IPPM WG
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2016

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
October 19, 2015

Two-Way Active Measurement Protocol (TWAMP) Data Model
draft-cmzrjp-ippm-twamp-yang-02

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 April 21, 2016.

Copyright Notice

Table of Contents

1. Introduction 3
1.1. Motivation 3
1.2. Terminology 3
1.3. Document Organization 3
2. Scope, Model, and Applicability 4
3. Data Model Overview 5
3.1. Control-Client 5
3.2. Server 6
3.3. Session-Sender 7
3.4. Session-Reflector 7
4. Data Model Parameters 7
4.1. Control-Client 7
4.2. Server 14
4.3. Session-Sender 18
4.4. Session-Reflector 21
5. Data Model 25
5.1. YANG Tree Diagram 25
5.2. YANG Module 27
6. Data Model Examples 43
6.1. Control-Client 43
6.2. Server 44
6.3. Session-Sender 45
6.4. Session-Reflector 46
7. Security Considerations 47
8. IANA Considerations 48
9. Acknowledgements 48
10. References 48
10.1. Normative References 48
10.2. Informative References 49
Appendix A. Detailed Data Model Examples 50
A.1. Control-Client 51
A.2. Server 52
A.3. Session-Sender 53
A.4. Session-Reflector 54
Appendix B. TWAMP Operational Commands 55
Authors' Addresses 55

1. Introduction

The Two-Way Active Measurement Protocol (TWAMP) [RFC5357] is used to measure network performance parameters such as latency, bandwidth, and packet loss by sending probe packets and measuring 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. This document addresses this gap by formally specifying the TWAMP data model using YANG.

1.1. Motivation

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

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 in YANG the TWAMP data model. Section 6 lists illustrative

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

2. Scope, Model, and Applicability

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

Figure 1 illustrates a redrawn version of the TWAMP logical model found in Section 1.2 of [RFC5357]. 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 Figure 1 are unspecified and may be proprietary protocols.

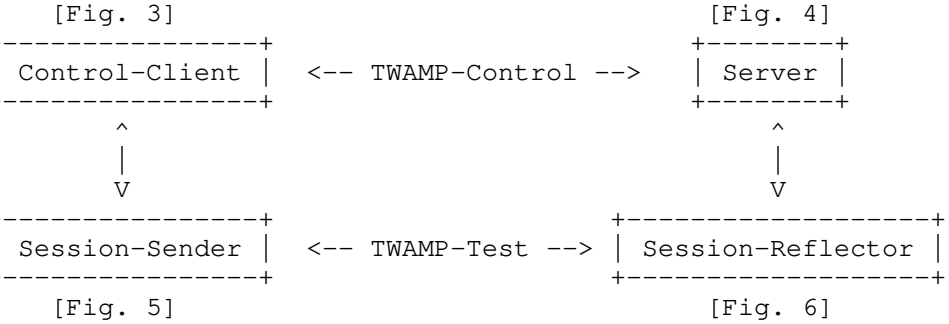


Figure 1: Annotated TWAMP logical model

As per [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. Appendix B considers TWAMP operational commands, which are also outside the scope of this document.

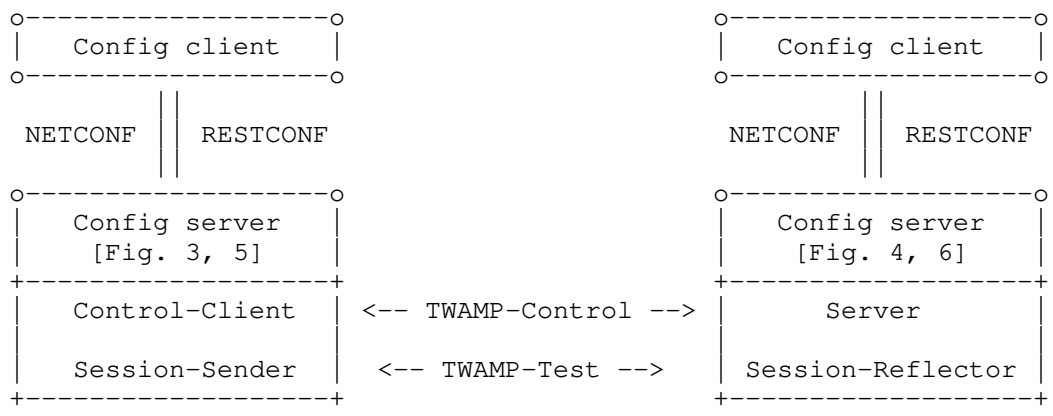


Figure 2: Simplified TWAMP model and protocols

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 number 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, for example, the source UDP port number 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 number of test packets and the packet distribution to be employed.

3.4. Session-Reflector

Each TWAMP Session-Reflector is associated with zero or more test sessions. For each test session, the REFWAIT parameter (Section 4.2 of [RFC5357] 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 twamp-client 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. client-admin-state) and items that are associated with individual control connections initiated by the Control-Client entity (twamp-client-ctrl-connection).

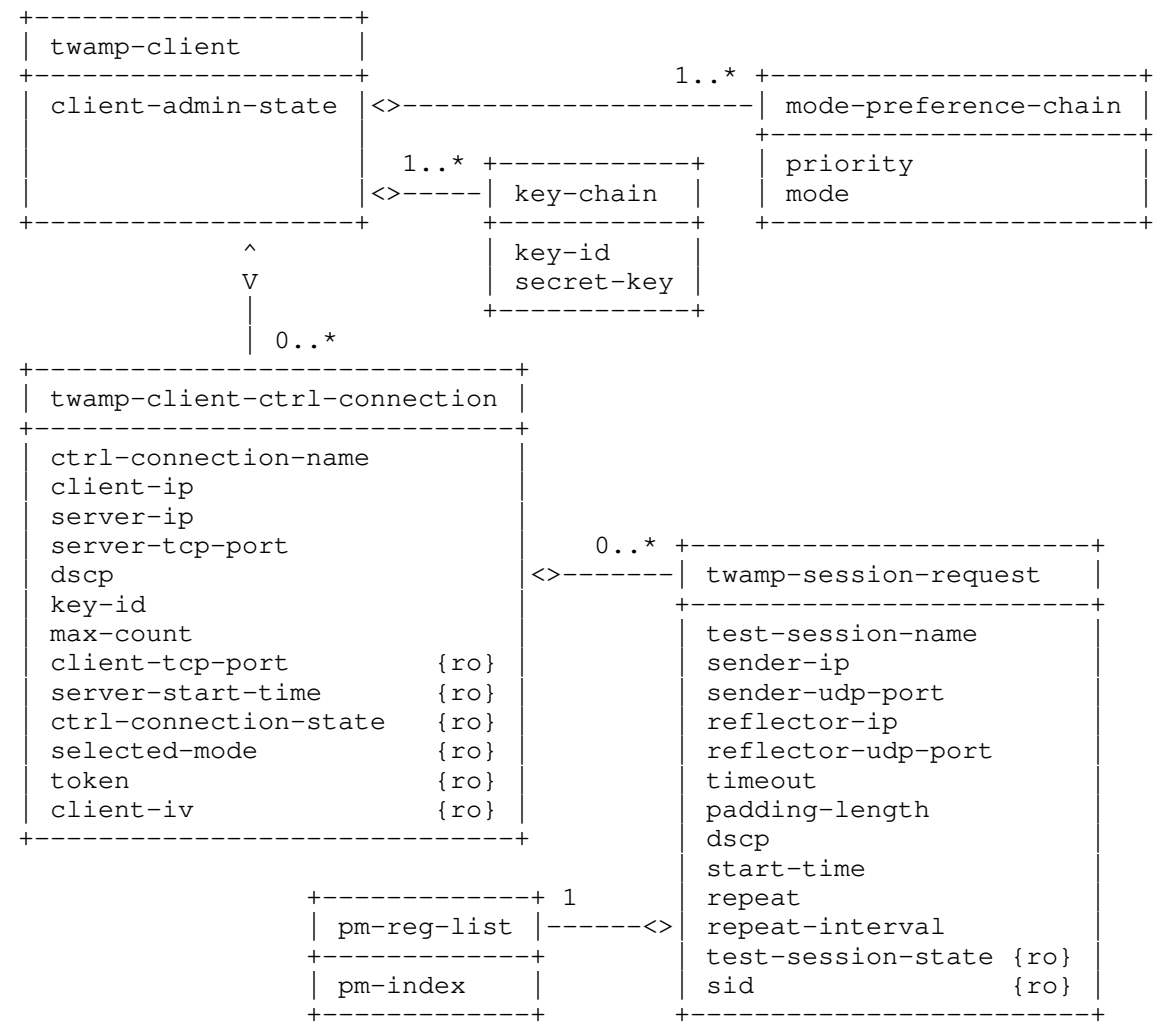


Figure 3: TWAMP Control-Client UML class diagram

The twamp-client container includes an administrative parameter (client-admin-state) that controls whether the device is allowed to initiate TWAMP control sessions.

