IPPM WG
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2015

R. Civil
Ciena Corporation
A. Morton
AT&T Labs
L. Zheng
Huawei Technologies
R. Rahman
Cisco Systems
M. Jethanandani
Ciena Corporation
K. Pentikousis, Ed.
EICT
March 9, 2015

Two-Way Active Measurement Protocol (TWAMP) Data Model
draft-cmzrjp-ippm-twamp-yang-00

Abstract

This document specifies a data model for client and server implementations of the Two-Way Active Measurement Protocol (TWAMP). We define the TWAMP data model through Unified Modeling Language (UML) class diagrams and formally specify it using YANG.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 10, 2015.

Copyright Notice

Table of Contents

1. Introduction

The Two-Way Active Measurement Protocol (TWAMP) [RFC5357] can be used to measure network performance parameters such as latency, bandwidth, and packet loss by sending probe packets and monitoring their experience in the network. To date, TWAMP implementations do not come with a standard management framework and, as such, configuration depends on the various proprietary mechanisms developed by the corresponding TWAMP vendor.

1.1. Motivation

For large, virtualized, and dynamically instantiated infrastructures where network functions are placed according to orchestration algorithms as discussed in [I-D.unify-nfvrg-challenges], proprietary mechanisms for managing TWAMP measurements have severe limitations. For current deployments, the lack of standardized programmable data model limits the flexibility to dynamically instantiate TWAMP-based measurement across equipment from different vendors.

We note that earlier efforts to define, for example, a TWAMP Management Information Base (MIB) [I-D.elteto-ippm-twamp-mib] did not advance. As we move forward, two major trends call for revisiting the standardization on TWAMP management aspects. First, we expect that in the coming years large-scale and multi-vendor TWAMP deployments will become the norm. From an operations perspective, dealing with several vendor-specific TWAMP configuration mechanisms is simply unsustainable in this context. Second, the increasingly software-defined and virtualized nature of network infrastructures, based on dynamic service chains [NSC] and programmable control and management planes [RFC7426] requires a well-defined data model for TWAMP implementations. This document defines such a TWAMP data model and specifies it formally using the YANG data modeling language [RFC6020].

1.2. Terminology

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

1.3. Document Organization

The rest of this document is organized as follows. Section 2 presents the scope and applicability of this document. Section 3 provides a high-level overview of the TWAMP data model. Section 4 details the configuration parameters of the data model and Section 5 specifies the YANG module. Section 6 lists illustrative examples

which conform to the YANG module specified in this document. Appendix A elaborates these examples further.

2. Scope, Model, and Applicability

The purpose of this document is the specification of vendor-independent data model for TWAMP implementations.

Figure 1 illustrates a redrawn version of the TWAMP logical model found in [RFC5357], Section 1.2. The figure is annotated with pointers to the UML diagrams provided in this document and associated with the data model of the four logical entities in a TWAMP deployment, namely the TWAMP Control-Client, Server, Session-Sender and Session-Reflector. As per [RFC5357], unlabeled links in the figure are unspecified and may be proprietary protocols.

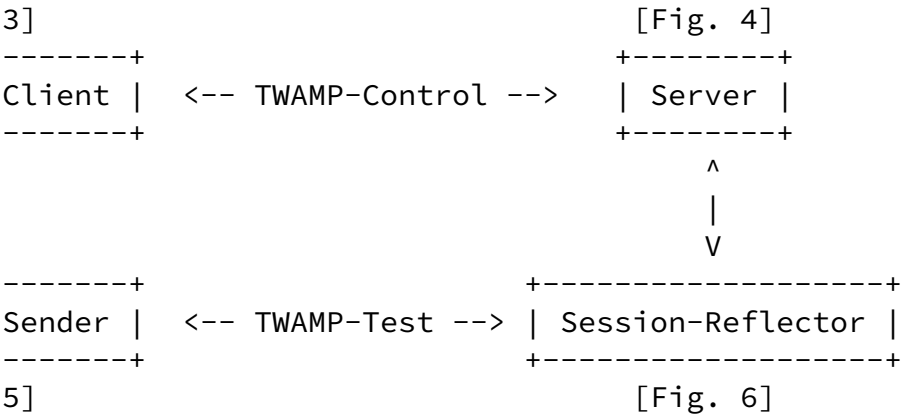


Figure 1: Annotated TWAMP logical model

As discussed in [RFC5357], a TWAMP implementation may follow a simplified logical model, in which the same node acts both as the Control-Client and Session-Sender, while another node acts at the same time as the TWAMP Server and Session-Reflector. Figure 2 illustrates this simplified logical model and indicates the interaction between the TWAMP configuration client and server using, for instance, NETCONF [RFC6241] or RESTCONF [I-D.ietf-netconf-restconf]. Note, however, that the specific protocol used to communicate the TWAMP configuration parameters specified herein is outside the scope of this document.

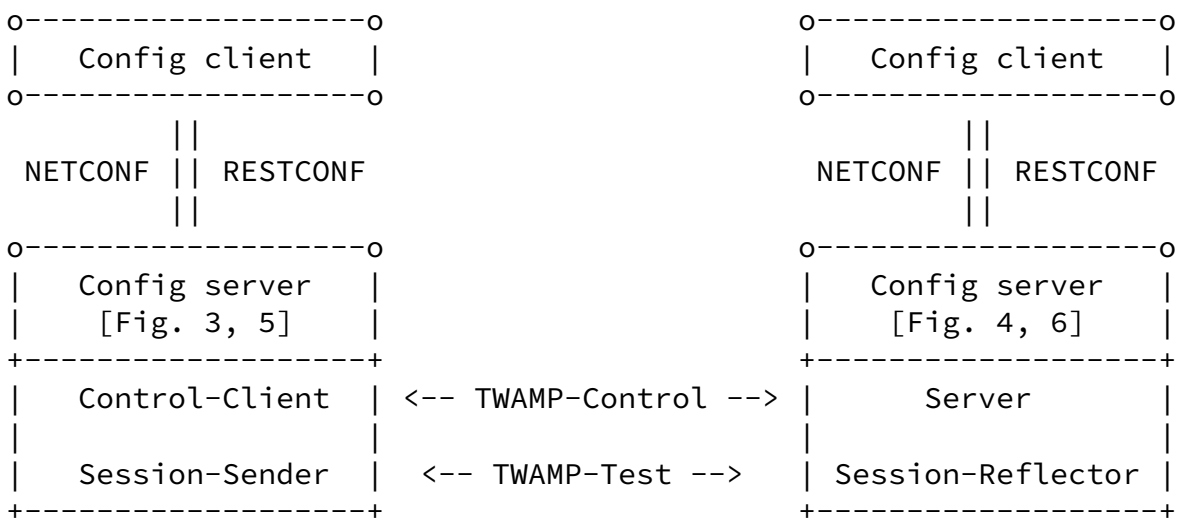


Figure 2: Simplified TWAMP model and protocols

3. Data Model Overview

A TWAMP data model includes four categories of configuration items. Global configuration items relate to parameters that are set on a per device level. For example, the administrative status of the device

with respect to whether it allows TWAMP sessions and if so in what capacity (e.g. Control-Client, Server or both) are typical instances of global configuration items. A second category includes attributes that can be configured on a per control connection basis, such as the Server IP address. A third category includes attributes related to per-test session attributes, for instance setting different values in the Differentiated Services Code Point (DSCP) field. Finally, the data model could include attributes that relate to the operational state of the TWAMP implementation.

As we describe the TWAMP data model in the remaining sections of this document, readers should keep in mind the functional entity grouping illustrated in Figure 1.

3.1. Control-Client

A TWAMP Control-Client has an administrative status field set at the device level that indicates whether the node is enabled to function as such.

Each TWAMP Control-Client is associated with zero or more TWAMP control connections. The main configuration parameters of each control connection are:

- A name which can be used to uniquely identify, at the Control-Client, a particular control connection. This name is necessary

  for programmability reasons because at the time of creation of a TWAMP control connection not all IP and TCP port information needed to uniquely identify the connection is available.

- The IP address of the interface the Control-Client will use for connections

- The IP address of the remote Server

- Authentication and Encryption attributes such as KeyID, Token and the Client Initialization Vector (Client-IV) [RFC4656].

Each TWAMP control connection, in turn, is associated with zero or more test sessions. For each test session we note the following configuration items:

- The test session name that uniquely identifies a particular test session at the Control-Client and Session-Sender. Similarly to the control connections above, this unique test session name is needed because at the time of creation of a test session the source UDP port is not known to uniquely identify the test session.

- The IP address and UDP port number of the Session-Sender of the path under test by TWAMP

- The IP address and UDP port number of the Session-Reflector of said path

- Information pertaining to the test packet stream, such as the test starting time or whether the test should be repeated.

3.2. Server

Each TWAMP Server has an administrative status field set at the device level to indicate whether the node is enabled to function as a TWAMP Server.

Each TWAMP Server is associated with zero or more control connections. Each control connection is uniquely identified by the 4-tuple {Control-Client IP address, Control-Client TCP port number, Server IP address, Server TCP port}. Control connection configuration items on a TWAMP Server are read-only.

3.3. Session-Sender

There is one TWAMP Session-Sender instance for each test session that is initiated from the sending device. Primary configuration fields include:

- The test session name that MUST be identical with the corresponding test session name on the TWAMP Control-Client

(Section 3.1)

- The control connection name, which along with the test session name uniquely identify the TWAMP Session-Sender instance

- Information pertaining to the test packet stream, such as, for example, the value used in the DSCP packet header field and the number of test packets.

3.4. Session-Reflector

Each TWAMP Session-Reflector is associated with zero or more test sessions. For each test session, the REFWAIT parameter can be configured. Read-only access to other data model parameters, such as the Sender IP address is foreseen. Each test session can be uniquely identified by the 4-tuple mentioned in Section 3.2.

4. Data Model Parameters

This section defines the TWAMP data model using UML and describes all associated parameters.

4.1. Control-Client

The twampClient container (see Figure 3) holds items that are related to the configuration of the TWAMP Control-Client logical entity. These are divided up into items that are associated with the configuration of the Control-Client as a whole (e.g. clientAdminState) and items that are associated with individual control connections initiated by that Control-Client entity (twampClientCtrlConnection).

