Benchmarking Working Group
Internet-Draft
Intended status: Standards Track
Expires: 29 November 2026

V. Vassilev
Lightside Instruments AS

28 May 2026

A YANG Data Model for Network Tester Management
draft-ietf-bmwg-network-tester-cfg-11

Abstract

This document specifies a YANG data model for use in network interconnect testing setups that contains instances of 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 29 November 2026.

Table of Contents

1. Introduction

There is a need for a standard mechanism for specification of the transactions part of network tests and their implementation over standardized network management protocols. Such a mechanism should allow the control and monitoring of the data plane traffic in a transactional manner. This document defines a YANG [RFC7950] data model which is composed of two YANG modules for test traffic generator and analyzer functions.

1.1. Acronyms

DUT: Device Under Test

TA: Traffic Analyzer

TG: Traffic Generator

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 forwarding path. A network interconnect tester is an entity that can either generate test traffic, analyze test traffic, or both. Figure 1, borrowed from [RFC2544], represents the horseshoe test setup topology consisting of a single tester and a single Device Under Test (DUT) connected in a network interconnect loop.

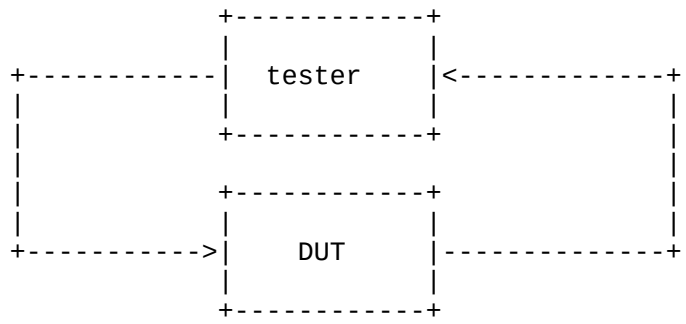

Figure 1: Test setup with tester that both sends the test traffic and receives it back

This document defines a YANG data 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 Network Configuration Protocol (NETCONF) [RFC6241].

1.3. Objectives

This section specifies the design objectives for the data model. It should:

* provide means to specify the generated traffic as streams of cyclic sequence of bursts with configurable frame size, frame data, interframe gap and interburst gap.

* have a mandatory single stream mode and optional multi stream mode.

* provide means for configuration of traffic streams with static frame data where frames with identical frame data are sent during the lifetime of the stream.

* provide means for configuration of traffic streams with dynamic frame data where frames contain fields with dynamic data like generation time and sequence number.

* allow third parties to augment the base modules with alternative dynamic fields of frame data extensions.

* provide means for realtime synchronization and orchestration of the generated streams.

* provide counters for received test traffic frames and octets.

* provide latency statistic in the case of test traffic with dynamic frame data that includes timestamp.

* provide sequence number errors in the case of test traffic with dynamic frame data that includes sequence number.

* provide means for capturing traffic frames data.

1.4. Data Model Overview

The proposed data model splits the design into two YANG modules - (1) Traffic Generator module (TG) and (2) Traffic Analyzer module (TA). The modules are implemented as augmentations of the "ietf-interfaces" [RFC8343] module adding configuration and state data that model the functionality of a network interconnect tester. The TA and TG modules concept is illustrated with the diagram depicted in Figure 2 with a tester with two interfaces (named "eth0" and "eth1") connected in a loop with single DUT.

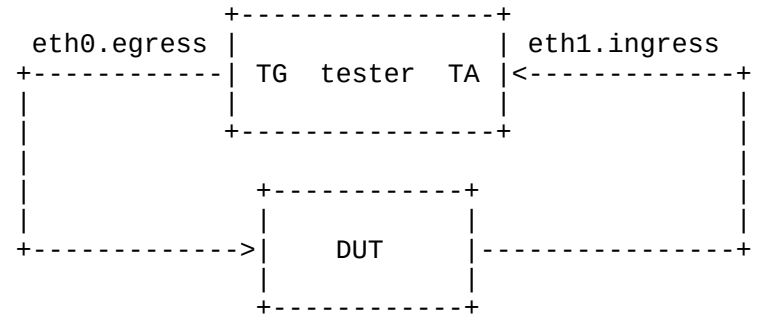

Figure 2: Test setup where TG and TA modules are colocated on the same entity

2. Using the Network Interconnect Tester Data Model

A basic example of how the data model can be used in transactional network test program 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. One of the examples demonstrates the use of the [RFC2544] defined test frame packet. The search algorithm logic that operates to control the trial configuration is outside the scope of this document.