The twamp-client container holds a list (mode-preference-chain) which specifies the preferred Mode values according to their preferred order of use, including the authentication and encryption Modes. Specifically, mode-preference-chain 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 mode-preference-chain 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 twamp-client container holds a list named key-chain which relates KeyIDs with the respective secret keys. Both the Server and the Control-Client use the same mappings from KeyIDs to shared secrets (key-id and secret-key in Figure 3, respectively). 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. The secret-key is the shared secret, an octet string of arbitrary length whose interpretation as a text string is unspecified. The key-id and secret-key encoding should follow Section 9.4 of [RFC6020]. The derived key length (dkLen in [RFC2898]) MUST be 128-bits for the AES Session-key used for encryption and a 256-bit HMAC-SHA1 Session-key used for authentication (see Section 6.10 of [RFC4656]).

Each twamp-client container also holds a list of twamp-client-ctrl-connection, 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 twamp-client-ctrl-connection per TWAMP-Control (TCP) connection that is to be initiated from this device.

The configuration items for twamp-client-ctrl-connection are:

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

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

server-ip
: The IP address belonging to the remote Server device, which the TWAMP-Control connection will be initiated to. This item is mandatory.

server-tcp-port
: 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) as per [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. The default value is 862.

dscp
: The DSCP value to be placed in the TCP header of TWAMP-Control packets generated by this Control-Client. The default value is 0.

key-id
: The key-id value that is selected for this TWAMP-Control connection.

max-count
: 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 max-count value SHOULD be 32768.

The following twamp-client-ctrl-connection parameters are read-only:

client-tcp-port
: The source TCP port number used in the TWAMP-Control packets belonging to this control connection.

server-start-time
: 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.

ctrl-connection-state
: The TWAMP-Control connection state can be either active or idle.

selected-mode
: The TWAMP 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].

client-iv
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 twamp-client-ctrl-connection holds a list of twamp-session-request. twamp-session-request 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 (see Section 3.5 of [RFC5357]). The Control-Client is also responsible for scheduling and results collection for TWAMP-Test sessions, so twamp-session-request will also hold information related these actions (e.g. pm-index, repeat-interval).

There SHALL be one instance of twamp-session-request for each TWAMP-Test session that is to be negotiated by this TWAMP-Control connection via a Request-TW-Session/Accept-Session exchange.

The configuration items for twamp-session-request are:

test-session-name
A unique name for this test session to be used for identification of this TWAMP-Test session on the Control-Client.

sender-ip
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-Test (UDP) packets belonging to this test session.

This value will be used to populate the sender address field of the Request-TW-Session message. If not configured, the device SHALL choose its own source IP address.

sender-udp-port
The UDP port number that is to be used by the Session-Sender for this TWAMP-Test session. A value of zero indicates that the Control-Client SHALL auto-allocate a UDP port number for this TWAMP-Test session. The configured (or auto-allocated) value is advertized in the Sender Port field of the Request-TW-session message (see also Section 3.5 of [RFC5357]). Note that in the scenario where a device auto-allocates a UDP port number for a session, and the repeat parameter for that session indicates that it should be repeated, the device is free to auto-allocate a different UDP port number when it negotiates the next (repeated) iteration of this session.

reflector-ip
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. This item is mandatory.

reflector-udp-port
This parameter defines the UDP port number that will be used by the Session-Reflector for this TWAMP-Test session. This value will be placed in the Receiver Port field of the Request-TW-Session message. If this value is not set, the device SHALL use the same port number as defined in the server-tcp-port parameter of this twamp-session-request's parent twamp-client-ctrl-connection.

timeout The length of time (in seconds) that the Session-Reflector should continue to respond to packets belonging to this TWAMP-Test session after a Stop-Sessions TWAMP-Control message has been received ([RFC5357], Section 3.8). This value will be placed in the Timeout field of the Request-TW-Session message. The default value is 2 seconds.

padding-length
The number of bytes of padding that will be added to the TWAMP-Test (UDP) packets generated by the Session-Sender. This value will be placed in the Padding Length field of the Request-TW-Session message ([RFC4656], Section 3.5).

dscp The DSCP value to be placed in the UDP header of TWAMP-Test packets generated by the Session-Sender, and in the UDP

header of the TWAMP-Test response packets generated by the Session-Reflector for this test session. This value will be placed in the Type-P Descriptor field of the Request-TW-Session message ([RFC5357]).

start-time
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. The default value of 0 indicates that the session will be started as soon as the Start-Sessions message is received.

repeat and repeat-interval
These two values together are used to determine if the TWAMP-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 TWAMP-Control connection for this test session is restarted - and negotiates a new instance of this TWAMP-Test session. This may occur immediately after the test session completes (if the repeat-interval is set to 0). Otherwise, the Control-Client will wait for the number of minutes specified in the repeat-interval parameter before negotiating the new instance of this TWAMP-Test session. The default value of repeat is 0, indicating that once the session has completed, it will not be renegotiated and restarted.

pm-reg-list
A list of one or more Performance Metric Registry Index values (see [I-D.ietf-ippm-metric-registry], which communicate packet stream characteristics and one or more metrics to be measured. All members of the pm-reg-list MUST have the same stream characteristics, such that they combine to specify all metrics that shall be measured on a single stream.

pm-index
One or more Numerical index values of a Registered Metric in the Performance Metric Registry [I-D.ietf-ippm-metric-registry] comprise the pm-reg-list. Output statistics are specified in the corresponding Registry entry.

The following twamp-session-request parameters are read-only:

test-session-state
The TWAMP-Test session state can be either accepted or indicate the respective error code.

sid The SID allocated by the Server for this TWAMP-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 twamp-server container (see Figure 4) holds items that are related to the configuration of the TWAMP Server logical entity (recall Figure 1).

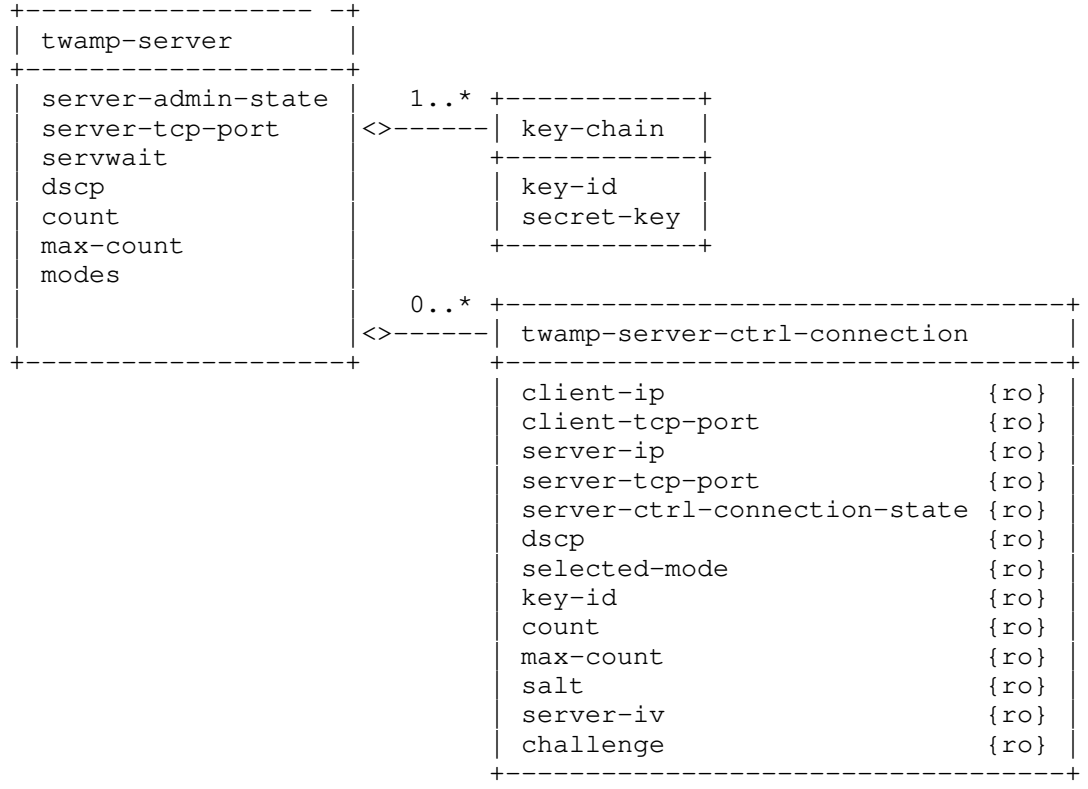


Figure 4: TWAMP Server UML class diagram

A device operating in the Server role cannot configure attributes on a per TWAMP-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 twamp-server container holds a list named key-chain 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. key-id tells the Server which shared-secret the Control-Client wishes to use for authentication or encryption.