+------------------+

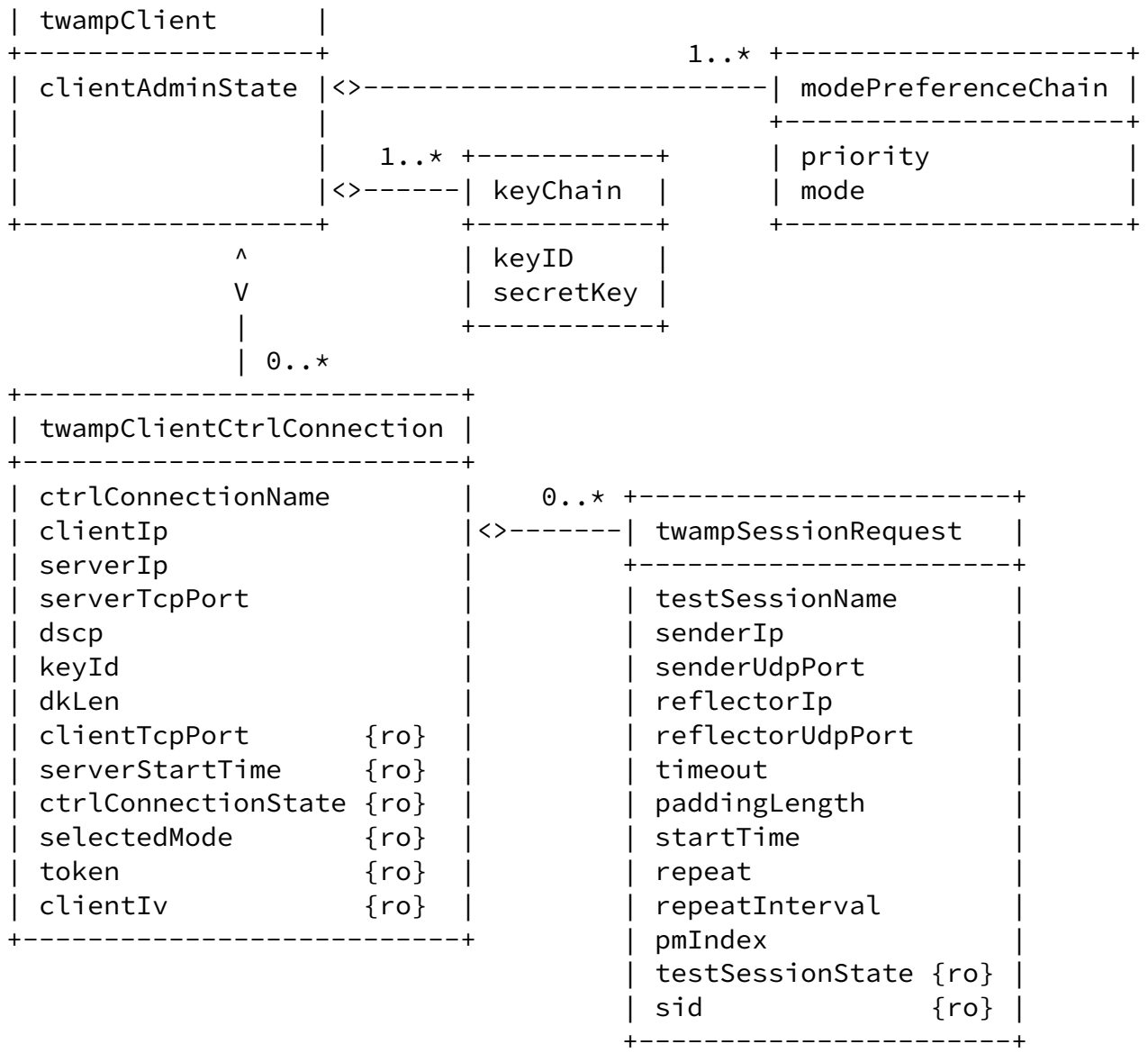


Figure 3: TWAMP Control-Client UML class diagram

The twampClient container includes an administrative parameter (clientAdminState) that controls whether the device is allowed to initiate TWAMP control and test sessions.

The twampClient container holds a list which specifies the preferred Mode values according to their preferred order of use, including the authentication and encryption Modes. Specifically, modePreferenceChain lists each priority (expressed as a 16-bit unsigned integer, where zero is the highest priority and subsequent values monotonically increasing) with their corresponding mode (expressed as a 32-bit Hexadecimal value). Depending on the Modes available in the Server Greeting, the Control-Client MUST choose the highest priority Mode from the configured modePreferenceChain list.

Note that the list of preferred Modes may set bit position combinations when necessary, such as when referring to the extended TWAMP features in [RFC5618], [RFC5938], and [RFC6038]. If the Control-Client cannot determine an acceptable Mode, it MUST respond with zero Mode bits set in the Set-up Response message, indicating it will not continue with the control connection.

In addition, the twampClient container holds a list named keyChain which relates KeyIDs with the respective secret keys. Both the Server and the Control-Client use the same mappings from KeyIDs to shared secrets. The Server, being prepared to conduct sessions with more than one Control-Client, uses KeyIDs to choose the appropriate secret key; a Control-Client would typically have different secret keys for different Servers. keyId is a UTF-8 string, up to 80 octets in length (if the string is shorter, it is padded with zero octets), that tells the Server which shared secret the Control-Client wishes to use to authenticate or encrypt. The secretKey is the shared secret, an octet string of arbitrary length whose interpretation as a text string is unspecified. In the interest of interoperability, however, the UTF-8 text encoding MUST be used for secretKey.

Each twampClient container also holds a list of twampClientCtrlConnection, where each item in the list describes a TWAMP control connection that will be initiated by this Control-Client. There SHALL be one instance of twampClientCtrlConnection per TWAMP Control (TCP) connection that is to be initiated from this device.

The configuration items for twampClientCtrlConnection are:

ctrlConnectionName
: A unique name used as a key to identify this individual TWAMP control connection on the Control-Client device.

clientIp
: The IP address of the local Control-Client device, to be placed in the source IP address field of the IP header in TWAMP TCP control packets belonging to this control connection. If not configured, the device SHALL choose its own source IP address.

serverIp
: The IP address belonging to the remote Server device to which the control connection will be initiated to.

serverTcpPort

This parameter defines the TCP port number that is to be used by this outgoing TWAMP control connection. Typically, this

is the well-known TWAMP port number (862) [RFC5357]. However, there are known realizations of TWAMP in the field that were implemented before this well-known port number was allocated. These early implementations allowed the port number to be configured. This parameter is therefore provided for backward compatibility reasons.

dscp The DSCP value to be placed in the TCP header of TWAMP-Control packets generated by this Control-Client.

keyId
The keyId value that is selected for this control connection. KeyID a UTF-8 string, up to 80 octets in length (if the string is shorter, it is padded with zero octets).

dkLen Intended length in octets of the derived key, a positive integer, at most (2^32 - 1) * hLen.

The following twampClientCtrlConnection parameters are read-only:

clientTcpPort
The source TCP port used in the TWAMP control packets belonging to this control connection.

serverStartTime
The Start-Time advertized by the Server in the Server-Start message ([RFC4656], Section 3.1). This is a timestamp representing the time when the current instantiation of the Server started operating.

ctrlConnectionState
The control connection state can be either active or idle.

selectedMode
The mode that the Control-Client has chosen for this control connection as set in the Mode field of the Set-Up-Response message ([RFC4656], Section 3.1).

token This parameter holds the 64 octets containing the

concatenation of a 16-octet challenge, a 16-octet AES Session-key used for encryption, and a 32-octet HMAC-SHA1 Session-key used for authentication. AES Session-key and HMAC Session-key are generated randomly by the Control-Client. AES Session-key and HMAC Session-key MUST be generated with sufficient entropy not to reduce the security of the underlying cipher [RFC4086]. The token itself is encrypted using the AES (Advanced Encryption Standard) in Cipher Block Chaining (CBC). Encryption MUST be performed

using an Initialization Vector (IV) of zero and a key derived from the shared secret associated with KeyID. Challenge is the same as transmitted by the Server (Section 4.2) in the clear; see also the last paragraph of Section 6 in [RFC4656].

clientIv
: The Control-Client Initialization Vector (Client-IV) is generated randomly by the Control-Client. Client-IV merely needs to be unique (i.e., it MUST never be repeated for different sessions using the same secret key; a simple way to achieve that without the use of cumbersome state is to generate the Client-IV values using a cryptographically secure pseudo-random number source.

Each twampClientCtrlConnection holds a list of twampSessionRequest. twampSessionRequest holds information associated with the Control-Client for this test session. This includes information that is associated with the Request-TW-Session/Accept-Session message exchange ([RFC5357], Section 3.5). The Control-Client is also responsible for scheduling and results collection for test sessions, so twampSessionRequest will also hold information related these actions (e.g. pmIndex, repeatInterval). There SHALL be one instance of twampSessionRequest for each test session that is to be negotiated by this control connection via a Request-TW-Session/Accept-Session exchange.