3. Traffic Generator Module Tree Diagram

The meaning of the symbols in the YANG tree diagrams are defined in [RFC8340]

```
   module: ietf-traffic-generator

     augment /if:interfaces/if:interface:
       +--rw traffic-generator!
          +--rw (type)
          |  +--:(single-stream)
          |  |  +--rw testframe-type?     identityref
          |  |  +--rw frame-size          uint32
          |  |  +--rw frame-data?         binary
          |  |  +--rw gap      uint32
          |  |  +--rw burst-gap?     uint32
          |  |  +--rw frames-per-burst?   uint32
          |  |  +--rw modifiers
          |  |     +--rw modifier* [id]
          |  |        +--rw id             uint32
          |  |        +--rw action         identityref
          |  |        +--rw offset         uint32
          |  |        +--rw mask           binary
          |  |        +--rw repetitions    uint32
          |  +--:(multi-stream)
          |     +--rw streams
          |        +--rw stream* [id]
          |           +--rw id                   uint32
          |           +--rw testframe-type?      identityref
          |           +--rw frame-size           uint32
          |           +--rw frame-data?          binary
          |           +--rw gap                  uint32
          |           +--rw burst-gap?           uint32
          |           +--rw frames-per-burst?    uint32
          |           +--rw frames-per-stream    uint32
          |           +--rw stream-gap           uint32
          |           +--rw modifiers
          |              +--rw modifier* [id]
          |                 +--rw id             uint32
          |                 +--rw action         identityref
          |                 +--rw offset         uint32
          |                 +--rw mask           binary
          |                 +--rw repetitions    uint32
          +--rw realtime-epoch?
          |        yang:date-and-time {realtime-epoch}?
          +--rw start-delay?              uint64
          +--rw total-frames?             uint64
```

4. Traffic Analyzer Module Tree Diagram

```
module: ietf-traffic-analyzer

  augment /if:interfaces/if:interface:
    +--rw traffic-analyzer!
       +--rw testframe-filter! {testframe-filter}?
       |  +--rw type      identityref
       |  +--rw mask?     binary
       |  +--rw data?     binary
       |  +--rw offset    uint32
       +--rw capture! {capture}?
       |  +--rw start-trigger!
       |  |  +--rw (start-trigger)
       |  |     +--:(frame-index)
       |  |     |  +--rw frame-index?       uint64
       |  |     +--:(testframe-index)
       |  |        +--rw testframe-index?   uint64
       |  +--rw stop-trigger!
       |     +--rw (stop-trigger)
       |        +--:(when-full)
       |           +--rw when-full?   empty
       +--ro state
          +--ro pkts?              yang:counter64
          +--ro octets?            yang:counter64
          +--ro idle-octets?       yang:counter64 {idle-octets-counter}?
          +--ro errors?            yang:counter64
          +--ro testframe-stats
          |  +--ro 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 last-sequence-error
          |     +--ro timestamp?   yang:date-and-time
          |     +--ro expected?    uint64
          |     +--ro received?    uint64
          +--ro capture {capture}?
             +--ro frame* [sequence-number]
                +--ro sequence-number    uint64
                +--ro timestamp?         yang:date-and-time
                +--ro length?            uint32
                +--ro data?              binary
```

5. Traffic Generator Module YANG

This module imports modules defined in [RFC8343] and [RFC9911].

<CODE BEGINS> file "ietf-traffic-generator@2026-05-28.yang"

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

  import ietf-interfaces {
    prefix if;
    reference
      "RFC 8343: A YANG Data Model for Interface Management";
  }
  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 9911: Common YANG Data Types";
  }

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

     Author:   Vladimir Vassilev
               <mailto:vladimir@lightside-instruments.com>";
  description
    "This module contains a collection of YANG definitions for
     management of the traffic generator functionality part
     of network interconnect testers.