Each incoming control connection that is active on the Server will be represented by an instance of a twamp-server-ctrl-connection object. All items in the twamp-server-ctrl-connection object are read-only, as we explain later in this section.

The twamp-server container items are as follows:

server-admin-state
: 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 sessions.

server-tcp-port
: This parameter defines the well known TCP port number that is used by TWAMP-Control. The Server will listen on this port number 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. The default value is 862.

servwait
: TWAMP-Control (TCP) session timeout, in seconds (([RFC5357], Section 3.1)).

dscp
: The DSCP value to be placed in the IP header of TWAMP-Control (TCP) packets generated by the Server. Section 3.1 of [RFC5357] specifies that the server SHOULD use the DSCP value from the Control-Client's TCP SYN. However, for practical purposes TWAMP will typically be implemented using a general purpose TCP stack provided by the underlying operating system, and such a stack may not provide this information to the user. Consequently, it is not always possible to

implement the behavior described in [RFC5357] in an OS-portable version of TWAMP. The default behavior if this item is not set is to use the DSCP value from the Control-Client's TCP SYN, as per Section 3.1 of [RFC5357].

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 the 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.

max-count
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 max-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.

There SHALL be one instance of twamp-server-ctrl-connection per incoming TWAMP-Control (TCP) connection that is received and active on the Server device. All items in the twamp-server-ctrl-connection are read-only. Each instance of twamp-server-ctrl-connection uses the following 4-tuple as its unique key: client-ip, client-tcp-port, server-ip, server-tcp-port.

The twamp-server-ctrl-connection container items are all read-only:

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

client-tcp-port
The source TCP port number used in the TWAMP-Control (TCP) packets belonging to this control connection.

server-ip

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

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

server-ctrl-connection-state
The Server TWAMP-Control connection state can be active or SERVWAIT.

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

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

key-id
The KeyID value that is in use by this TWAMP-Control connection. The Control-Client selects the key-id for the control connection.

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

max-count

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

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.

server-iv
The Server Initialization Vector (IV) is generated randomly by the Server.

challenge
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.

4.3. Session-Sender

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

The twamp-session-sender container includes an administrative parameter (session-sender-admin-state) that controls whether the device is allowed to initiate TWAMP test sessions.

There is one instance of twamp-sender-test-session for each TWAMP-Test session for which packets are being sent.

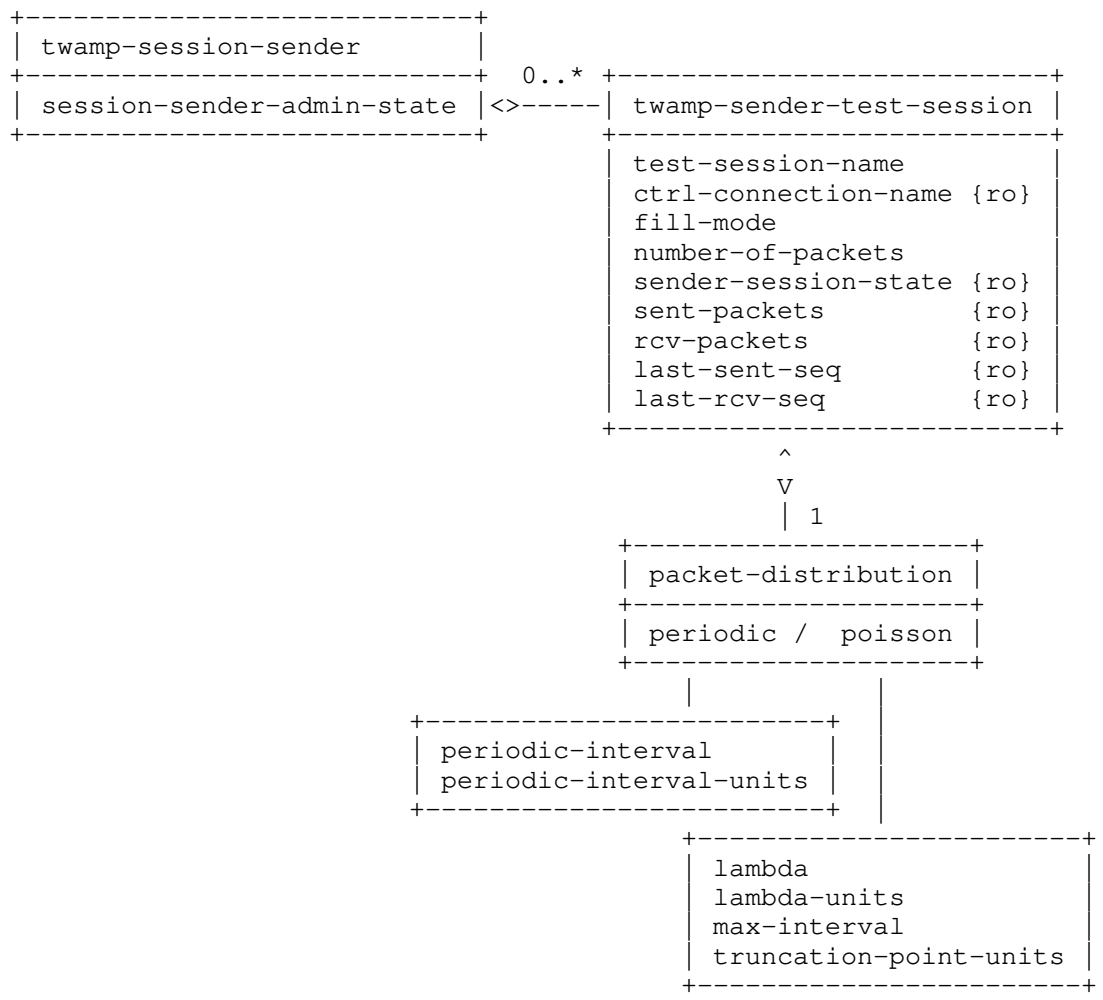


Figure 5: TWAMP Session-Sender UML class diagram

The twamp-sender-test-session container items are:

test-session-name
: A unique name for this TWAMP-Test session to be used for identifying this test session by the Session-Sender logical entity.

ctrl-connection-name
: The name of the parent TWAMP-Control connection that is responsible for negotiating this TWAMP-Test session.

fill-mode
: Indicates whether the padding added to the TWAMP-Test (UDP) packets will contain pseudo-random numbers, or whether it should consist of all zeroes, as per Section 4.2.1 of [RFC5357].

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

packet-distribution
: Defines whether TWAMP-Test (UDP) packets are to be transmitted with a fixed interval between them, or whether a Poisson distribution is to be used.

periodic-interval and periodic-interval-units
: If packet-distribution is set to periodic, these two values are used together to determine the period to wait between the first bits of TWAMP-Test (UDP) packet transmissions for this test session. periodic-interval-units is one of seconds, milliseconds, microseconds, nanoseconds; see [RFC3432].

lambda and lambda-units
: If packet-distribution is Poisson, the lambda parameter determines the corresponding average rate of packet transmission. lambda-units defines the units of lambda in reciprocal seconds; see [RFC3432].

max-interval
: If packet-distribution is Poisson, then this parameter keeps a stream active by setting a maximum time between packet transmissions.

truncation-point-units
: One of seconds, milliseconds, microseconds, nanoseconds.