The configuration items for twampSessionRequest are:

testSessionName
: A unique name for this test session to be used as a key for this test session on the Control-Client.

senderIp
: The IP address of the Session-Sender device, which is to be placed in the source IP address field of the IP header in TWAMP UDP test packets belonging to this test session. This value will be used to populate the sender address field of the Request-TW-Session message.

senderUdpPort
: The UDP port number that is to be used by the Session-Sender for this test session. A value of zero indicates that the Control-Client will auto-allocate a UDP port for this test session. This value is advertized in the sender port field of the Request-TW-session message.

reflectorIp

: The IP address belonging to the remote Session-Reflector device to which the TWAMP test session will be initiated. This value will be used to populate the receiver address field of the Request-TW-Session message.

reflectorUdpPort
: This parameter defines the UDP port number that will be used by the Session-Reflector for this test session. This value will be placed in the receiver port field of the Request-TW-Session message.

timeout
: The length of time (in seconds) that the Session-Reflector should continue to respond to packets belonging to this session after a Stop-Sessions control message has been received ([RFC5357], Section 3.8). This value will be placed in the timeout field of the Request-TW-Session message.

paddingLength
: The number of bytes of padding that will be added to the UDP test packets generated by the Session-Sender. This value will be placed in the Padding Length field of the Request-TW-Session message [RFC6038].

startTime
: Time when the session is to be started (but not before the

Start-Sessions command is issued). This value is placed in the Start Time field of the Request-TW-Session message. A value of 0 indicates that the session will be started as soon as the Start-Sessions message is received.

repeat and repeatInterval
These two values together are used to determine if the test session is to be run repeatedly. Once a test session has completed, the repeat parameter is checked. If the value indicates that this test session is to run again, then the parent control connection for this test session is restarted - and negotiates a new instance of this test session. This may happen immediately after the test session completes (if the repeatInterval is set to 0). Otherwise, the Control-Client will wait for the number of minutes specified in the repeatInterval before negotiating the new instance of this test session.

pmIndex Numerical index value of a Registered Metric in the Performance Metric Registry [I-D.ietf-ippm-metric-registry]. Output Statistics will be specified in the Registry entry.

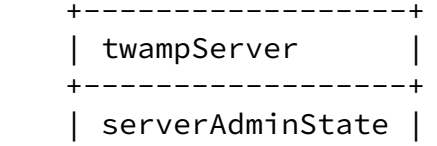
The following twampSessionRequest parameters are read-only:

testSessionState
The test session state can be either accepted or indicate the respective error code.

sid The SID allocated by the Server for this test session, and communicated back to the Control-Client in the SID field of the Accept-Session message; see Section 4.3 of [RFC6038].

4.2. Server

The twampServer container (see Figure 4) holds items that are related to the configuration of the TWAMP Server logical entity (recall Figure 1).



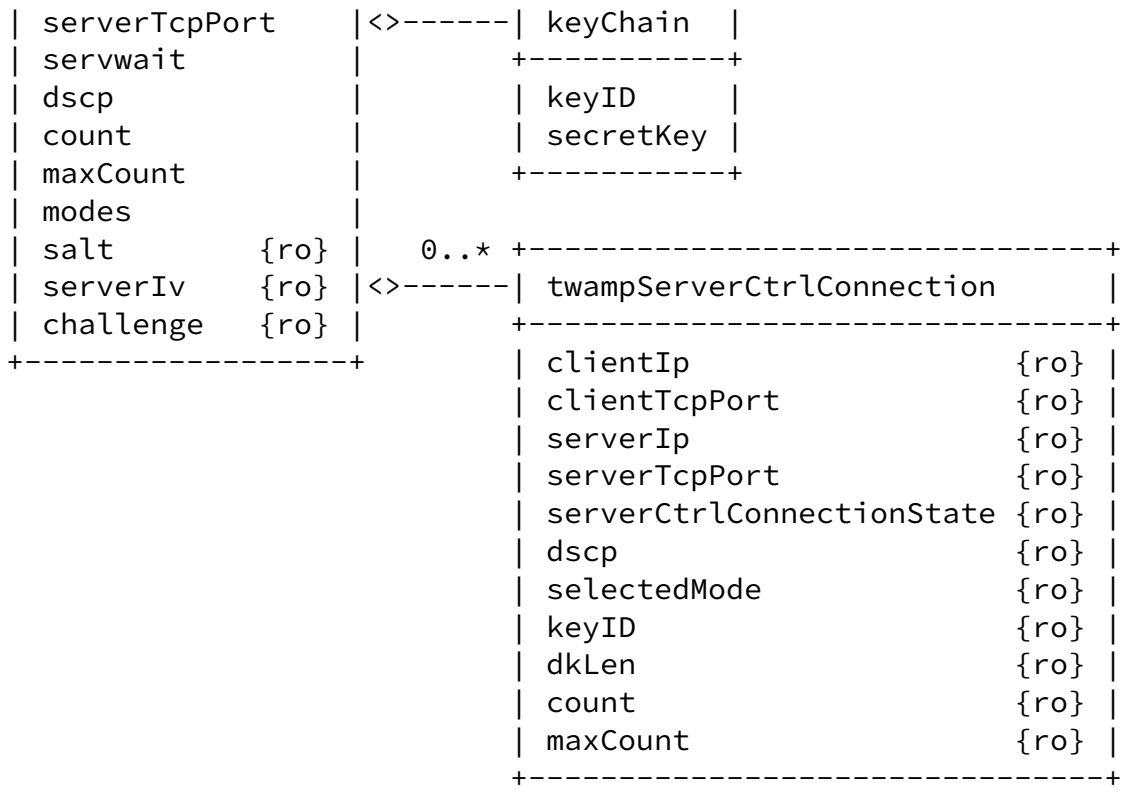


Figure 4: TWAMP Server UML class diagram

A device operating in the Server role cannot configure attributes on a per control connection basis, as it has no foreknowledge of what incoming TWAMP control connections it will receive. As such, any parameter that the Server might want to apply to an incoming control connection must be configured at the overall Server level, and will then be applied to all incoming TWAMP control connections.

Each twampServer container holds a list named keyChain which relates KeyIDs with the respective secret keys. As mentioned in Section 4.1, both the Server and the Control-Client use the same mappings from KeyIDs to shared secrets. The Server, being prepared to conduct sessions with more than one Control-Client, uses KeyIDs to choose the appropriate secret key; a Control-Client would typically have different secret keys for different Servers. keyId is a UTF-8 string, up to 80 octets in length (if the string is shorter, it is padded with zero octets), that tells the Server which shared secret the Control-Client wishes to use to authenticate or encrypt.

Each incoming control connection that is active on the Server will be represented by an instance of a twampServerCtrlConnection object. All items in the twampServerCtrlConnection object are read-only.

The twampServer container items are as follows:

serverAdminState
: This administrative parameter controls whether the device is allowed to operate as a TWAMP Server. As defined in [RFC5357] the roles of Server and Session-Reflector can be played by the same host; recall Figure 2. For a host operating in this manner, this parameter controls whether the device is allowed to respond to TWAMP control and test sessions.

serverTcpPort
: This parameter defines the well known TCP port number that is used by TWAMP. The Server will listen on this port for incoming TWAMP control connections. Although this is defined as a fixed value (862) in [RFC5357], there are several realizations of TWAMP in the field that were implemented before this well-known port number was allocated. These early implementations allowed the port number to be configured. This parameter is therefore provided for backward compatibility reasons.

servwait
: TWAMP Control (TCP) session timeout, in seconds.

dscp
: The DSCP value to be placed in the TCP header of TWAMP-Control packets generated by the Server.

count
: Parameter used in deriving a key from a shared secret as described in Section 3.1 of [RFC4656], and are communicated to the Control-client as part of Server Greeting message. Count MUST be a power of 2. Count MUST be at least 1024.

Count SHOULD be increased as more computing power becomes common.

maxCount

If an attacking system sets the maximum value in Count (2**32), then the system under attack would stall for a significant period of time while it attempts to generate keys. Therefore, TWAMP-compliant systems SHOULD have a configuration control to limit the maximum Count value. The default maximum Count value SHOULD be 32768.

modes
: The bit mask of TWAMP Modes this Server instance is willing to support; see IANA TWAMP Modes Registry. Each bit position set represents a mode; see TWAMP-Modes at http://www.iana.org/assignments/twamp-parameters/twamp-parameters.xhtml. Note: Modes requiring Authentication or Encryption MUST include the related attributes.

The following parameters are read-only:

salt
: A parameter used in deriving a key from a shared secret as described in Section 3.1 of [RFC4656]. Salt MUST be generated pseudo-randomly (independently of anything else in the RFC) and is communicated to the Control-Client as part of the Server greeting message.

serverIv
: The Server Initialization Vector (IV) is generated randomly by the server.

challenge
: Challenge is a random sequence of octets generated by the server. As described in Section 4.1 challenge is used by the Control-Client to prove possession of a shared secret.

There SHALL be one instance of twampServerCtrlConnection per incoming TWAMP TCP Control connection that is received and active on the Server device. All items in the twampServerCtrlConnection are read-only. Each instance of twampServerCtrlConnection uses the following 4-tuple as its unique key: clientIp, clientTcpPort, serverIp, serverTcpPort.

The twampServerCtrlConnection container items are all read-only:

clientIp

The IP address on the remote Control-Client device, which is the source IP address used in the TWAMP TCP control packets belonging to this control connection.

clientTcpPort
The source TCP port used in the TWAMP TCP control packets belonging to this control connection.

serverIp
The IP address of the local Server device, which is the destination IP address used in the TWAMP TCP control packets belonging to this control connection.

serverTcpPort
The destination TCP port used in the TWAMP TCP control packets belonging to this control connection. This will usually be the same value as is configured under twampServer. However, in the event that the user re-configured twampServer:serverTcpPort after this control connection was initiated, this value will indicate the serverTcpPort that is actually in use for this control connection.

serverCtrlConnectionState
The Server control connection state can be active or SERVWAIT.

dscp
The DSCP value used in the header of the TCP control packets sent by the Server for this control connection. This will usually be the same value as is configured for twampServer:dscp under the twampServer. However, in the event that the user re-configures twampServer:dscp after this control connection is already in progress, this read-only value will show the actual dscp value in use by this control connection.

selectedMode
The mode that was chosen for this control connection as set in the Mode field of the Set-Up-Response message.

keyId
The keyId value that is in use by this control connection. KeyID a UTF-8 string, up to 80 octets in length (if the string is shorter, it is padded with zero octets). The Control-Client selects the keyID for the control connection.

dkLen

The dkLen value that is in use by this control connection. This will usually be the same value as is configured under twampServer. However, in the event that the user re-configured twampServer:dkLen after this control connection is already in progress, this read-only value will show the actual dkLen that is in use for this control connection.

count

The count value that is in use by this control connection. This will usually be the same value as is configured under twampServer. However, in the event that the user re-configured twampServer:count after this control connection is already in progress, this read-only value will show the actual count that is in use for this control connection.