     Copyright (c) 2026 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 Revised BSD License set
     forth in Section 4.c of the IETF Trust's Legal Provisions
     Relating to IETF Documents
     (https://trustee.ietf.org/license-info).

     All revisions of IETF and IANA published modules can be found
     at the YANG Parameters registry group
     (https://www.iana.org/assignments/yang-parameters).
```

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

    revision 2026-05-28 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: A YANG Data Model for
                   Network Tester Management";
    }

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

    feature realtime-epoch {
      description
        "The device can generate traffic precisely
         at configured realtime epoch.";
    }

    identity testframe-type {
      description
        "Base identity for all test frame types.";
    }

    identity static {
      base testframe-type;
      description
        "Indicates static test frame.
         The frame data and size are constant.";
    }

    identity dynamic {
      base testframe-type;
      description
        "Base identity for dynamic test frame.

         When used itself, it identifies dynamic test frame
         where the last 18 octets of the payload contain
         incrementing sequence number field (8 octets)
         followed by timestamp field in the
         IEEE 1588-2008 format (10 octets). If frame data is defined
         for the last 18 octets of the payload, it will be ignored
         and overwritten with dynamic data according to this
         specification.";
    }
```

```
    identity modifier-action-type {
      description
        "Base identity for all modifier action types.";
    }

    identity increment {
      base modifier-action-type;
      description
        "Indicates increment modifier action.";
    }

    identity decrement {
      base modifier-action-type;
      description
        "Indicates decrement modifier action.";
    }

    identity random {
      base modifier-action-type;
      description
        "Indicates random modifier action.";
    }

    grouping common-data {
      description
        "Common configuration data.";
      leaf realtime-epoch {
        if-feature "realtime-epoch";
        type yang:date-and-time;
        description
          "If this leaf is present the stream generation will start
           at the specified realtime epoch.";
      }
      leaf start-delay {
        type uint64;
        units "octets";
        description
          "If this leaf is present, the traffic generation will start
           after the specified number of idle octets have elapsed.";
      }
      leaf total-frames {
        type uint64;
        description
          "If this leaf is present the traffic generation will stop
           after the specified number of frames are generated.";
      }
    }
```

```
    grouping burst-data {
      description
        "Generated traffic burst parameters.";
      leaf testframe-type {
        type identityref {
          base nttg:testframe-type;
        }
        default "nttg:static";
        description
          "In case of dynamic test frames this leaf
           can specify a derived dynamic test frame identity.
           For static test frames there is no practical use
           of derived identities.";
      }
      leaf frame-size {
        type uint32;
        units "octets";
        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),
           * Frame Check Sequence (FCS) (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.";
      }
      leaf frame-data {
        type binary;
        description
          "The raw frame data.
           The length of the specified data can be
           shorter, then the '../frame-size' value
           specifying only the header or the header and the
           payload with or without the 4-byte FCS data
           in the case of a Ethernet frame. If the frame data string
           specified is longer, then '../frame-size' the excess
```

```
            of the specified frame data is truncated.

            In multi-stream mode, if 'frame-data' leaf for one of the
            streams is not specified, it is the value specified for
            the closest stream that precedes it that is used.";
        }
        leaf gap {
          type uint32;
          units "octets";
          mandatory true;
          description
            "Length of the period between generated frames.
             For example, for Ethernet interfaces the following
             definition applies:

             The gap between transmission of frames is the sum
             of the minimum interframe gap (IFG) as defined in
             IEEE 802.3 plus any additional idle octets plus
             the preamble (7 octets) and the start of frame
             delimiter (1 octet). 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 gap. Thus, the minimum
             value for standard Ethernet transmission should be considered
             20 octets.";
        }
        leaf burst-gap {
          type uint32;
          units "octets";
          description
            "Is similar to the gap but takes place between
             any two bursts of the stream.";
        }
        leaf frames-per-burst {
          type uint32;
          description
            "Indicates number of frames contained in a burst.";
        }
      }

      grouping modifier-data {
        description
          "Frame data modification parameters.";
        container modifiers {
```

```
        description
          "Container holding the configured modifiers list.";
        list modifier {
          key "id";
          description
            "Each modifier specifies the action to be performed
             on data at a certain offset.";
          leaf id {
            type uint32;
            description
              "Number specifying the identifier of the modifier.";
          }
          leaf action {
            type identityref {
              base nttg:modifier-action-type;
            }
            mandatory true;
            description
              "Modifier action type.";
          }
          leaf offset {
            type uint32;
            units "octets";
            mandatory true;
            description
              "Offset of the modified data of the frame.";
          }
          leaf mask {
            type binary;
            mandatory true;
            description
              "Bit mask of the actual bits affected by the modifier.";
          }
          leaf repetitions {
            type uint32;
            mandatory true;
            description
              "Count of the packets that will repeat the data before
               the modifier makes the next update.";
          }
        }
      }
    }

    grouping multi-stream-data {
      description
        "Multi stream traffic generation parameters.";
      container streams {
```

```
        description
          "A container holding the stream list.";
        list stream {
          key "id";
          description
            "Each stream repeats a burst until frames-per-stream
             count is reached followed by stream-gap delay.";
          leaf id {
            type uint32;
            description
              "Number specifying the order of the stream.";
          }
          uses 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 stream-gap {
            type uint32;
            units "octets";
            mandatory true;
            description
              "Idle period after the last frame of the last burst.";
          }
          uses modifier-data;
        }
      }
    }

    augment "/if:interfaces/if:interface" {
      description
        "Traffic generator augmentations of 'ietf-interfaces'.";
      container traffic-generator {
        presence "Enables the traffic generator.";
        description
          "Traffic generator data.";
        choice type {
          mandatory true;
          description
            "Choice of the type of the generator:
             Single or multi stream.";
          case single-stream {
            uses burst-data;
            uses modifier-data;
          }
```

```
          case multi-stream {
            uses multi-stream-data;
          }
        }
        uses common-data;
      }
    }
  }
```