The following twamp-sender-test-session parameters are read-only:

sender-session-state
: This read-only item can be either Active or Idle.

sent-packets
: The number of TWAMP-Test (UDP) packets belonging to this session that have been transmitted by the Session-Sender.

rcv-packets
: The number of TWAMP-Test (UDP) 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 sent-packets - rcv-packets.

last-sent-seq
The value in the sequence number field of the last TWAMP-Test (UDP) packet transmitted for this test session. Sequence numbers start from zero, so this should always be one less than the sent-packets value.

last-rcv-seq
The value in the sequence number field of the last TWAMP-Test (UDP) packet received for this test session. In the case of packet loss in the Session-Sender to Session-Reflector direction, this value minus the last-sent-seq will quantify the number of packets that were lost in the Session-Sender to Session-Reflector direction.

4.4. Session-Reflector

The twamp-session-reflector 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 TWAMP-Test session must be configured at the overall Session-Reflector level, and will then be applied to all incoming sessions.

The twamp-session-sender container includes an administrative parameter (session-reflector-admin-state) that controls whether the device is allowed to respond to incoming TWAMP test sessions. Each incoming TWAMP-Test session that is active on the Session-Reflector will be represented by an instance of a twamp-reflector-test-session object. All items in the twamp-reflector-test-session object are read-only.

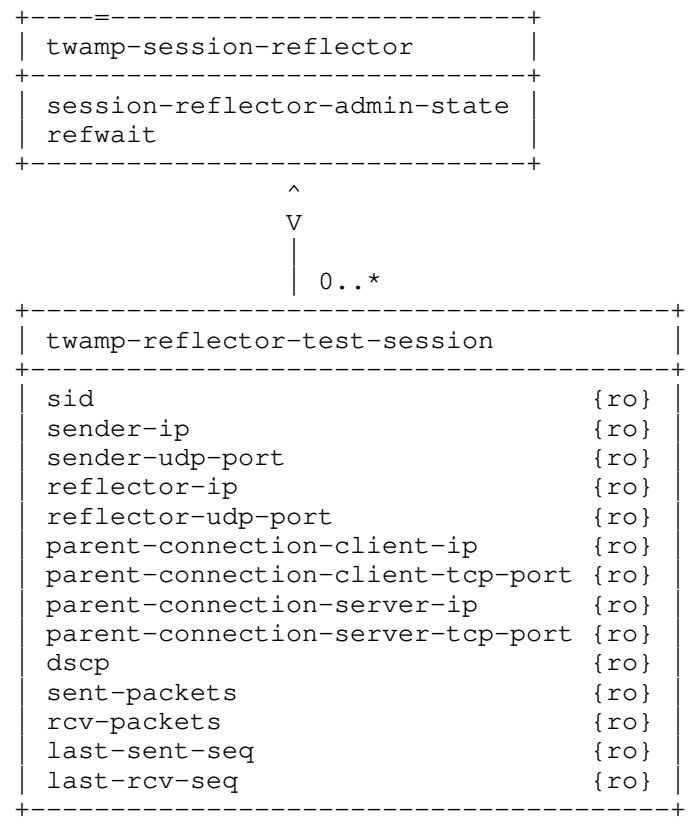


Figure 6: TWAMP Session-Reflector UML class diagram

The twamp-session-reflector 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 twamp-reflector-test-session are indexed by a session identifier (sid). This value 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 twamp-client:twamp-client-ctrl-connection:twamp-session-request:sid and obtain the SID (see Figure 3). The user may then use this SID value as an index to retrieve an individual twamp-session-reflector:twamp-reflector-test-session 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 twamp-reflector-test-session 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 TWAMP-Test session contains the following 4-tuple: {parent-connection-client-ip, parent-connection-client-tcp-port, parent-connection-server-ip, parent-connection-server-tcp-port}. This 4-tuple corresponds to the equivalent 4-tuple {client-ip, client-tcp-port, server-ip, server-tcp-port} in the twamp-server-ctrl-connection object. This 4-tuple allows the user to trace back from the TWAMP-Test session to the (parent) TWAMP-Control connection that negotiated this test session.

All data under twamp-reflector-test-session is read-only:

sid
: An auto-allocated identifier for this TWAMP-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.

sender-ip
: The IP address on the remote device, which is the source IP address used in the TWAMP-Test (UDP) packets belonging to this test session.

sender-udp-port
: The source UDP port used in the TWAMP-Test packets belonging to this test session.

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

reflector-udp-port
: The destination UDP port number used in the TWAMP-Test (UDP) test packets belonging to this test session.

parent-connection-client-ip
: The IP address on the Control-Client device, which is the source IP address used in the TWAMP-Control (TCP) packets belonging to the parent control connection that negotiated this test session.

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

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

parent-connection-server-tcp-port
: The destination TCP port number used in the TWAMP-Control (TCP) packets belonging to the parent control connection that negotiated this test session.

dscp
: The DSCP value present in the IP header of TWAMP-Test (UDP) packets belonging to this test session.

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

rcv-packets
: The number of TWAMP-Test (UDP) 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 sent-packets and rcv-packets values should always be identical.

last-sent-seq
: The value in the sequence number field of the last TWAMP-Test (UDP) response packet transmitted for this test session.

last-rcv-seq
: The value in the sequence number field of the last TWAMP-Test (UDP) packet received for this test session.

5. Data Model

This section formally specifies the TWAMP data model using YANG.

5.1. YANG Tree Diagram

This section presents a simplified graphical representation of the TWAMP data model using a YANG tree diagram. Readers should keep in mind that the limit of 72 characters per line forces us to introduce artificial line breaks in some tree diagram nodes.

```
module: ietf-twamp
   +--rw twamp
      +--rw twamp-client! {control-client}?
         |  +--rw client-admin-state              boolean
         |  +--rw mode-preference-chain* [priority]
         |  |  +--rw priority uint16
         |  |  +--rw mode? mode
         |  +--rw key-chain* [key-id]
         |  |  +--rw key-id        string
         |  |  +--rw secret-key?   string
         |  +--rw twamp-client-ctrl-connection* [ctrl-connection-name]
         |     +--rw ctrl-connection-name     string
         |     +--rw client-ip?               inet:ip-address
         |     +--rw server-ip                inet:ip-address
         |     +--rw server-tcp-port?         inet:port-number
         |     +--rw dscp?                    inet:dscp
         |     +--rw key-id?                  string
         |     +--rw max-count?               uint32
         |     +--ro client-tcp-port?         inet:port-number
         |     +--ro server-start-time?       uint64
         |     +--ro ctrl-connection-state?   ctrl-connection-state
         |     +--ro selected-mode?           mode
         |     +--ro token?                   binary
         |     +--ro client-iv?               binary
         |     +--rw twamp-session-request* [test-session-name]
         |        +--rw test-session-name     string
         |        +--rw sender-ip?            inet:ip-address
         |        +--rw sender-udp-port?      inet:port-number
         |        +--rw reflector-ip          inet:ip-address
         |        +--rw reflector-udp-port?   inet:port-number
         |        +--rw timeout?              uint64
         |        +--rw padding-length?       uint32
         |        +--rw dscp?                 inet:dscp
         |        +--rw start-time?           uint64
         |        +--rw repeat?               uint32
         |        +--rw repeat-interval?      uint32
         |        +--rw pm-reg-list* [pm-index]
```

```
|        |  +--rw pm-index    uint16
|        +--ro test-session-state?   test-session-state
|        +--ro sid?                  string
+--rw twamp-server! {server}?
|  +--rw server-admin-state              boolean
|  +--rw server-tcp-port?                inet:port-number
|  +--rw servwait?                       uint32
|  +--rw dscp?                           inet:dscp
|  +--rw count?                          uint32
|  +--rw max-count?                      uint32
|  +--rw modes?                          mode
|  +--rw key-chain* [key-id]
|  |  +--rw key-id        string
|  |  +--rw secret-key?   string
|  +--ro twamp-server-ctrl-connection* \
|           [client-ip client-tcp-port \
|            server-ip server-tcp-port]
|     +--ro client-ip                          inet:ip-address
|     +--ro client-tcp-port                    inet:port-number
|     +--ro server-ip                          inet:ip-address
|     +--ro server-tcp-port                    inet:port-number
|     +--ro server-ctrl-connection-state? \
|            server-ctrl-connection-state
|     +--ro dscp?                              inet:dscp
|     +--ro selected-mode?                     mode
|     +--ro key-id?                            string
|     +--ro count?                             uint32
|     +--ro max-count?                         uint32
|     +--ro salt?                              binary
|     +--ro server-iv?                         binary
|     +--ro challenge?                         binary
+--rw twamp-session-sender {session-sender}?
|  +--rw session-sender-admin-state         boolean
|  +--rw twamp-sender-test-session* [test-session-name]
|     +--rw test-session-name          string
|     +--ro ctrl-connection-name?      string
|     +--rw fill-mode?                 fill-mode
|     +--rw number-of-packets?         uint32
|     +--rw (packet-distribution)?
|     |  +--:(periodic)
|     |  |  +--rw periodic-interval?         uint32
|     |  |  +--rw periodic-interval-units?   units
|     |  +--:(poisson)
|     |     +--rw lambda?                    uint32
|     |     +--rw lambda-units?              uint32
|     |     +--rw max-interval?              uint32
|     |     +--rw truncation-point-units?    units
|     +--ro sender-session-state?      sender-session-state
```

```
          |     +--ro sent-packets?              uint32
          |     +--ro rcv-packets?               uint32
          |     +--ro last-sent-seq?             uint32
          |     +--ro last-rcv-seq?              uint32
          +--rw twamp-session-reflector {session-reflector}?
             +--rw session-reflector-admin-state        boolean
             +--rw refwait?                           uint32
             +--ro twamp-reflector-test-session* \
                   [sender-ip sender-udp-port \
                    reflector-ip reflector-udp-port]
                +--ro sid?                                  string
                +--ro sender-ip                             inet:ip-address
                +--ro sender-udp-port                       inet:port-number
                +--ro reflector-ip                          inet:ip-address
                +--ro reflector-udp-port                    inet:port-number
                +--ro parent-connection-client-ip?          inet:ip-address
                +--ro parent-connection-client-tcp-port?    inet:port-number
                +--ro parent-connection-server-ip?          inet:ip-address
                +--ro parent-connection-server-tcp-port?    inet:port-number
                +--ro dscp?                                 inet:dscp
                +--ro sent-packets?                         uint32
                +--ro rcv-packets?                          uint32
                +--ro last-sent-seq?                        uint32
                +--ro last-rcv-seq?                         uint32
```