maxCount

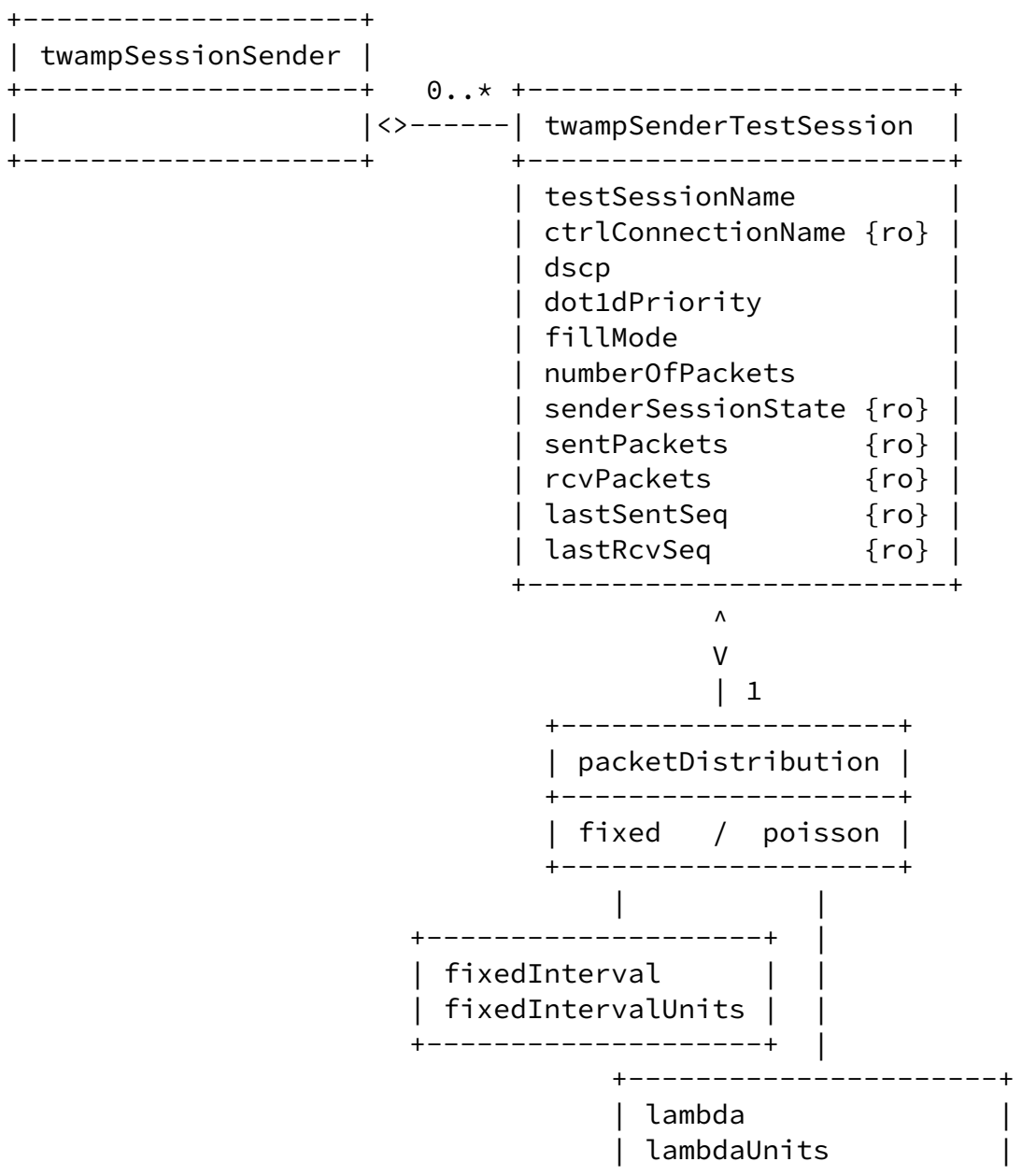
The maxCount value that is in use by this control connection. This will usually be the same value as is configured under twampServer. However, in the event that the user re-configured twampServer:maxCount after this control connection is already in progress, this read-only value will show the actual maxCount that is in use for this control connection.

4.3. Session-Sender

The twampSessionSender container, illustrated in Figure 5, holds items that are related to the configuration of the TWAMP Session-Sender logical entity.

There are no global configuration items that apply to the Session-Sender entity as a whole.

There is one instance of twampSenderTestSession for each test session for which packets are being sent.


twampSessionSender
0..*
twampSenderTestSession
testSessionName
ctrlConnectionName {ro}
dscp
dot1dPriority
fillMode
numberOfPackets
senderSessionState {ro}
sentPackets {ro}
rcvPackets {ro}
lastSentSeq {ro}
lastRcvSeq {ro}
1
packetDistribution
fixed / poisson
fixedInterval
fixedIntervalUnits
lambda
lambdaUnits

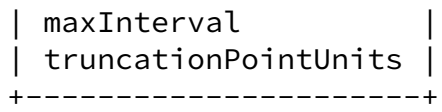


Figure 5: TWAMP Session-Sender UML class diagram

The twampSenderTestSession container items are:

testSessionName
: A unique name for this test session to be used as a key for this test session by the Session-Sender logical entity.

ctrlConnectionName
: The name of the parent control connection that is responsible for negotiating this test session.

dscp
: The DSCP value to be placed in the UDP header of TWAMP-Control packets generated by the Session-Sender.

dot1dPriority
: Priority Code Point (PCP) value to place in the Ethernet header of the TWAMP UDP test frames transmitted for this test session.

fillMode
: Indicates whether the padding added to the UDP test packets will contain pseudo-random numbers, or whether it should consist of all zeroes, as per Section 4.2.1 of [RFC5357].

numberOfPackets
: The overall number of UDP test packets to be transmitted by the Session-Sender for this test session.

packetDistribution
: Defines whether test packets are to be transmitted with a fixed interval between them, or whether a Poisson distribution is to be used.

fixedInterval and fixedIntervalUnits
: If packetDistribution is set to fixed, these two values are used together to determine the fixed time to wait between test packet transmissions for this test session.

fixedInterval is an unsigned floating point number, 8 significant digits. fixedIntervalUnits is one of seconds, milliseconds, microseconds, nanoseconds.

lambda and lambdaUnits
: If packetDistribution is Poisson, the lambda parameter defines the average rate of packet transmission. lambda is an unsigned floating point number, 8 significant digits. lambdaUnits defines the units of lambda in reciprocal seconds.

maxInterval
: If packetDistribution is Poisson, then this parameter keeps a stream active by setting a maximum time between packet transmissions.

truncationPointUnits
: One of seconds, milliseconds, microseconds, nanoseconds.

The following twampSenderTestSession parameters are read-only:

senderSessionState

This read-only item can be either Active or Idle.

sentPackets
: The number of UDP test packets belonging to this session that have been transmitted by the Session-Sender.

rcvPackets
: The number of UDP test packets belonging to this session that have been received from the Session-Reflector. The round trip loss for a test session can be calculated as sentPackets - rcvPackets.

lastSentSeq
: The value in the sequence number field of the last UDP test packet transmitted for this test session. Sequence numbers start from zero - so this should always be one less than the sentPackets value.

lastRcvSeq

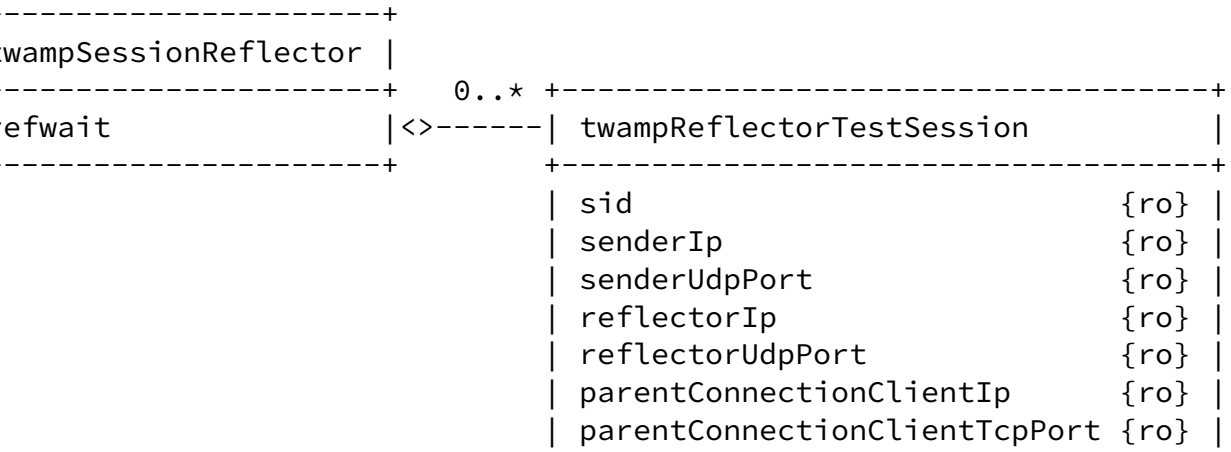
The value in the sequence number field of the last UDP test packet received for this test session. In the case of packet loss in the Session-Sender -> Session-Reflector direction, this value minus the lastSentSeq will identify the number of packets that were lost in the Session-Sender -> Session-Reflector direction.

4.4. Session-Reflector

The twampSessionReflector container, illustrated in Figure 6, holds items that are related to the configuration of the TWAMP Session-Reflector logical entity.

A device operating in the Session-Reflector role cannot configure attributes on a per-session basis, as it has no foreknowledge of what incoming sessions it will receive. As such, any parameter that the Session-Reflector might want to apply to an incoming test session must be configured at the overall Session-Reflector level, and will then be applied to all incoming sessions.

Each incoming test session that is active on the Session-Reflector will be represented by an instance of a twampReflectorTestSession object. All items in the twampReflectorTestSession object are read-only.