<CODE ENDS>

6. Traffic Analyzer Module YANG

This module imports modules defined in [RFC8343] and [RFC9911].

<CODE BEGINS> file "ietf-traffic-analyzer@2026-05-28.yang"

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

  import ietf-interfaces {
    prefix if;
    reference
      "RFC 8343: A YANG Data Model for Interface Management";
  }
  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 9911: Common YANG Data Types";
  }

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

     Author:   Vladimir Vassilev
               <mailto:vladimir@lightside-instruments.com>";
  description
    "This module contains a collection of YANG definitions for
     management of the traffic analyzer functionality part
     of network interconnect testers.

     Copyright (c) 2026 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 Revised BSD License set
       forth in Section 4.c of the IETF Trust's Legal Provisions
       Relating to IETF Documents
       (https://trustee.ietf.org/license-info).

       All revisions of IETF and IANA published modules can be found
       at the YANG Parameters registry group
       (https://www.iana.org/assignments/yang-parameters).

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

    revision 2026-05-28 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: A YANG Data Model for
                   Network Tester Management";
    }

    feature testframe-filter {
      description
        "Indicates that the device implements
         a filter that can specify a subset of packets to be
         analyzed as testframes.";
    }

    feature idle-octets-counter {
      description
        "Indicates that the device implements
         idle-octets counter that accumulates the time
         the link is not utilized. The minimum required
         idle gaps are not counted as idle octets.";
    }

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

    identity testframe-filter {
      description
        "Base testframe-filter identity.";
    }
```

```
    identity bit-field-match {
      base ntta:testframe-filter;
      description
        "Bit field matching filter. Frames that do not have
         equal values for all the bits of the specified data
         at the offsets specified with bits set in the mask
         are not processed as test frames.";
    }

    grouping statistics-data {
      description
        "Analyzer statistics.";
      leaf pkts {
        type yang:counter64;
        description
          "Total number of packets analyzed.";
      }
      leaf octets {
        type yang:counter64;
        description
          "This counter is identical with the in-octets/out-octets
           counters defined in RFC8343 except that it counts the
           octets since the analyzer was created.";
      }
      leaf idle-octets {
        if-feature "idle-octets-counter";
        type yang:counter64;
        description
          "Total accumulated period with no frame transmission
           taking place measured in octets at the current link
           speed. This counter does not increment only when the
           interface is operating at 100% of its capacity.