5.2. YANG Module

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

```
<CODE BEGINS> file "ietf-twamp@2015-10-19.yang"
module ietf-twamp {
  namespace "urn:ietf:params:xml:ns:yang:ietf-twamp";
   //namespace need to be assigned by IANA
  prefix "ietf-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 "2015-10-19" {
    description "01 version. RFC5357, RFC5618, RFC5938 and RFC6038
```

```
       is covered. draft-ietf-ippm-metric-registry is also considered";

     reference "draft-cmzrjp-ippm-twamp-yang";
     }

     feature control-client {
       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 session-sender {
       description "This feature relates to the device functions as
       the TWAMP Session-Sender.";
     }

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

     typedef ctrl-connection-state {
       type enumeration {
         enum active {
           description "Control session is active.";
         }
         enum idle {
           description "Control session is idle.";
         }
       }
       description "Control connection state";
     }

     typedef mode {
       type bits {
         bit unauthenticated {
           position "0";
           description "Unauthenticated";
         }
         bit authenticated {
           position "1";
           description "Authenticated";
         }
         bit encrypted {
```

```
          position "2";
          description "Encrypted";
        }
        bit unauth-test-encrpyt-control {
          position "3";
          description "Mixed Security Mode per RFC 5618. Test
          protocol security mode in Unauthenticated mode,
          Control protocol in Encrypted mode.";
        }
        bit individual-session-control {
          position "4";
          description "Individual session control per RFC5938.";
        }
        bit reflect-octets {
          position "5";
          description "Reflect octets capability per RFC6038.";
        }
        bit symmetrical-size {
          position "6";
          description "Symmetrical size per RFC6038.";
        }
      }
      description "Authentication mode bit mask";
    }

    typedef test-session-state {
      type enumeration {
        enum ok {
          value 0;
          description "Test session is accepted.";
        }
        enum failed {
          value 1;
          description "Failure, reason unspecified (catch-all).";
        }
        enum internal-error {
          value 2;
          description "Internal error.";
        }
        enum not-supported {
          value 3;
          description "Some aspect of request is not supported.";
        }
        enum permanent-resource-limit {
          value 4;
          description "Cannot perform request due to
          permanent resource limitations.";
        }
```

```
        enum temp-resource-limit {
          value 5;
          description "Cannot perform request due to
          temporary resource limitations.";
        }
      }
      description "Test session state";
    }

    typedef server-ctrl-connection-state {
      type enumeration {
        enum "active" {
          description "Active";
        }
        enum "servwait" {
          description "Servwait";
        }
      }
      description "Server control connection state";
    }

    typedef fill-mode {
      type enumeration {
        enum zero {
          description "Zero";
        }
        enum random {
          description "Random";
        }
      }
      description "Indicates whether the padding added to the
      UDP test packets will contain pseudo-random numbers, or
      whether it should consist of all zeroes.";
    }

    typedef units  {
      type enumeration {
        enum seconds {
          description "Seconds";
        }
        enum milliseconds {
          description "Milliseconds";
        }
        enum microseconds {
          description "Microseconds";
        }
        enum nanoseconds {
          description "Nanoseconds";
```

```
        }
      }
      description "Time units";
    }

    typedef sender-session-state {
      type enumeration {
        enum setup {
          description "Test session is active.";
        }
        enum failure {
          description "Test session is idle.";
        }
      }
      description "Sender session state.";
    }

    grouping maintenance-statistics {
      description "Maintenance statistics grouping";
      leaf sent-packets {
        type uint32;
        config "false";
        description "Packets sent";
      }
      leaf rcv-packets {
        type uint32;
        config "false";
        description "Packets received";
      }
      leaf last-sent-seq {
        type uint32;
        config "false";
        description "Last sent sequence number";
      }
      leaf last-rcv-seq {
        type uint32;
        config "false";
        description "Last received sequence number";
      }
    }

    container twamp {
      description "Top level container";
      container twamp-client {
        if-feature control-client;
        presence "twamp-client";
        description "Twamp client container";
        leaf client-admin-state {
```

```
          type boolean;
          mandatory "true";
          description "Indicates whether this device is allowed to run
          TWAMP to initiate control sessions";
        }

        list mode-preference-chain {
          key "priority";
          unique "mode";
          leaf priority {
            type uint16;
            description "priority";
          }
          leaf mode {
            type mode;
            description "Authentication mode bit mask";
          }
          description "Authentication mode preference";
        }

        list key-chain {
          key "key-id";
          leaf key-id {
            type string {
              length "1..80";
            }
            description "Key ID";
          }
          leaf secret-key {
            type string;
            description "Secret key";
          }
          description "Key chain";
        }

        list twamp-client-ctrl-connection {
          key "ctrl-connection-name";
          description "Twamp client control connections";
          leaf ctrl-connection-name {
            type string;
            description "A unique name used as a key to identify this
              individual TWAMP control connection on the
              Control-Client device.";
          }
          leaf client-ip {
            type inet:ip-address;
            description "Client IP address";
          }
```

```
          leaf server-ip {
            type inet:ip-address;
            mandatory "true";
            description "Server IP address";
          }
          leaf server-tcp-port {
            type inet:port-number;
            default "862";
            description "Server tcp port";
          }
          leaf dscp{
            type inet:dscp;
            default "0";
            description "The DSCP value to be placed in the IP header
              of the TWAMP TCP Control packets generated
              by the Control-Client";
          }
          leaf key-id {
            type string {
              length "1..80";
            }
            description "Key ID";
          }
          leaf max-count {
            type uint32 {
              range 1024..4294967295;
            }
            default 32768;
            description "Max count value.";
          }
          leaf client-tcp-port {
            type inet:port-number;
            config "false";
            description "Client TCP port";
          }
          leaf server-start-time {
            type uint64;
            config "false";
            description "The Start-Time advertized by the Server in
            the Server-Start message";
          }
          leaf ctrl-connection-state {
            type ctrl-connection-state;
            config "false";
            description "Control connection state";
          }
          leaf selected-mode {
            type mode;
```

```
            config "false";
            description "The TWAMP mode that the Control-Client has
            chosen for this control connection as set in the Mode
            field of the Set-Up-Response message";
          }
          leaf token {
              type binary {
              length "64";
            }
            config "false";
            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 client-iv{
            type binary {
              length "16";
            }
            config "false";
            description "16 octets, Client-IV is generated randomly
              by the Control-Client.";
          }

          list twamp-session-request {
            key "test-session-name";
            description "Twamp session requests";
            leaf test-session-name {
              type string;
              description "A unique name for this test session to be
              used as a key for this test session on the
              Control-Client.";
            }
            leaf sender-ip {
              type inet:ip-address;
              description "Sender IP address";
            }
            leaf sender-udp-port {
              type inet:port-number;
              description "Sender UDP port";
            }
            leaf reflector-ip {
              type inet:ip-address;
              mandatory "true";
              description "Reflector IP address.";
            }
            leaf reflector-udp-port {
              type inet:port-number;
```

```
            description "Reflector UDP port. If this value is not
            set, the device shall use the same port number as
            defined in the server-tcp-port parameter of this
            twamp-session-request's
            parent client-control-connection.";
          }
          leaf timeout {
            type uint64;
            default "2";
            description "The time (in seconds)Session-Reflector MUST
            wait after receiving a Stop-Session message.";
          }
          leaf padding-length {
            type uint32{
              range "64..4096";
            }
            description "The number of bytes of padding that should
                be added to the UDP test packets generated by the
                sender. Jumbo sized packets supported.";
          }
          leaf dscp {
            type inet:dscp;
            description "The DSCP value to be placed in the UDP
            header of TWAMP-Test packets generated by the
            Session-Sender, and in the UDP header of the TWAMP-Test
            response packets generated by the Session-Reflector
            for this test session.";
          }
          leaf start-time {
            type uint64;
            default "0";
            description "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. The default value of 0
            indicates that the session will be started as soon
            as the Start-Sessions message is received.";
          }
          leaf repeat {
            type uint32;