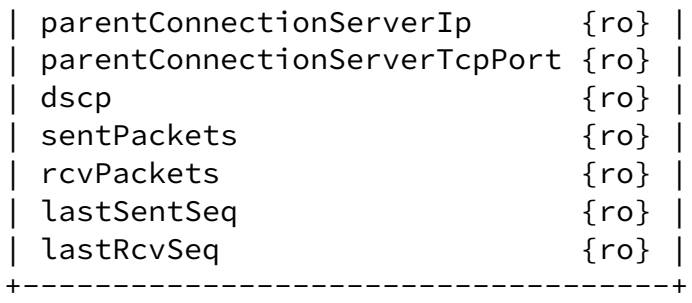


Figure 6: TWAMP Session-Reflector UML class diagram

The twampSessionReflector configuration items are:

refwait
: The Session-Reflector MAY discontinue any session that has been started when no packet associated with that session has been received for REFWAIT seconds. The default value of REFWAIT SHALL be 900 seconds, and this waiting time MAY be configurable. This timeout allows a Session-Reflector to free up resources in case of failure.

Instances of twampSessionReflector:twampReflectorTestSession are indexed by a session identifier (SID). This is a value that is auto-allocated by the Server as test session requests are received, and communicated back to the Control-Client in the SID field of the Accept-Session message; see Section 4.3 of [RFC6038].

When attempting to retrieve operational data for active test sessions from a Session-Reflector device, the user will not know what sessions are currently active on that device, or what SIDs have been auto-allocated for these test sessions. If the user has network access to the Control-Client device, then it is possible to read the data for this session under twampClient:twampClientCtrlConnection:twampSessionRequest and obtain the SID (see Figure 3). The user may then use this SID value as an index to retrieve an individual

twampSessionReflector:twampReflectorTestSession instance on the Session-Reflector device.

If the user has no network access to the Control-Client device, then the only option is to retrieve all twampReflectorTestSession

instances from the Session-Reflector device. This could be problematic if a large number of test sessions are currently active on that device.

Each Session-Reflector test session contains the following 4-tuple: {parentConnectionClientIp, parentConnectionClientTcpPort, parentConnectionServerIp, parentConnectionServerTcpPort}. This 4-tuple corresponds to the equivalent 4-tuple {clientIp, clientTcpPort, serverIp, serverTcpPort} in the twampServerCtrlConnection object. This four4-tuple allows the user to trace back from the test session to the parent control connection that negotiated this test session.

All data under twampReflectorTestSession is read-only:

sid
: An auto-allocated identifier for this test session, that is unique within the context of this Server/Session-Reflector device only. This value will be communicated to the Control-Client that requested the test session in the SID field of the Accept-Session message.

senderIp
: The IP address on the remote device, which is the source IP address used in the TWAMP UDP test packets belonging to this test session.

senderUdpPort
: The source UDP port used in the TWAMP UDP test packets belonging to this test session.

reflectorIp
: The IP address of the local Session-Reflector device, which is the destination IP address used in the TWAMP UDP test packets belonging to this test session.

reflectorUdpPort
: The destination UDP port used in the TWAMP UDP test packets belonging to this test session.

parentConnectionClientIp
: The IP address on the Control-Client device, which is the source IP address used in the TWAMP TCP control packets

belonging to the parent control connection that negotiated this test session.

parentConnectionClientTcpPort
: The source TCP port used in the TWAMP TCP control packets belonging to the parent control connection that negotiated this test session.

parentConnectionServerIp
: The IP address of the Server device, which is the destination IP address used in the TWAMP TCP control packets belonging to the parent control connection that negotiated this test session.

parentConnectionServerTcpPort
: The destination TCP port used in the TWAMP TCP control packets belonging to the parent control connection that negotiated this test session.

dscp
: The DSCP value present in the UDP header of TWAMP test packets belonging to this test session.

sentPackets
: The number of UDP test response packets that have been sent by the Session-Reflector for this test session.

rcvPackets
: The number of UDP test packets that have been received by the Session-Reflector for this test session. Since the Session-Reflector should respond to every test packet it receives, the sentPackets and rcvPackets values should always be identical.

lastSentSeq
: The value in the sequence number field of the last UDP test response packet transmitted for this test session.

lastRcvSeq
: The value in the sequence number field of the last UDP test packet received for this test session.

5. Data Model

5.1. Tree Diagram

This section presents the TWAMP YANG data tree defined in this document. Readers should keep in mind that the limit of 72

characters per line forces us to introduce artificial line breaks in some tree nodes.

```
module: twamp
   +--rw twamp
      +--rw twampClient {controlClient}?
      |  +--rw clientAdminState             boolean
      |  +--rw modePreferenceChain* [priority]
      |  |  +--rw priority    uint16
      |  |  +--rw mode?       enumeration
      |  +--rw keyChain* [keyId]
      |  |  +--rw keyId        string
      |  |  +--rw secretKey?   string
      |  +--rw twampClientCtrlConnection* [ctrlConnectionName]
      |     +--rw ctrlConnectionName     string
      |     +--rw clientIp?              inet:ip-address
      |     +--rw serverIp?              inet:ip-address
      |     +--rw serverTcpPort?         inet:port-number
      |     +--rw dscp?                  inet:dscp
      |     +--rw keyId?                 string
      |     +--rw dkLen?                 uint32
      |     +--ro clientTcpPort?         inet:port-number
      |     +--ro serverStartTime?       uint64
      |     +--ro ctrlConnectionState?   enumeration
      |     +--ro selectedMode?          enumeration
      |     +--ro token?                 string
      |     +--ro clientIv?              string
      |     +--rw twampSessionRequest* [testSessionName]
      |        +--rw testSessionName     string
      |        +--rw senderIp?           inet:ip-address
      |        +--rw senderUdpPort?      inet:port-number
      |        +--rw reflectorIp?        inet:ip-address
      |        +--rw reflectorUdpPort?   inet:port-number
      |        +--rw timeout?            uint64
      |        +--rw paddingLength?      uint32
      |        +--rw startTime?          uint64
      |        +--rw repeat?             boolean
      |        +--rw repeatInterval?     uint32
      |        +--rw pmIndex?            uint16
      |        +--ro testSessionState?   enumeration
      |        +--ro sid?                string
      +--rw twampServer {server}?
```

```
|  +--rw serverAdminState             boolean
|  +--rw serverTcpPort?               inet:port-number
|  +--rw servwait?                    uint32
|  +--rw dscp?                        inet:dscp
|  +--rw count?                       uint32
|  +--rw maxCount?                    uint32
```



```
|  +--rw modes?                       bits
|  +--ro salt?                        string
|  +--ro serverIv?                    string
|  +--rw challenge?                   string
|  +--rw keyChain* [keyId]
|  |  +--rw keyId        string
|  |  +--rw secretKey?   string
|  +--ro twampServerCtrlConnection* \
                [clientIp clientTcpPort serverIp serverTcpPort]
|     +--ro clientIp                      inet:ip-address
|     +--ro clientTcpPort                 inet:port-number
|     +--ro serverIp                      inet:ip-address
|     +--ro serverTcpPort                 inet:port-number
|     +--ro serverCtrlConnectionState?    enumeration
|     +--ro dscp?                         inet:dscp
|     +--ro selectedMode?                 enumeration
|     +--ro keyId?                        string
|     +--ro dkLen?                        uint32
|     +--ro count?                        uint32
|     +--ro maxCount?                     uint32
+--rw twampSessionSender {sessionSender}?
|  +--rw twampSenderTestSession* [testSessionName]
|     +--rw testSessionName          string
|     +--ro ctrlConnectionName?      string
|     +--rw dscp?                    inet:dscp
|     +--rw dot1dPriority?           uint8
|     +--rw fillMode?                enumeration
|     +--rw numberOfPackets?         uint32
|     +--rw (packetDistribution)?
|     |  +--:(fixed)
|     |  |  +--rw fixedInterval?          uint32
|     |  |  +--rw fixedIntervalUnits?     enumeration
|     |  +--:(poisson)
|     |     +--rw lambda?                 uint32
|     |     +--rw lambdaUnits?            uint32
```

```
|     |     +--rw maxInterval?            uint32
|     |     +--rw truncationPointUnits?   enumeration
|     +--ro senderSessionState?     enumeration
|     +--ro sentPackets?            uint32
|     +--ro rcvPackets?             uint32
|     +--ro lastSentSeq?            uint32
|     +--ro lastRcvSeq?             uint32
+--rw twampSessionReflector {sessionReflector}?
   +--rw refwait?                      uint32
   +--ro twampReflectorTestSession* \
          [senderIp senderUdpPort reflectorIp reflectorUdpPort]
      +--ro sid?                              string
      +--ro senderIp                          inet:ip-address
```



```
      +--ro senderUdpPort                     inet:port-number
      +--ro reflectorIp                       inet:ip-address
      +--ro reflectorUdpPort                  inet:port-number
      +--ro parentConnectionClientIp?         inet:ip-address
      +--ro parentConnectionClientTcpPort?    inet:port-number
      +--ro parentConnectionServerIp?         inet:ip-address
      +--ro parentConnectionServerTcpPort?    inet:port-number
      +--ro dscp?                             inet:dscp
      +--ro sentPackets?                      uint32
      +--ro rcvPackets?                       uint32
      +--ro lastSentSeq?                      uint32
      +--ro lastRcvSeq?                       uint32
```

