.1. 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:


Registrant Contact: The IESG.

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

7.2. YANG Module Name Registration

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

name: ietf-traffic-generator
prefix: tg
reference: RFC XXXX

name: ietf-traffic-analyzer
prefix: ta
reference: RFC XXXX

8. Security Considerations

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

9. References

9.1. Normative References

9.2. Informative References


[A.1. Basic Test Program]

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

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

# Configure DUTs and enable traffic-analyzers
nett.node("dut0").edit(
  "create /interfaces/interface[name='e0'] -- type=ethernetCsmacd")
nett.node("dut0").edit(
  "create /interfaces/interface[name='e1'] -- type=ethernetCsmacd")
nett.node("dut0").edit(
  "create /flows/flow[id='t0'] -- match/in-port=e0 "
  "actions/action[order='0']/output-action/out-port=e1")
nett.node("tester1").edit(  
```

Appendix A. Examples

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

```
+-------------+          +------------+         +------------+
|             | e0    e0 |            | e1   e0 |            |
| tester0    TG|-------->|    dut0    |>------->|TA  tester1 |
|             |          |            |         |            |
+-------------+          +------------+         +------------+
```

A.1. Basic Test Program

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

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

# Configure DUTs and enable traffic-analyzers
nett.node("dut0").edit(
  "create /interfaces/interface[name='e0'] -- type=ethernetCsmacd")
nett.node("dut0").edit(
  "create /interfaces/interface[name='e1'] -- type=ethernetCsmacd")
nett.node("dut0").edit(
  "create /flows/flow[id='t0'] -- match/in-port=e0 "
  "actions/action[order='0']/output-action/out-port=e1")
nett.node("tester1").edit(  
```

Vassilev  Expires September 5, 2020  [Page 24]
"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")
...