            default "0";
            description "Determines if the test session is to be
            run repeatedly. The default value of repeat is 0,
            indicating that once the session has completed, it
            will not be renegotiated and restarted";
          }
          leaf repeat-interval  {
            when "../repeat!='0'" {
```

```
                description "When repeat is not 0, the test is to be
                repeated";
              }
              type uint32;
              description "Repeat interval (in minutes)";
            }

            list pm-reg-list {
              key "pm-index";
              leaf pm-index {
                type uint16;
                description "One or more Numerical index values of a
                Registered Metric in the Performance Metric Registry";
              }
              description "A list of one or more pm-index values,
              which communicate packet stream characteristics and one
              or more metrics to be measured.";
            }
            leaf test-session-state {
              type test-session-state;
              config "false";
              description "Test session state";
            }
            leaf sid{
              type string;
              config "false";
              description "The SID allocated by the Server for
              this test session";
            }
          }
        }
      }

      container twamp-server{
        if-feature server;
        presence "twamp-server";
        description "Twamp sever container";
        leaf server-admin-state{
          type boolean;
          mandatory "true";
          description "Indicates whether this device is allowed to run
            TWAMP to respond to control sessions";
        }
        leaf server-tcp-port {
          type inet:port-number;
          default "862";
          description "This parameter defines the well known TCP port
          number that is used by TWAMP.";
```

```
        }
        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
          TCP TWAMP-Control packets generated by the Server";
        }
        leaf count {
          type uint32 {
            range 1024..4294967295;
          }
          description "Parameter used in deriving a key from a
          shared secret ";
        }
        leaf max-count {
          type uint32 {
            range 1024..4294967295;
          }
          default 32768;
          description "Max count value.";
        }
        leaf modes {
          type mode;
          description "The bit mask of TWAMP Modes this Server
          instance is willing to support.";
        }

        list key-chain {
          key "key-id";
          leaf key-id {
            type string {
              length "1..80";
            }
            description "Key IDs.";
          }
          leaf secret-key {
            type string;
            description "Secret keys.";
          }
          description "KeyIDs with the respective secret keys.";
        }
```

```
        list twamp-server-ctrl-connection {
          key "client-ip client-tcp-port server-ip server-tcp-port";
          config "false";
          description "Twamp server control connections";
          leaf client-ip {
            type inet:ip-address;
            description "Client IP address";
          }
          leaf client-tcp-port {
            type inet:port-number;
            description "Client TCP port";
          }
          leaf server-ip {
            type inet:ip-address;
            description "Server IP address";
          }
          leaf server-tcp-port {
            type inet:port-number;
            description "Server TCP port";
          }
          leaf server-ctrl-connection-state {
            type server-ctrl-connection-state;
            description "Server control connection state";
          }
          leaf dscp {
            type inet:dscp;
            description "The DSCP value used in the IP 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 twamp-server:dscp under the twamp-server.
            However, in the event that the user re-configures
            twamp-server: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 selected-mode {
            type mode;
            description "The mode that was chosen for this control
            connection as set in the Mode field of the
            Set-Up-Response message.";
          }
          leaf key-id {
            type string {
              length "1..80";
            }
            description "The key-id value that is in use by 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 twamp-server. However, in the event that
            the user re-configured twamp-server:count after this
            control connection is already in progress, this read-only
            value will show the different count that is in use for
            this control connection.";
          }
          leaf max-count {
            type uint32 {
              range 1024..4294967295;
            }
            description "The max-count value that is in use by this
            control connection. This will usually be the same value
            as is configured under twamp-server. However, in the
            event that the user re-configured twamp-server:max-count
            after this control connection is already in progress,
            this read-only value will show the different max-count
            that is in use for this control connection.";
          }
          leaf salt{
            type binary {
              length "16";
            }
            description "Salt MUST be generated pseudo-randomly";
          }
          leaf server-iv {
            type binary {
              length "16";
            }
            description "16 octets, Server-IV is generated randomly
            by the Control-Client.";
          }
          leaf challenge {
            type binary {
              length "16";
            }
            description "Challenge is a random sequence of octets
            generated by the Server";
          }
        }
      }

      container twamp-session-sender{
```

```
        if-feature session-sender;
        description "Twamp session sender container";
        leaf session-sender-admin-state {
          type boolean;
          mandatory "true";
          description "Indicates whether this device is allowed to run
          TWAMP to initiate test sessions";
        }
        list twamp-sender-test-session{
          key "test-session-name";
          description "Twamp sender test sessions";
          leaf test-session-name {
            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 ctrl-connection-name {
            type string;
            config "false";
            description "The name of the parent control connection
            that is responsible for negotiating this test session.";
          }
          leaf fill-mode {
            type fill-mode;
            default zero;
            description "Indicates whether the padding added to the
            UDP test packets will contain pseudo-random numbers, or
            whether it should consist of all zeroes.";
          }
          leaf number-of-packets {
            type uint32;
            description "The overall number of UDP test packets to be
              transmitted by the sender for this test session.";
          }
          choice packet-distribution {
            description "Packet distributions, poisson or periodic";
            case periodic {
              leaf periodic-interval {
                type uint32;
                description "Periodic interval";
              }
              leaf periodic-interval-units  {
                type units;
                description "Periodic interval units";
              }
            }
            case poisson {
```

```
            leaf lambda{
              type uint32;
              description "The average rate of
              packet transmission.";
            }
            leaf lambda-units{
              type uint32;
              description "Lambda units.";
            }
            leaf max-interval{
              type uint32;
              description "maximum time between packet
              transmissions.";
            }
            leaf truncation-point-units{
              type units;
              description "Truncation point units";
            }
          }
        }
        leaf sender-session-state {
          type sender-session-state;
          config "false";
          description "Sender session state.";
        }
        uses maintenance-statistics;
      }
    }

    container twamp-session-reflector {
      if-feature session-reflector;
      description "Twamp session reflector container";
      leaf session-reflector-admin-state {
        type boolean;
        mandatory "true";
        description "Indicates whether this device is allowed to run
        TWAMP to respond to test sessions";
      }
      leaf refwait {
        type uint32 {
          range 1..604800;
        }
        default 900;
        description "REFWAIT (TWAMP test session timeout),
          the default value is 900";
      }

      list twamp-reflector-test-session {
```

```
          key "sender-ip sender-udp-port reflector-ip
              reflector-udp-port";
          config "false";
          description "Twamp reflector test sessions";
          leaf sid{
            type string;
            description "An auto-allocated identifier for this test
            session, that is unique within the context of this
            Server/Session-Reflector device only. ";
          }
          leaf sender-ip {
            type inet:ip-address;
            description "Sender IP address.";
          }
          leaf sender-udp-port {
            type inet:port-number;
            description "Sender UDP port.";
          }
          leaf reflector-ip {
            type inet:ip-address;
            description "Reflector IP address.";
          }
          leaf reflector-udp-port {
            type inet:port-number;
            description "Reflector UDP port.";
          }
          leaf parent-connection-client-ip {
            type inet:ip-address;
            description "Parent connction client IP address.";
          }
          leaf parent-connection-client-tcp-port {
            type inet:port-number;
            description "Parent connection client TCP port.";
          }
          leaf parent-connection-server-ip {
            type inet:ip-address;
            description "Parent connection server IP address.";
          }
          leaf parent-connection-server-tcp-port {
            type inet:port-number;
            description "Parent connection server TCP port";
          }
          leaf dscp {
            type inet:dscp;
            description "The DSCP value present in the IP header of
            TWAMP UDP test packets belonging to this test session.";
          }
          uses maintenance-statistics;
```

```
        }
      }
    }
  }
```