5.2. YANG Module

This section presents the TWAMP YANG module defined in this document.

```
<CODE BEGINS> file "ietf-twamp@2015-03-06.yang"
module twamp {
  namespace "urn:ietf:params:xml:ns:yang:ietf-twamp";
   //namespace need to be assigned by IANA
  prefix "twamp";

  import ietf-inet-types {
    prefix inet;
  }

  organization
```

```
    "IETF IPPM (IP Performance Metrics) Working Group";

  contact
    "draft-cmzrjp-ippm-twamp-yang@tools.ietf.org";

  description "TWAMP Data Model";

  revision "2015-03-06" {
    description "Initial version. RFC5357 is covered.
      RFC5618, RFC5938 and RFC6038 are not covered.";
  }

  feature controlClient {
    description "This feature relates to the device functions as the
      TWAMP Control-Client.";
  }

  feature server {
    description "This feature relates to the device functions as the
      TWAMP Server.";
```



```
  }

  feature sessionSender {
    description "This feature relates to the device functions as the
      TWAMP Session-Sender.";
  }

  feature sessionReflector {
    description "This feature relates to the device functions as the
      TWAMP Session-Reflector.";
  }

  grouping maintenanceStatistics {
    leaf sentPackets {
      config "false";
      type uint32;
    }
    leaf rcvPackets {
      config "false";
      type uint32;
    }
```

```
      leaf lastSentSeq {
        config "false";
        type uint32;
      }
      leaf lastRcvSeq {
        config "false";
        type uint32;
      }
    }

    container twamp {
      container twampClient {
        if-feature controlClient;
        leaf clientAdminState {
          mandatory "true";
          type boolean;
          description "Indicates whether this device is allowed to run
            TWAMP to initiate control/test sessions";
        }

        list modePreferenceChain {
          key "priority";
          unique "mode";

          leaf priority {
            type uint16;
          }
```



```
          leaf mode {
            type enumeration {
              enum unauthenticated {
                value "1";
              }
              enum authenticated {
                value "2";
                }
              enum encrypted {
                value "4";
              }
              enum unauthtestencrpytcontrol {
                value "8";
              }
```

```
            enum individualsessioncontrol {
              value "16";
              }
            enum reflectoctets {
              value "32";
            }
            enum symmetricalsize {
              value "64";
            }
          }
        }
      }

      list keyChain {
        key "keyId";
        leaf keyId {
          type string {
            length "1..80";
          }
        }
        leaf secretKey {
          type string;
        }
      }

      list twampClientCtrlConnection {
        key "ctrlConnectionName";
        leaf ctrlConnectionName {
          type "string";
          description "A unique name used as a key to identify this
            individual TWAMP control connection on the
            Control-Client device.";
        }
        leaf clientIp {
```



```
          type inet:ip-address;
        }
        leaf serverIp {
          config "true";
          type inet:ip-address;
        }
        leaf serverTcpPort {
```

```
        type inet:port-number;
      }
      leaf dscp{
        type inet:dscp;
        description "The DSCP value to be placed in the IP header
          of the TWAMP TCP Control packets generated
          by the Control-Client";
      }
      leaf keyId {
        type string {
          length "1..80";
        }
      }
      leaf dkLen {
        type uint32;
      }
      leaf clientTcpPort {
        config "false";
        type inet:port-number;
      }
      leaf serverStartTime {
        config "false";
        type uint64;
      }
      leaf ctrlConnectionState {
        config "false";
        type enumeration {
          enum active {
            description "Control session is active.";
          }
          enum idle {
            description "Control session is idle.";
          }
        }
      }
      leaf selectedMode {
        config "false";
        type enumeration {
          enum unauthenticated {
            value "1";
          }
```

```
      enum authenticated {
        value "2";
        }
      enum encrypted {
        value "4";
      }
      enum unauthtestencrpytcontrol {
        value "8";
      }
      enum individualsessioncontrol {
        value "16";
        }
      enum reflectoctets {
        value "32";
      }
      enum symmetricalsize {
        value "64";
      }
    }
  }
  leaf token {
    config "false";
    type string {
      length "1..64";
    }
    description "64 octets, containing the concatenation of a
      16-octet challenge, a 16-octet AES Session-key used
      for encryption, and a 32-octet HMAC-SHA1 Session-key
      used for authentication";
  }
  leaf clientIv{
    config "false";
    type string {
      length "1..16";
    }
    description "16 octets, Client-IV is generated randomly
      by the Control-Client.";
  }

  list twampSessionRequest {
    key "testSessionName";
    leaf testSessionName {
      type "string";
    }
    leaf senderIp {
      type inet:ip-address;
    }
    leaf senderUdpPort {
```

```
            type inet:port-number;
          }
          leaf reflectorIp {
            type inet:ip-address;
          }
          leaf reflectorUdpPort {
            type inet:port-number;
          }
          leaf timeout {
            type uint64;
            description "The time Session-Reflector MUST wait after
                receiving a Stop-Session message";
          }
          leaf paddingLength {
            type uint32{
              range "64..1500";
            }
            description "The number of bytes of padding that should
                be added to the UDP test packets generated by the
                sender.";
          }
          leaf startTime {
            type uint64;
          }
          leaf repeat  {
            type boolean;
          }
          leaf repeatInterval  {
            type uint32;
            when "repeat='true'";
            description "Repeat interval (in minutes)";
          }
          leaf pmIndex {
            type uint16;
            description "Numerical index value of a Registered
                Metric in the Performance Metric Registry";
          }
          leaf testSessionState {
            config "false";
            type enumeration {
              enum ok {
                value 0;
                description "Test session is accepted.";
```

```
              }
              enum failed {
                value 1;
                description "Failure, reason unspecified
                    (catch-all).";
```



```
              }
              enum internalError {
                value 2;
                description "Internal error.";
              }
              enum notSupported {
                value 3;
                description "Some aspect of request is not
                    supported.";
              }
              enum permanentResLimit {
                value 4;
                description "Cannot perform request due to
                    permanent resource limitations.";
              }
              enum tempResLimit {
                value 5;
                description "Cannot perform request due to
                    temporary resource limitations.";
              }
            }
          }
          leaf sid{
            config "false";
            type string;
          }
        }
      }
    }

    container twampServer{
      if-feature server;
      leaf serverAdminState{
        type boolean;
        mandatory "true";
        description "Indicates whether this device is allowed to run
```

```
          TWAMP to respond to control/test sessions";
        }
        leaf serverTcpPort {
          type inet:port-number;
          default "862";
        }
        leaf servwait {
          type uint32 {
            range 1..604800;
          }
          default 900;
          description "SERVWAIT (TWAMP Control (TCP) session timeout),
```



```
            default value is 900";
        }
        leaf dscp {
          type inet:dscp;
          description "The DSCP value to be placed in the IP header
            of the TWAMP TCP Control packets generated by the Server";
        }
        leaf count {
          type uint32 {
            range 1024..4294967295;
          }
        }
        leaf maxCount {
          type uint32 {
            range 1024..4294967295;
          }
          default 32768;
        }
        leaf modes {
          type bits {
            bit unauthenticated {
              position 0;
            }
            bit authenticated {
              position 1;
            }
            bit encrypted {
              position 2;
            }
```

```
          bit unauthtestencryptcontrol {
            position 3;
          }
          bit individualsessioncontrol {
            position 4;
          }
          bit reflectoctets {
            position 5;
          }
          bit symmetricalsize {
            position 6;
          }
        }
      }
      leaf salt{
        config "false";
        type string {
          length "1..16";
        }
```



```
          description "Salt MUST be generated pseudo-randomly";
      }
      leaf serverIv {
        config "false";
        type string {
          length "1..16";
        }
        description "16 octets, Server-IV is generated randomly
          by the Control-Client.";
      }
      leaf challenge {
        config "false";
        type string {
          length "1..16";
        }
        description "Challenge is a random sequence of octets
          generated by the Server";
      }

      list keyChain {
        key "keyId";
        leaf keyId {
```

```
          type string {
            length "1..80";
          }
        }
        leaf secretKey {
          type string;
        }
      }

      list twampServerCtrlConnection {
        key "clientIp clientTcpPort serverIp serverTcpPort";
        config "false";
        leaf clientIp {
          type inet:ip-address;
        }
        leaf clientTcpPort {
          type inet:port-number;
        }
        leaf serverIp {
          type inet:ip-address;
        }
        leaf serverTcpPort {
          type inet:port-number;
        }
        leaf serverCtrlConnectionState {
          type enumeration {
```



```
            enum "active";
            enum "servwait";
          }
        }
        leaf dscp {
          type inet:dscp;
          description "The DSCP value used in the header of the TCP
            control packets sent by the Server for this control
            connection. This will usually be the same value as is
            configured for twampServer:dscp under the twampServer.
            However, in the event that the user re-configures
            twampServer:dscp after this control connection is already
            in progress, this read-only value will show the actual
            dscp value in use by this control connection.";
        }
```

```
        leaf selectedMode {
          type enumeration {
            enum unauthenticated {
              value "1";
            }
            enum authenticated {
              value "2";
              }
            enum encrypted {
              value "4";
            }
            enum unauthtestencrpytcontrol {
              value "8";
            }
            enum individualsessioncontrol {
              value "16";
              }
            enum reflectoctets {
              value "32";
            }
            enum symmetricalsize {
              value "64";
            }
          }
          description "The mode that was chosen for this control
            connection as set in the Mode field of the Set-Up-Response
            message.";
        }
        leaf keyId {
          type string {
            length "1..80";
          }
          description "The keyId value that is in use by this control
```



```
            connection.";
        }
        leaf dkLen {
          type uint32;
          description "The dkLen value that is in use by this control
            connection. This will usually be the same value as is
            configured under twampServer. In the event that the user
            re-configured twampServer:dkLen after this control
```

```
            connection is already in progress, this read-only value
            will show the actual dkLen that is in use for this
            control connection.";
        }
        leaf count {
          type uint32 {
            range 1024..4294967295;
          }
          description "The count value that is in use by this control
            connection. This will usually be the same value as is
            configured under twampServer. However, in the event that
            the user re-configured twampServer:count after this control
            connection is already in progress, this read-only value
            will show the actual count that is in use for this
            control connection.";
        }
        leaf maxCount {
          type uint32 {
            range 1024..4294967295;
          }
          description "The maxCount value that is in use by this
            control connection. This will usually be the same value
            as is configured under twampServer. However, in the event
            that the user re-configured twampServer:maxCount after
            this control connection is already in progress, this
            read-only value will show the actual maxCount that is
            in use for this control connection.";
        }
      }
    }

    container twampSessionSender {
      if-feature sessionSender;
      list twampSenderTestSession {
        key "testSessionName";
        leaf testSessionName {
          type string;
          description "A unique name for this test session to be used
            as a key for this test session by the Session-Sender
            logical entity.";
```



```
        }
```

```
        leaf ctrlConnectionName {
          config "false";
            type "string";
            description "The name of the parent control connection
              that is responsible for negotiating this test session.";
        }
        leaf dscp {
          type inet:dscp;
          description "The DSCP value to be placed in the header of
            TWAMP UDP test packets generated by the sender.";
        }
        leaf dot1dPriority {
          type uint8 {
            range "0..7";
          }
        }
        leaf fillMode {
          type enumeration {
            enum zero;
            enum random;
          }
          default zero;
        }
        leaf numberOfPackets {
          type uint32;
          description "The overall number of UDP test packets to be
            transmitted by the sender for this test session.";
        }
        choice packetDistribution {
          case fixed {
            leaf fixedInterval {
              type uint32;
            }
            leaf fixedIntervalUnits  {
              type enumeration {
                enum seconds;
                enum milliseconds;
                enum microseconds;
                enum nanoseconds;
              }
            }
          }
          case poisson {
            leaf lambda {
              type uint32;
            }
            leaf lambdaUnits {
```

```
            type uint32;
          }
          leaf maxInterval {
            type uint32;
          }
          leaf truncationPointUnits {
            type enumeration {
              enum seconds;
              enum milliseconds;
              enum microseconds;
              enum nanoseconds;
            }
          }
        }
      }
      leaf senderSessionState {
        config "false";
        type enumeration {
          enum setup {
            description "Test session is active.";
          }
          enum failure {
            description "Test session is idle.";
          }
        }
      }
      uses maintenanceStatistics;
    }
  }

  container twampSessionReflector {
    if-feature sessionReflector;
    leaf refwait {
      config "true";
      type uint32 {
        range 1..604800;
      }
      default 900;
      description "REFWAIT(TWAMP test session timeout),
        the default value is 900";
    }

    list twampReflectorTestSession {
      key "senderIp senderUdpPort reflectorIp reflectorUdpPort";
      config "false";
      leaf sid {
```

```
          type string;
        }
```



```
        leaf senderIp {
          type inet:ip-address;
        }
        leaf senderUdpPort {
          type inet:port-number;
        }
        leaf reflectorIp {
          type inet:ip-address;
        }
        leaf reflectorUdpPort {
          type inet:port-number;
        }
        leaf parentConnectionClientIp {
          type inet:ip-address;
        }
        leaf parentConnectionClientTcpPort {
          type inet:port-number;
        }
        leaf parentConnectionServerIp {
          type inet:ip-address;
        }
        leaf parentConnectionServerTcpPort {
          type inet:port-number;
        }
        leaf dscp {
          type inet:dscp;
          description "The DSCP value placed in the header of TWAMP
            UDP test packets generated by the Session-Sender.";
        }
        uses maintenanceStatistics;
      }
    }
  }
}
<CODE ENDS>
```