           Octets not counted in '../octets' but not idle are,
           for example, layer 1 framing octets - for Ethernet interfaces
           7+1 preamble octets per packet as well as the 12-octets
           minimum gap.";
      }
      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 received testframes containing
```

```
           either sequence number, payload checksum,
           timestamp or any combination of these features.";
        leaf pkts {
          type yang:counter64;
          description
            "Total count of detected testframes.";
        }
        leaf sequence-errors {
          type yang:counter64;
          description
            "Total count of testframes 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 testframes with
             payload errors.";
        }
        container latency {
          description
            "Latency statistics.";
          leaf samples {
            type uint64;
            description
              "Total count of testframes used for estimating
               the latency statistics. When every testframe is
               used, samples='../pkts'.";
          }
          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.";
          }
        }
        container last-sequence-error {
          description
            "Last sequence error state data.";
          leaf timestamp {
            type yang:date-and-time;
            description
              "Timestamp of the moment a testframe with
               unexpected sequence number was received.";
          }
          leaf expected {
            type uint64;
            description
              "Expected sequence number.";
          }
          leaf received {
            type uint64;
            description
              "Received sequence number.";
          }
        }
      }
    }

    grouping capture-config-data {
      description
        "Grouping with a capture configuration container.";
      container capture {
        if-feature "capture";
        presence "Enables the capture functionality";
        description
          "Contains capture parameters.";
        container start-trigger {
          presence "Enables trigger condition that starts
                    the capture process.";
          description
            "Configures when the capture start is triggered.";
          choice start-trigger {
            mandatory true;
            description
```

```
              "Choice of start-trigger condition.";
            case frame-index {
              description
                "Start capturing frames at the specified frame index.";
              leaf frame-index {
                type uint64;
                description
                  "First captured frame index.";
              }
            }
            case testframe-index {
              description
                "Start capturing frames at the specified
                 testframe index.";
              leaf testframe-index {
                type uint64;
                description
                  "First captured testframe index.";
              }
            }
          }
        }
        container stop-trigger {
          presence "Enables trigger condition that stops
                    the capture process.";
          description
            "Configures when the capture is stopped.";
          choice stop-trigger {
            mandatory true;
            description
              "Choice of stop-trigger condition.";
            case when-full {
              description
                "Stops capturing when the implementation can not store
                 more frames.";
              leaf when-full {
                type empty;
                description
                  "When present in configuration capture stops when
                   the capture buffer is full.";
              }
            }
          }
        }
      }
    }

    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
               of the same length but can be shorter
               depending on implementation limitations.";
          }
          leaf data {
            type binary;
            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 testframe-filter {
        if-feature "testframe-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 ntta:testframe-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 {
        presence "Enables the traffic analyzer.";
        description
          "Traffic analyzer for ingress direction.";
        uses filter-data;
        uses capture-config-data;
        container state {
          config false;
          description
            "State data.";
          uses statistics-data;
          uses capture-data;
        }
      }
    }

    augment "/if:interfaces/if:interface/ntta:traffic-analyzer/"
          + "ntta:testframe-filter" {
      when "derived-from-or-self(ntta:type, 'ntta:bit-field-match')";
      description
        "Logical AND of masked bit fields.";
      leaf mask {
        type binary;
        description
          "Specifies bit field mask for comparison.
           Non-masked bit fields are ignored.";
      }
      leaf data {
        type binary;
        description
```

