

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 7 January 2026

V. Vassilev
Lightside Instruments AS
6 July 2025

A YANG Data Model for Network Tester Management
draft-ietf-bmwg-network-tester-cfg-06

Abstract

This document introduces new YANG model for use in network interconnect testing containing modules 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 7 January 2026.

Copyright Notice

Copyright (c) 2025 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology	2
1.1.1. Definitions and Acronyms	2
1.1.2. Tree Diagram	3
1.2. Problem Statement	3
1.3. Objectives	3
1.4. Solution	4
2. Using the network interconnect tester model	5
3. Traffic Generator Module Tree Diagram	5
4. Traffic Analyzer Module Tree Diagram	6
5. Traffic Generator Module YANG	7
6. Traffic Analyzer Module YANG	14
7. IANA Considerations	22
7.1. URI Registration	22
7.2. YANG Module Name Registration	22
8. Security Considerations	23
8.1. ietf-traffic-generator.yang	23
8.2. ietf-traffic-analyzer.yang	23
9. References	24
9.1. Normative References	24
9.2. Informative References	24
Appendix A. Examples	25
A.1. Basic Test Program	25
A.2. Generating RFC2544 Testframes	26
A.3. Generating frames with dynamic data fields	27
Author's Address	27

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 two YANG modules for test traffic generator and analyzer.

The YANG modules in this document conform to the Network Management Datastore Architecture (NMDA) defined in RFC 8342.

1.1. Terminology

1.1.1. Definitions and Acronyms

DUT: Device Under Test

TA: Traffic Analyzer

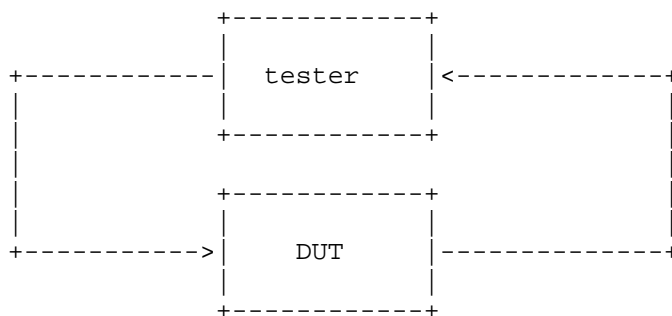
TG: Traffic Generator

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.



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. Objectives

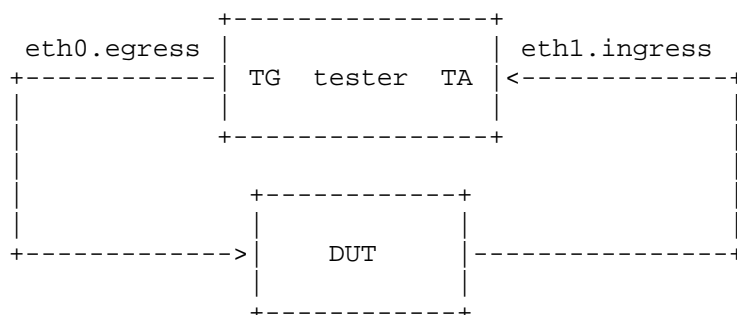
This section specifies the design objectives for the model. It should:

- * 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 module 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. Solution

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

```

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?         string
      | | +--rw interframe-gap      uint32
      | | +--rw interburst-gap?     uint32
      | | +--rw frames-per-burst?   uint32
      | | +--rw modifiers
      | |   +--rw modifier* [id]
      | |   |
      | |   | +--rw id              uint32
      | |   | +--rw action          identityref
      | |   | +--rw offset          uint32
      | |   | +--rw mask            string
      | |   | +--rw repetitions     uint32
      | |
      | | +--:(multi-stream)
      | |   +--rw streams
      | |   |
      | |   | +--rw stream* [id]
      | |   | |
      | |   | | +--rw id              uint32
      | |   | | +--rw testframe-type? identityref
      | |   | | +--rw frame-size      uint32
      | |   | | +--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 modifiers
      | |   | |   +--rw modifier* [id]
      | |   | |   |
      | |   | |   | +--rw id          uint32
      | |   | |   | +--rw action      identityref
      | |   | |   | +--rw offset      uint32
      | |   | |   | +--rw mask        string
      | |   | |   | +--rw repetitions uint32
      | |
      | +--rw realtime-epoch?
      | |   yang:date-and-time {realtime-epoch}?
      +--rw total-frames?