6. Data Model Examples

This section presents a simple but complete example of configuring

all four entities in Figure 1, based on the YANG module specified in Section 5. The example is illustrative in nature, but aims to be self-contained, i.e. were it to be executed in a real TWAMP implementation it would lead to a correctly configured test session. A more elaborated example, which also includes authentication parameters, is provided in Appendix A.

6.1. Control-Client

The following configuration example shows a Control-Client with clientAdminState enabled. In a real implementation this would permit the Control-Client functional entity to initiate TWAMP control connections and test sessions.

```
<twamp xmlns="urn:ietf:params:xml:ns:yang:ietf-twamp">
   <twampClient>
      <clientAdminState>True</clientAdminState>
   </twampClient>
</twamp>
```

The following configuration example shows a Control-Client with two instances of twampClientCtrlConnection, one called "RouterA" and another called "RouterB". Each control connection is to a different Server. The control connection named "RouterA" has two test session requests. The control connection with name "RouterB" has no test session requests.

```
   <twamp xmlns="urn:ietf:params:xml:ns:yang:ietf-twamp">
      <twampClient>

         <twampClientCtrlConnection>
            <ctrlConnectionName>RouterA</ctrlConnectionName>
            <clientIp>203.0.113.1</clientIp>
            <serverIp>203.0.113.2</serverIp>
            <twampSessionRequest>
               <testSessionName>Test1</testSessionName>
               <senderIp>10.1.1.1</senderIp>
               <senderUdpPort>4000</senderUdpPort>
               <reflectorIp>10.1.1.2</reflectorIp>
               <reflectorUdpPort>5000</reflectorUdpPort>
               <startTime>0</startTime>
            </twampSessionRequest>
            <twampSessionRequest>
               <testSessionName>Test2</testSessionName>
               <senderIp>203.0.113.1</senderIp>
               <senderUdpPort>4001</senderUdpPort>
               <reflectorIp>203.0.113.2</reflectorIp>
               <reflectorUdpPort>5001</reflectorUdpPort>
               <startTime>0</startTime>
            </twampSessionRequest>
         </twampClientCtrlConnection>
```

```
        <twampClientCtrlConnection>
           <ctrlConnectionName>RouterB</ctrlConnectionName>
           <clientIp>203.0.113.1</clientIp>
           <serverIp>203.0.113.3</serverIp>
        </twampClientCtrlConnection>

     </twampClient>
  </twamp>
```

6.2. Server

This configuration example shows a Server with serverAdminState enabled, which permits the device to respond to TWAMP control connections and test sessions.

```
<twamp xmlns="urn:ietf:params:xml:ns:yang:ietf-twamp">
   <twampServer>
      <serverAdminState>True</serverAdminState>
   </twampServer>
</twamp>
```


The following example presents a Server with the control connection corresponding to the control connection name (ctrlConnectionName) "RouterA" presented in Section 6.1.

```
  <twamp xmlns="urn:ietf:params:xml:ns:yang:ietf-twamp">
     <twampServer>

        <twampServerCtrlConnection>
           <clientIp>203.0.113.1</clientIp>
           <clientTcpPort>16341</clientTcpPort>
           <serverIp>203.0.113.2</serverIp>
           <serverTcpPort>862</serverTcpPort>
           <serverCtrlConnectionState>active</serverCtrlConnectionState>
        </twampServerCtrlConnection>

     </twampServer>
  </twamp>
```

6.3. Session-Sender

The following configuration example shows a Session-Sender with the two test sessions presented earlier in Section 6.1.


```
<twamp xmlns="urn:ietf:params:xml:ns:yang:ietf-twamp">
   <twampSessionSender>

      <twampSenderTestSession>
         <testSessionName>Test1</testSessionName>
         <ctrlConnectionName>RouterA</ctrlConnectionName>  // read-only
         <numberOfPackets>900</numberOfPackets>
         <packetDistribution>
            <fixedInterval>1</fixedInterval>
            <fixedIntervalUnits>seconds</fixedIntervalUnits>
         </packetDistribution>
```

```
        </twampSenderTestSession>

        <twampSenderTestSession>
           <testSessionName>Test2</testSessionName>
           <ctrlConnectionName>RouterA</ctrlConnectionName>  // read-only
           <numberOfPackets>900</numberOfPackets>
           <packetDistribution>
              <lambda>1</lambda>
              <lambdaUnits>1</lambdaUnits>
              <maxInterval>2</maxInterval>
              <truncationPointunits>seconds</truncationPointunits>
          </packetDistribution>
        </twampSenderTestSession>

    </twampSessionSender>
 </twamp>
```

6.4. Session-Reflector

The following example shows the two Session-Reflector test sessions corresponding to the test sessions presented in Section 6.3.

```
<twamp xmlns="urn:ietf:params:xml:ns:yang:ietf-twamp">
   <twampSessionReflector>

        <twampReflectorTestSession>
          <sid>1232</sid>
          <senderIp>10.1.1.1</senderIp>
          <reflectorIp>10.1.1.2</reflectorIp>
          <senderUdpPort>4000</senderUdpPort>
          <reflectorUdpPort>5000</reflectorUdpPort>
          <parentConnectionClientIp>
                    203.0.113.1
                  </parentConnectionClientIp>
          <parentConnectionClientTcpPort>
                    16341
                  </parentConnectionClientTcpPort>
```



```
          <parentConnectionServerIp>
                    203.0.113.2
                  </parentConnectionServerIp>
          <parentConnectionServerTcpPort>
```

```
                    862
                  </parentConnectionServerTcpPort>
            <sentPackets>2</sentPackets>
            <rcvPackets>2</rcvPackets>
            <lastSentSeq>1</lastSentSeq>
            <lastRcvSeq>1</lastRcvSeq>
         </twampReflectorTestSession>

         <twampReflectorTestSession>
            <sid>178943</sid>
            <senderIp>203.0.113.1</senderIp>
            <reflectorIp>192.68.0.2</reflectorIp>
            <senderUdpPort>4001</senderUdpPort>
            <parentConnectionClientIp>
                    203.0.113.1
                  </parentConnectionClientIp>
            <parentConnectionClientTcpPort>
                    16341
                  </parentConnectionClientTcpPort>
            <parentConnectionServerIp>
                    203.0.113.2
                  </parentConnectionServerIp>
            <parentConnectionServerTcpPort>
                    862
                  </parentConnectionServerTcpPort>
            <reflectorUdpPort>5001</reflectorUdpPort>
            <sentPackets>21</sentPackets>
            <rcvPackets>21</rcvPackets>
            <lastSentSeq>20</lastSentSeq>
            <lastRcvSeq>20</lastRcvSeq>
         </twampReflectorTestSession>

      </twampSessionReflector>
   </twamp>
```

7. Security Considerations

TBD

8. IANA Considerations

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

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

Registrant Contact: The IPPM WG of the IETF.

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

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

name: ietf-twamp

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

prefix: twamp

reference: RFC XXXX

9. Acknowledgements

Haoxing Shen contributed to the definition of the YANG module in Section 5.

Kostas Pentikousis is partially supported by FP7 UNIFY (http://fp7-unify.eu), a research project partially funded by the European Community under the Seventh Framework Program (grant agreement no. 619609). The views expressed here are those of the authors only. The European Commission is not liable for any use that may be made of the information in this document.

10. References

10.1. Normative References

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

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC4656] Shalunov, S., Teitelbaum, B., Karp, A., Boote, J., and M. Zekauskas, "A One-way Active Measurement Protocol (OWAMP)", RFC 4656, September 2006.

[RFC5357] Hedayat, K., Krzanowski, R., Morton, A., Yum, K., and J. Babiarz, "A Two-Way Active Measurement Protocol (TWAMP)", RFC 5357, October 2008.