<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 client-admin-state enabled. In a real implementation following Figure 2 this would permit the initiation of TWAMP-Control connections and TWAMP-Test sessions.

```
<twamp xmlns="urn:ietf:params:xml:ns:yang:ietf-twamp">
   <twamp-client>
      <client-admin-state>True</client-admin-state>
   </twamp-client>
</twamp>
```

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

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

      <twamp-client-ctrl-connection>
         <ctrl-connection-name>RouterA</ctrl-connection-name>
         <client-ip>203.0.113.1</client-ip>
         <server-ip>203.0.113.2</server-ip>
         <twamp-session-request>
            <test-session-name>Test1</test-session-name>
            <sender-ip>10.1.1.1</sender-ip>
            <sender-udp-port>4000</sender-udp-port>
            <reflector-ip>10.1.1.2</reflector-ip>
            <reflector-udp-port>5000</reflector-udp-port>
            <start-time>0</start-time>
         </twamp-session-request>
         <twamp-session-request>
            <test-session-name>Test2</test-session-name>
            <sender-ip>203.0.113.1</sender-ip>
            <sender-udp-port>4001</sender-udp-port>
            <reflector-ip>203.0.113.2</reflector-ip>
            <reflector-udp-port>5001</reflector-udp-port>
            <start-time>0</start-time>
         </twamp-session-request>
      </twamp-client-ctrl-connection>

      <twamp-client-ctrl-connection>
         <ctrl-connection-name>RouterB</ctrl-connection-name>
         <client-ip>203.0.113.1</client-ip>
         <server-ip>203.0.113.3</server-ip>
      </twamp-client-ctrl-connection>

   </twamp-client>
</twamp>
```

6.2. Server

This configuration example shows a Server with server-admin-state enabled, which permits a device following Figure 2 to respond to TWAMP-Control connections and TWAMP-Test sessions.

```
<twamp xmlns="urn:ietf:params:xml:ns:yang:ietf-twamp">
   <twamp-server>
      <server-admin-state>True</server-admin-state>
   </twamp-server>
</twamp>
```

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

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

      <twamp-server-ctrl-connection>
         <client-ip>203.0.113.1</client-ip>
         <client-tcp-port>16341</client-tcp-port>
         <server-ip>203.0.113.2</server-ip>
         <server-tcp-port>862</server-tcp-port>
         <server-ctrl-connection-state>
            active
         </server-ctrl-connection-state>
      </twamp-server-ctrl-connection>

   </twamp-server>
</twamp>
```

6.3. Session-Sender

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

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

        <twamp-sender-test-session>
           <test-session-name>Test1</test-session-name>// read-only
           <ctrl-connection-name>RouterA</ctrl-connection-name>
           <number-of-packets>900</number-of-packets>
           <packet-distribution>
              <periodic-interval>1</periodic-interval>
              <periodic-interval-units>seconds</periodic-interval-units>
           </packet-distribution>
        </twamp-sender-test-session>

        <twamp-sender-test-session>
           <test-session-name>Test2</test-session-name>
           <ctrl-connection-name>
             RouterA
           </ctrl-connection-name> // read-only
           <number-of-packets>900</number-of-packets>
           <packet-distribution>
              <lambda>1</lambda>
              <lambda-units>1</lambda-units>
              <max-interval>2</max-interval>
              <truncation-point-units>seconds</truncation-point-units>
          </packet-distribution>
        </twamp-sender-test-session>

     </twamp-session-sender>
  </twamp>
```

6.4. Session-Reflector

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

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

      <twamp-reflector-test-session>
        <sid>1232</sid>
        <sender-ip>10.1.1.1</sender-ip>
        <reflector-ip>10.1.1.2</reflector-ip>
        <sender-udp-port>4000</sender-udp-port>
        <reflector-udp-port>5000</reflector-udp-port>
        <parent-connection-client-ip>
           203.0.113.1
         </parent-connection-client-ip>
        <parent-connection-client-tcp-port>
```

```
              16341
             </parent-connection-client-tcp-port>
            <parent-connection-server-ip>
              203.0.113.2
             </parent-connection-server-ip>
            <parent-connection-server-tcp-port>
              862
             </parent-connection-server-tcp-port>
            <sent-packets>2</sent-packets>
            <rcv-packets>2</rcv-packets>
            <last-sent-seq>1</last-sent-seq>
            <last-rcv-seq>1</last-rcv-seq>
          </twamp-reflector-test-session>

          <twamp-reflector-test-session>
            <sid>178943</sid>
            <sender-ip>203.0.113.1</sender-ip>
            <reflector-ip>192.68.0.2</reflector-ip>
            <sender-udp-port>4001</sender-udp-port>
            <parent-connection-client-ip>
              203.0.113.1
             </parent-connection-client-ip>
            <parent-connection-client-tcp-port>
              16341
             </parent-connection-client-tcp-port>
            <parent-connection-server-ip>
              203.0.113.2
             </parent-connection-server-ip>
            <parent-connection-server-tcp-port>
              862
            </parent-connection-server-tcp-port>
            <reflector-udp-port>5001</reflector-udp-port>
            <sent-packets>21</sent-packets>
            <rcv-packets>21</rcv-packets>
            <last-sent-seq>20</last-sent-seq>
            <last-rcv-seq>20</last-rcv-seq>
          </twamp-reflector-test-session>

       </twamp-session-reflector>
    </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

We thank Gregory Mirsky, Kevin D'Souza, and Robert Sherman for their thorough and constructive reviews, comments and text suggestions.

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, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3432] Raisanen, V., Grotefeld, G., and A. Morton, "Network performance measurement with periodic streams", RFC 3432, DOI 10.17487/RFC3432, November 2002, <http://www.rfc-editor.org/info/rfc3432>.

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

[RFC4656] Shalunov, S., Teitelbaum, B., Karp, A., Boote, J., and M. Zekauskas, "A One-way Active Measurement Protocol (OWAMP)", RFC 4656, DOI 10.17487/RFC4656, September 2006, <http://www.rfc-editor.org/info/rfc4656>.

[RFC5357] Hedayat, K., Krzanowski, R., Morton, A., Yum, K., and J. Babiarz, "A Two-Way Active Measurement Protocol (TWAMP)", RFC 5357, DOI 10.17487/RFC5357, October 2008, <http://www.rfc-editor.org/info/rfc5357>.

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

[RFC6038] Morton, A. and L. Ciavattone, "Two-Way Active Measurement Protocol (TWAMP) Reflect Octets and Symmetrical Size Features", RFC 6038, DOI 10.17487/RFC6038, October 2010, <http://www.rfc-editor.org/info/rfc6038>.

10.2. Informative References

[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-05 (work in progress), October 2015.

[I-D.ietf-netconf-restconf] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-08 (work in progress), October 2015.

[I-D.unify-nfvrg-challenges] Szabo, R., Csaszar, A., Pentikousis, K., Kind, M., Daino, D., Qiang, Z., and H. Woesner, "Unifying Carrier and Cloud Networks: Problem Statement and Challenges", draft-unify-nfvrg-challenges-02 (work in progress), July 2015.

[I-D.unify-nfvrg-devops]
Meirosu, C., Manzalini, A., Steinert, R., Marchetto, G., Papafili, I., Pentikousis, K., and S. Wright, "DevOps for Software-Defined Telecom Infrastructures", draft-unify-nfvrg-devops-03 (work in progress), October 2015.

[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.

[RFC2898] Kaliski, B., "PKCS #5: Password-Based Cryptography Specification Version 2.0", RFC 2898, DOI 10.17487/RFC2898, September 2000, <http://www.rfc-editor.org/info/rfc2898>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <http://www.rfc-editor.org/info/rfc4086>.

[RFC5618] Morton, A. and K. Hedayat, "Mixed Security Mode for the Two-Way Active Measurement Protocol (TWAMP)", RFC 5618, DOI 10.17487/RFC5618, August 2009, <http://www.rfc-editor.org/info/rfc5618>.

[RFC5938] Morton, A. and M. Chiba, "Individual Session Control Feature for the Two-Way Active Measurement Protocol (TWAMP)", RFC 5938, DOI 10.17487/RFC5938, August 2010, <http://www.rfc-editor.org/info/rfc5938>.

[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, <http://www.rfc-editor.org/info/rfc6241>.

[RFC7426] Haleplidis, E., Ed., Pentikousis, K., Ed., Denazis, S., Hadi Salim, J., Meyer, D., and O. Koufopavlou, "Software-Defined Networking (SDN): Layers and Architecture Terminology", RFC 7426, DOI 10.17487/RFC7426, January 2015, <http://www.rfc-editor.org/info/rfc7426>.

Appendix A. Detailed Data Model Examples

This appendix extends 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">
      <twamp-client>

         <client-admin-state>True</client-admin-state>

         <mode-preference-chain>
            <priority>0</priority>
            <mode>0x00000002</mode>
         </mode-preference-chain>
         <mode-preference-chain>
            <priority>1</priority>
            <mode>0x00000001</mode>
         </mode-preference-chain>

         <keychain>
            <keyid>KeyClient1ToRouterA</keyid>
            <secret-key>secret1</secret-key>
         </keychain>
         <keychain>
            <keyid>KeyForRouterB</keyid>
            <secret-key>secret2</secret-key>
         </keychain>

         <twamp-client-ctrl-connection>
            <ctrl-connection-name>RouterA</ctrl-connection-name>
            <client-ip>203.0.113.1</client-ip>
            <server-ip>203.0.113.2</server-ip>
            <dscp>32</dscp>
            <key-id>KeyClient1ToRouterA</key-id>
            <twamp-session-request>
               <test-session-name>Test1</test-session-name>
               <sender-ip>10.1.1.1</sender-ip>
               <sender-udp-port>4000</sender-udp-port>
               <reflector-ip>10.1.1.2</reflector-ip>
               <reflector-udp-port>5000</reflector-udp-port>
               <padding-length>0</padding-length>
               <start-time>0</start-time>
               <test-session-state>ok</test-session-state>
               <sid>1232</sid>
            </twamp-session-request>
            <twamp-session-request>
               <test-session-name>Test2</test-session-name>
               <sender-ip>203.0.113.1</sender-ip>
               <sender-udp-port>4001</sender-udp-port>
               <reflector-ip>203.0.113.2</reflector-ip>
               <reflector-udp-port>5001</reflector-udp-port>
```

```
               <paddingLenth>32</paddingLenth>
               <start-time>0</start-time>
               <test-session-state>ok</test-session-state>
               <sid>178943</sid>
            </twamp-session-request>
         </twamp-client-ctrl-connection>

      </twamp-client>
   </twamp>
```