```
          "Specify data to be matched according to the specified mask.";
      }
      leaf offset {
        type uint32;
        mandatory true;
        description
          "Offset in the frame.";
      }
    }
  }
```

<CODE ENDS>

7. IANA Considerations

This document registers two URIs and two YANG modules.

7.1. URI Registration

IANA is requested to register the following URI in the "ns" registry within the "IETF XML Registry" group [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-traffic-generator
Registrant Contact: The IESG
XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-traffic-analyzer
Registrant Contact: The IESG
XML: N/A; the requested URI is an XML namespace.

7.2. YANG Module Name Registration

IANA is requested to register the following YANG module in the "YANG Module Names" registry [RFC6020] within the "YANG Parameters" registry group.

Name: ietf-traffic-generator
Maintained by IANA? Y/N
Namespace: urn:ietf:params:xml:ns:yang:ietf-traffic-generator
Prefix: nttg
Reference: RFC XXXX

Name: ietf-traffic-analyzer
Maintained by IANA? Y/N
Namespace: urn:ietf:params:xml:ns:yang:ietf-traffic-analyzer
Prefix: ntta
Reference: RFC XXXX

8. Operational Considerations

The configuration nodes defined in the ietf-traffic-generator module allow deterministic specification of traffic as a sequences of frames each followed by a gap. Unless 'realtime-epoch' leaf is present the interface starts generation upon the creation or any change of the configuration under '/if:interfaces/if:interface/nttg:traffic-generator'. Multiple traffic generators on different interfaces can be synchronized with the 'start-delay' leaf. Traffic generation continues until as many frames as specified by the 'total-frames' leaf when it is present are generated or until the traffic generator generator configuration under the 'traffic-generator' container is deleted causing the traffic generation to stop or changed causing the traffic generation to stop then restart.

The presence of '/if:interfaces/if:interface/ntta:traffic-analyzer' container defined in the ietf-traffic-analyzer module enables the traffic analyzer functionality on an interface. The traffic analyzer continues its operation until the 'traffic-analyzer' container is deleted or changed causing the traffic analyzer to stop then restart discarding its state including statistics and captured frames.

9. Security Considerations

This section is modeled after the template described in Section 3.7.1 of [RFC9907]. The YANG modules "ietf-traffic-generator" and "ietf-traffic-analyzer" define a data model that is designed to be accessed via YANG-based management protocols, such as the Network Configuration Protocol (NETCONF) [RFC6241]. These YANG-based management protocols (1) have to use a secure transport layer (e.g., Secure Shell (SSH) [RFC4252], TLS [RFC8446], and QUIC [RFC9000]) and (2) have to use mutual authentication. The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF users to a preconfigured subset of all available NETCONF protocol operations and content.

9.1. ietf-traffic-generator.yang

The ietf-traffic-generator YANG module controls a stateless traffic generator which is intended to be used for testing and verification purposes but can be used for malicious purposes like generating network traffic part of a Denial-of-Service (DoS) attack. This should be taken into consideration when granting write access to the following container and descendant data nodes:

* /if:interfaces/if:interface/nttg:traffic-generator

9.2. ietf-traffic-analyzer.yang

The ietf-traffic-analyzer YANG module controls a traffic analyzer which is designed for use in testing and verification but can be used for reading information contained in packets sent and received on any of the interfaces on systems that implement the capture feature. This should be taken into consideration when granting read access to the following container and descendant data nodes:

* /if:interfaces/if:interface/ntta:traffic-analyzer/ntta:capture

10. References

10.1. Normative References

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

[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>.

[RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <https://www.rfc-editor.org/info/rfc7224>.

[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>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <https://www.rfc-editor.org/info/rfc8343>.

[RFC9911] Schnwlder, J., Ed., "Common YANG Data Types", RFC 9911, DOI 10.17487/RFC9911, December 2025, <https://www.rfc-editor.org/info/rfc9911>.

10.2. Informative References

[IEEE1588] IEEE, "IEEE 1588-2008", 2008.

[IEEE802.3-2014]
IEEE WG802.3 - Ethernet Working Group, "IEEE 802.3-2014", 2014.

[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>.

[RFC4252] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, DOI 10.17487/RFC4252, January 2006, <https://www.rfc-editor.org/info/rfc4252>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[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>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

[RFC9907] Bierman, A., Boucadair, M., Ed., and Q. Wu, "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 9907, DOI 10.17487/RFC9907, March 2026, <https://www.rfc-editor.org/info/rfc9907>.

Appendix A. Examples

The topology depicted in Figure 3 is used for the examples in this appendix:

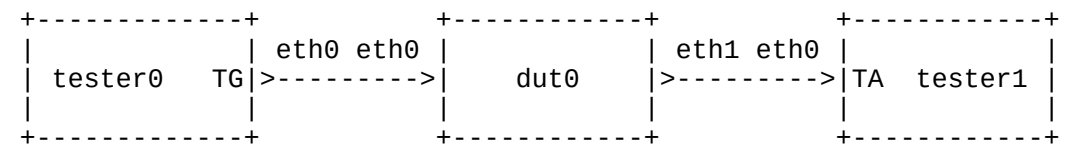

Figure 3: Test setup where TG and TA modules are on distinct entities that are not colocated

A.1. Basic Test Program

This pseudo code program orchestrates a network test and shows how the model can be used:

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

# Configure DUTs and enable traffic-analyzers
net.node("dut0").edit( \
  "create /interfaces/interface[name='eth0'] -- type=ethernetCsmacd")
net.node("dut0").edit(
  "create /interfaces/interface[name='eth1'] -- type=ethernetCsmacd")
net.node("dut0").edit(
  "create /flows/flow[id='t0'] -- match/in-port=eth0 "
  "actions/action[order='0']/output-action/out-port=eth1")

net.node("tester1").edit(
  "create /interfaces/interface[name='eth0']/traffic-analyzer")
net.commit()

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

# Start traffic
net.node("tester0).edit(
  "create /interfaces/interface[name='eth0']/traffic-generator -- "
  "frame-size=64 gap=20")

net.commit()

time.sleep(60)

# Stop traffic
net.node("tester1").edit("delete /interfaces/interface[name='eth0']/"
                         "traffic-generator")
net.commit()
```