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

[RFC6038] Morton, A. and L. Ciavattone, "Two-Way Active Measurement Protocol (TWAMP) Reflect Octets and Symmetrical Size Features", RFC 6038, October 2010.

10.2. Informative References

[I-D.elteto-ippm-twamp-mib]
Elteto, T. and G. Mirsky, "Two-Way Active Measurement Protocol (TWAMP) Management Information Base (MIB)", draft-elteto-ippm-twamp-mib-01 (work in progress), January 2014.

[I-D.ietf-ippm-metric-registry]
Bagnulo, M., Claise, B., Eardley, P., Morton, A., and A. Akhter, "Registry for Performance Metrics", draft-ietf-ippm-metric-registry-02 (work in progress), February 2015.

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-04 (work in progress), January 2015.

[I-D.unify-nfvrg-challenges]
Szabo, R., Csaszar, A., Pentikousis, K., Kind, M., and D. Daino, "Unifying Carrier and Cloud Networks: Problem Statement and Challenges", draft-unify-nfvrg-challenges-00 (work in progress), October 2014.

[NSC] John, W., Pentikousis, K., et al., "Research directions in network service chaining", Proc. SDN for Future Networks and Services (SDN4FNS), Trento, Italy IEEE, November 2013.

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC5618] Morton, A. and K. Hedayat, "Mixed Security Mode for the

Two-Way Active Measurement Protocol (TWAMP)", RFC 5618, August 2009.

[RFC5938] Morton, A. and M. Chiba, "Individual Session Control Feature for the Two-Way Active Measurement Protocol (TWAMP)", RFC 5938, August 2010.

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC7426] Haleplidis, E., Pentikousis, K., Denazis, S., Hadi Salim, J., Meyer, D., and O. Koufopavlou, "Software-Defined Networking (SDN): Layers and Architecture Terminology", RFC 7426, January 2015.

Appendix A. Detailed Data Model Examples

In this section we extend the example presented in Section 6 by configuring more fields such as authentication parameters, dscp values and so on.

A.1. Control-Client

```
<twamp xmlns="urn:ietf:params:xml:ns:yang:ietf-twamp">
   <twampClient>

      <clientAdminState>True</clientAdminState>

      <modePreferenceChain>
         <priority>0</priority>
         <mode>0x00000002</mode>
      </modePreferenceChain>
      <modePreferenceChain>
         <priority>1</priority>
         <mode>0x00000001</mode>
      </modePreferenceChain>

      <keychain>
         <keyid>KeyClient1ToRouterA</keyid>
```

```
            <secretKey>secret1</secretKey>
         </keychain>
         <keychain>
            <keyid>KeyForRouterB</keyid>
            <secretKey>secret2</secretKey>
         </keychain>

         <twampClientCtrlConnection>
            <ctrlConnectionName>RouterA</ctrlConnectionName>
            <clientIp>203.0.113.1</clientIp>
            <serverIp>203.0.113.2</serverIp>
            <dscp>32</dscp>
            <keyId>KeyClient1ToRouterA</keyId>
            <dkLen>1024</dkLen>
            <twampSessionRequest>
```



```
               <testSessionName>Test1</testSessionName>
               <senderIp>10.1.1.1</senderIp>
               <senderUdpPort>4000</senderUdpPort>
               <reflectorIp>10.1.1.2</reflectorIp>
               <reflectorUdpPort>5000</reflectorUdpPort>
               <paddingLength>0</paddingLength>
               <startTime>0</startTime>
               <testSessionState>ok</testSessionState>
               <sid>1232</sid>
            </twampSessionRequest>
            <twampSessionRequest>
               <testSessionName>Test2</testSessionName>
               <senderIp>203.0.113.1</senderIp>
               <senderUdpPort>4001</senderUdpPort>
               <reflectorIp>203.0.113.2</reflectorIp>
               <reflectorUdpPort>5001</reflectorUdpPort>
               <paddingLenth>32</paddingLenth>
               <startTime>0</startTime>
               <testSessionState>ok</testSessionState>
               <sid>178943</sid>
            </twampSessionRequest>
         </twampClientCtrlConnection>

      </twampClient>
   </twamp>
```

A.2. Server


```
  <twamp xmlns="urn:ietf:params:xml:ns:yang:ietf-twamp">
     <twampServer>

        <serverAdminState>True</serverAdminState>
        <servwait>1800</servwait>
        <dscp>32</dscp>
        <modes>0x00000003</modes>
        <dkLen>1024</dkLen>
        <count>256</count>

        <keychain>
           <keyid>KeyClient1ToRouterA</keyid>
           <secretKey>secret1</secretKey>
        </keychain>
        <keychain>
           <keyid>KeyClient10ToRouterA</keyid>
           <secretKey>secret10</secretKey>
        </keychain>
```

```
        <twampServerCtrlConnection>
           <clientIp>203.0.113.1</clientIp>
           <clientTcpPort>16341</clientTcpPort>
           <serverIp>203.0.113.2</serverIp>
           <serverTcpPort>862</serverTcpPort>
           <serverCtrlConnectionState>active</serverCtrlConnectionState>
           <dscp>32</dscp>
           <selectedMode>0x00000002</selectedMode>
           <keyId>KeyClient1ToRouterA</keyId>
           <count>1024</count>
        </twampServerCtrlConnection>

     </twampServer>
  </twamp>
```

A.3. Session-Sender


```
 <twamp xmlns="urn:ietf:params:xml:ns:yang:ietf-twamp">
    <twampSessionSender>

       <twampSenderTestSession>
          <testSessionName>Test1</testSessionName>
          <ctrlConnectionName>RouterA</ctrlConnectionName>  // read-only
          <dscp>32</dscp>
          <fillMode>zero</fillMode>
          <numberOfPackets>900</numberOfPackets>
          <packetDistribution>
             <fixedInterval>1</fixedInterval>
             <fixedIntervalUnits>seconds</fixedIntervalUnits>
```

```
          </packetDistribution>
          <senderSessionState>Active</senderSessionState>
          <sentPackets>2</sentPackets>
          <rcvPackets>2</rcvPackets>
          <lastSentSeq>1</lastSentSeq>
          <lastRcvSeq>1</lastRcvSeq>
       </twampSenderTestSession>

       <twampSenderTestSession>
          <testSessionName>Test2</testSessionName>
          <ctrlConnectionName>RouterA</ctrlConnectionName>  // read-only
          <dscp>32</dscp>
          <fillMode>random</fillMode>
          <numberOfPackets>900</numberOfPackets>
          <packetDistribution>
            <lambda>1</lambda>
             <lambdaUnits>1</lambdaUnits>
             <maxInterval>2</maxInterval>
             <truncationPointunits>seconds</truncationPointunits>
          </packetDistribution>
          <senderSessionState>Active</senderSessionState>
          <sentPackets>21</sentPackets>
          <rcvPackets>21</rcvPackets>
          <lastSentSeq>20</lastSentSeq>
          <lastRcvSeq>20</lastRcvSeq>
       </twampSenderTestSession>

    </twampSessionSender>
 </twamp>
```

A.4. Session-Reflector

```
   <twamp xmlns="urn:ietf:params:xml:ns:yang:ietf-twamp">
      <twampSessionReflector>

         <twampReflectorTestSession>
```



```
            <sid>1232</sid>
            <senderIp>10.1.1.1</senderIp>
            <reflectorIp>10.1.1.2</reflectorIp>
            <senderUdpPort>4000</senderUdpPort>
            <reflectorUdpPort>5000</reflectorUdpPort>
```

```
            <parentConnectionClientIp>
               203.0.113.1
            </parentConnectionClientIp>
            <parentConnectionClientTcpPort>
               16341
            </parentConnectionClientTcpPort>
            <parentConnectionServerIp>
               203.0.113.2
            </parentConnectionServerIp>
            <parentConnectionServerTcpPort>
               862
            </parentConnectionServerTcpPort>
            <dscp>32</dscp>
            <sentPackets>2</sentPackets>
            <rcvPackets>2</rcvPackets>
            <lastSentSeq>1</lastSentSeq>
            <lastRcvSeq>1</lastRcvSeq>
         </twampReflectorTestSession>

         <twampReflectorTestSession>
            <sid>178943</sid>
            <senderIp>203.0.113.1</senderIp>
            <reflectorIp>192.68.0.2</reflectorIp>
            <senderUdpPort>4001</senderUdpPort>
            <parentConnectionClientIp>
               203.0.113.1
            </parentConnectionClientIp>
            <parentConnectionClientTcpPort>
               16341
            </parentConnectionClientTcpPort>
            <parentConnectionServerIp>
               203.0.113.2
            </parentConnectionServerIp>
            <parentConnectionServerTcpPort>
               862
            </parentConnectionServerTcpPort>
            <reflectorUdpPort>5001</reflectorUdpPort>
            <dscp>32</dscp>
            <sentPackets>21</sentPackets>
            <rcvPackets>21</rcvPackets>
            <lastSentSeq>20</lastSentSeq>
            <lastRcvSeq>20</lastRcvSeq>
         </twampReflectorTestSession>
```

```
      </twampSessionReflector>
   </twamp>
```

Authors' Addresses

Ruth Civil
Ciena Corporation
307 Legget Drive
Kanata, ON K2K 3C8
Canada

Email: gcivil@ciena.com
URI: www.ciena.com

Al Morton
AT&T Labs
200 Laurel Avenue South
Middletown,, NJ 07748
USA

Phone: +1 732 420 1571
Fax: +1 732 368 1192
Email: acmorton@att.com
URI: http://home.comcast.net/~acmacm/

Lianshu Zheng
Huawei Technologies
China

Email: vero.zheng@huawei.com

Reshad Rahman
Cisco Systems
2000 Innovation Drive
Kanata, ON K2K 3E8
Canada

Email: rrahman@cisco.com

Mahesh Jethanandani
Ciena Corporation
3939 North 1st Street
San Jose, CA 95134
USA

Email: mjethanandani@gmail.com
URI: www.ciena.com

Kostas Pentikousis (editor)
EICT GmbH
EUREF-Campus Haus 13
Torgauer Strasse 12-15
10829 Berlin
Germany

Email: k.pentikousis@eict.de