```

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?    string
      |   +--rw data?    string
      +--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 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 last-sequence-error
      |   |   |   +--ro timestamp?    yang:date-and-time
      |   |   |   +--ro expected?     yang:counter64
      |   |   |   +--ro received?     yang:counter64
      +--ro capture {capture}?
      |   +--ro frame* [sequence-number]
      |   |   +--ro sequence-number    uint64
      |   |   +--ro timestamp?        yang:date-and-time
      |   |   +--ro length?            uint32
      |   |   +--ro data?              string

```

5. Traffic Generator Module YANG

<CODE BEGINS> file "ietf-traffic-generator@2024-10-21.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 6991: Common YANG Data Types";
  }

  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@lightside-instruments.com>";
  description
    "This module contains a collection of YANG definitions for
    description and management of network interconnect testers.

    Copyright (c) 2024 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).

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

  revision 2024-10-21 {
    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 testframe types.";
}

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

identity dynamic {
  base testframe-type;
  description
    "Identity to be used as base for dynamic
    testframe type identities defined
    in external modules.

    When used itself it identifies dynamic testframe
    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
```

```
    "Identity for increment modifier action.";
}

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

identity random {
    base modifier-action-type;
    description
        "Identity for 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 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 testframes this leaf specifies
             the dynamic testframe identity.";
    }
    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."
}
leaf frame-data {
  type string {
    pattern '([0-9A-F]{2})*';
  }
  description
    "The raw frame data specified as hexadecimal string.
    The string length of the specified data can be
    shorter then the 2*../frame-size value
    specifying only the header or the header and the
    payload with or without the 4 byte CRC Checksum
    in the case of a Ethernet frame. If the frame data string
    specified is longer then ../frame-size*2 the excess
    of the specified frame data is trunkated.

    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 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 modifier-data {
  description
    "Modifier parameters.";
  container modifiers {
    description
      "Container holding the configured modifiers list.";
    list modifier {
      key "id";
      description
        "Each modifier specifies action to be performed
        on data at 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
          "In case of dynamic testframes this leaf specifies
          the dynamic testframe identity.";
      }
    }
  }
}
```

```
    leaf offset {
      type uint32;
      mandatory true;
      description
        "Offset in octets of the modified data of the frame.";
    }
    leaf mask {
      type string {
        pattern '([0-9A-F]{2})*';
      }
      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
      "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 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.";
    }
    uses modifier-data;
}
}
}
```

```
augment "/if:interfaces/if:interface" {
    description
        "Traffic generator augmentations of ietf-interfaces.";
    container traffic-generator {
        description
            "Traffic generator configuration data.";
        choice type {
            description
                "Choice of the type of the data model 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

<CODE BEGINS> file "ietf-traffic-analyzer@2024-10-21.yang"

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

    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/>
  WG List:    <mailto:bmwg@ietf.org>

  Editor:     Vladimir 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) 2024 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).

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

revision 2024-10-21 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: A YANG Data Model for
    Network Tester Management";
}

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

```
feature idle-octets-counter {
  description
    "This feature 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
    "This feature indicates that the device implements
    packet capture functionality.";
}

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

identity bit-field-match {
  base ntt:ntta:testframe-filter;
  description
    "Bit field matching filter according to template data and mask.";
}

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. Octets not counted in ../octets but not idle are
      for example layer 1 framing octets - for Ethernet interfaces
      7+1 preamble octets per packet.";
```



```
}
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 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.";
    }
  }
  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 yang:counter64;
      description
        "Expected sequence number.";
    }
    leaf received {
      type yang:counter64;
      description
        "Received sequence number.";
    }
  }
}

grouping capture-config-data {
  description
    "Grouping with a capture configuration container.";
  container capture {
    if-feature "capture";
```

```
description
  "Contains capture parameters.";

container start-trigger {
  description
    "Configures when the capture start is triggered.";
  choice start-trigger {
    description
      "If none of the cases in this choice are configured the
       capture process starts from the first frame received.";
    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
          "Starts capture at specified testframe index.";
      }
    }
  }
}

container stop-trigger {
  description
    "Configures when the capture is stopped.";
  choice stop-trigger {
    description
      "If none of the cases in this choice are configured the
       captured frames are always the last frames received for
       as many frames the implementation can buffer.";
    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 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 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 ntt: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 string {
      pattern '([0-9A-F]{2})*';
    }
    description
      "Specifies bit field mask for comparison.
       Non-masked bit fields are ignored.";
  }
  leaf data {
    type string {
      pattern '([0-9A-F]{2})*';
    }
    description
      "Specify data to be matched according to the specified mask.";
  }
}
```

<CODE ENDS>

7. IANA Considerations

This document registers two URIs and two YANG modules.

7.1. URI Registration

This document registers two 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

Registrant Contact: The IESG.

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

7.2. YANG Module Name Registration

This document registers two 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: nttg
reference: RFC XXXX
```

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

8. Security Considerations

The YANG modules defined in this document are designed to be accessed via the NETCONF protocol RFC 6241 [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory to implement secure transport is SSH RFC 6242 [RFC6242]. The NETCONF access control model RFC 6536 [RFC6536] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

There are a number of data nodes defined in this YANG module which are writable/creatable/deletable (i.e. config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g. edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

8.1. 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
```

8.2. 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

9. References

9.1. Normative References

- [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>>.
- [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>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<https://www.rfc-editor.org/info/rfc6536>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [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>>.
- [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>>.

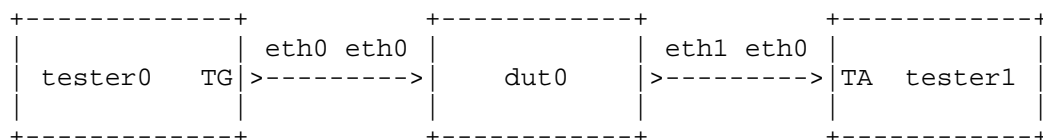
9.2. Informative References

- [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>>.
- [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 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 interframe-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 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='eth0']/"
  "traffic-generator -- frame-data="
  "6CA96F0000026CA96F00000108004500"
  "002ED4A500000A115816C0000201C000"
  "0202C0200007001A0000010203040506"
  "0708090A0B0C0D0E0F101112")
...
```

A.3. Generating frames with dynamic data fields

Adding modifier to single stream traffic generator configuration. In this example the 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=FFFF action=increment repetitions=1")
...
```

Author's Address

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