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

# Report
sent_pkts=delta("tester0",before,after,
  "/interfaces/interface[name='eth0']/statistics/out-unicast-pkts")

received_pkts=delta("tester1",before,after,
  "/interfaces/interface[name='eth0']/statistics/in-unicast-pkts")

latency_max=absolute(after,
  "/interfaces/interface[name='eth0']/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 Test Frames

Appendix C.2.6.4 of [RFC2544] specifies a detailed format. The 'frame-data' leaf provides an operator with full control over the generated frames payload. Notice that the encoding used to specify the value of the 'frame-data' leaf in this pseudocode example is not the encoding NETCONF would use on the wire which is Base64 but instead it is specified with human-readable format identical to the 'hex-string' type defined in [RFC9911].

```
...
net.node("tester1").edit(
  "merge /interfaces/interface[name='eth0']/"
  "traffic-generator -- frame-data="
  "6c:a9:6f:00:00:02:6c:a9:6f:00:00:01:08:00:45:00:"
  "00:2e:d4:a5:00:00:0a:11:58:16:c0:00:02:01:C0:00:"
  "02:02:c0:20:00:07:00:1a:00:00:01:02:03:04:05:06:"
  "07:08:09:0a:0b:0c:0d:0e:0f:10:11:12")
...
```

The resulting configuration serialized as XML is shown below in Appendix A.4

A.3. Generating Frames with Dynamic Data Fields

Adding a modifier functionality to the single stream traffic generator configuration. In this example, the highest 16 bits of the source MAC address of the generated Ethernet frame will be incremented.

```
...
net.node("tester1").edit(
  "merge /interfaces/interface[name='eth0']/"
  "traffic-generator/modifiers/modifier[id='0'] -- "
  "offset=6 mask=ff:ff action=increment repetitions=1")
...
```

A.4. Example: NETCONF <get-config> Reply

This section gives an example of a reply to the NETCONF <get-config> request for <running> for a device that implements the ietf-traffic-generator module and its configuration was modified according to the examples above.

```
<rpc-reply
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
    message-id="101">
  <data>
    <interfaces
        xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
        xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type"
        xmlns:nttg="urn:ietf:params:xml:ns:yang:ietf-traffic-generator">
      <interface>
        <name>eth0</name>
        <type>ianaift:ethernetCsmacd</type>
        <nttg:traffic-generator>
          <frame-size>64</frame-size>
          <frame-data>
              bKlvAAACbKlvAAABCABFAAAu1KUAAAoRWBbAAAIB
              wAACAsAgAAcAGgAAAQIDBAUGBwgJCgsMDQ4PEBES
          </frame-data>
          <gap>20</gap>
        </nttg:traffic-generator>
      </interface>
    </interfaces>
  </data>
</rpc-reply>
```

Author's Address

Vladimir Vassilev
Lightside Instruments AS
Email: vladimir@lightside-instruments.com