A.2. Server

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

         <server-admin-state>True</server-admin-state>
         <servwait>1800</servwait>
         <dscp>32</dscp>
         <modes>0x00000003</modes>
         <count>256</count>

         <keychain>
            <keyid>KeyClient1ToRouterA</keyid>
            <secret-key>secret1</secret-key>
         </keychain>
         <keychain>
            <keyid>KeyClient10ToRouterA</keyid>
            <secret-key>secret10</secret-key>
         </keychain>

         <twamp-server-ctrl-connection>
            <client-ip>203.0.113.1</client-ip>
            <client-tcp-port>16341</client-tcp-port>
            <server-ip>203.0.113.2</server-ip>
            <server-tcp-port>862</server-tcp-port>
            <server-ctrl-connection-state>
               active
             </server-ctrl-connection-state>
            <dscp>32</dscp>
            <selected-mode>0x00000002</selected-mode>
            <key-id>KeyClient1ToRouterA</key-id>
            <count>1024</count>
         </twamp-server-ctrl-connection>

      </twamp-server>
   </twamp>
```

A.3. Session-Sender

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

        <twamp-sender-test-session>
           <test-session-name>Test1</test-session-name> // read-only
           <ctrl-connection-name>RouterA</ctrl-connection-name>
           <dscp>32</dscp>
           <fill-mode>zero</fill-mode>
           <number-of-packets>900</number-of-packets>
           <packet-distribution>
              <periodic-interval>1</periodic-interval>
              <periodic-interval-units>seconds</periodic-interval-units>
           </packet-distribution>
           <sender-session-state>Active</sender-session-state>
           <sent-packets>2</sent-packets>
           <rcv-packets>2</rcv-packets>
           <last-sent-seq>1</last-sent-seq>
           <last-rcv-seq>1</last-rcv-seq>
        </twamp-sender-test-session>

        <twamp-sender-test-session>
           <test-session-name>Test2</test-session-name>
           <ctrl-connection-name>
             RouterA
           </ctrl-connection-name> // read-only
           <dscp>32</dscp>
           <fill-mode>random</fill-mode>
           <number-of-packets>900</number-of-packets>
           <packet-distribution>
             <lambda>1</lambda>
             <lambda-units>1</lambda-units>
             <max-interval>2</max-interval>
             <truncation-point-units>seconds</truncation-point-units>
           </packet-distribution>
           <sender-session-state>Active</sender-session-state>
           <sent-packets>21</sent-packets>
           <rcv-packets>21</rcv-packets>
           <last-sent-seq>20</last-sent-seq>
           <last-rcv-seq>20</last-rcv-seq>
        </twamp-sender-test-session>

     </twamp-session-sender>
  </twamp>
```

A.4. Session-Reflector

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

         <twamp-reflector-test-session>
            <sid>1232</sid>
            <sender-ip>10.1.1.1</sender-ip>
            <reflector-ip>10.1.1.2</reflector-ip>
            <sender-udp-port>4000</sender-udp-port>
            <reflector-udp-port>5000</reflector-udp-port>
            <parent-connection-client-ip>
               203.0.113.1
            </parent-connection-client-ip>
            <parent-connection-client-tcp-port>
               16341
            </parent-connection-client-tcp-port>
            <parent-connection-server-ip>
               203.0.113.2
            </parent-connection-server-ip>
            <parent-connection-server-tcp-port>
               862
            </parent-connection-server-tcp-port>
            <dscp>32</dscp>
            <sent-packets>2</sent-packets>
            <rcv-packets>2</rcv-packets>
            <last-sent-seq>1</last-sent-seq>
            <last-rcv-seq>1</last-rcv-seq>
         </twamp-reflector-test-session>

         <twamp-reflector-test-session>
            <sid>178943</sid>
            <sender-ip>203.0.113.1</sender-ip>
            <reflector-ip>192.68.0.2</reflector-ip>
            <sender-udp-port>4001</sender-udp-port>
            <parent-connection-client-ip>
               203.0.113.1
            </parent-connection-client-ip>
            <parent-connection-client-tcp-port>
               16341
            </parent-connection-client-tcp-port>
            <parent-connection-server-ip>
               203.0.113.2
            </parent-connection-server-ip>
            <parent-connection-server-tcp-port>
               862
            </parent-connection-server-tcp-port>
            <reflector-udp-port>5001</reflector-udp-port>
```

```
            <dscp>32</dscp>
            <sent-packets>21</sent-packets>
            <rcv-packets>21</rcv-packets>
            <last-sent-seq>20</last-sent-seq>
            <last-rcv-seq>20</last-rcv-seq>
          </twamp-reflector-test-session>

      </twamp-session-reflector>
   </twamp>
```

Appendix B. TWAMP Operational Commands

This document is targeted at configuration details for TWAMP. Operational actions such as how TWAMP sessions are started/stopped, how results are retrieved, or stored results are cleared, and so on, are not addressed by this configuration model and are out of scope of this document.

TWAMP operational commands could be performed programmatically or manually, e.g. using a command-line interface (CLI). With respect to programmability, YANG can be used to define NETCONF Remote Procedure Calls (RPC), therefore it would be possible to define RPC operations for actions such as starting or stopping control or test sessions or groups of sessions; retrieving results; clearing stored results, and so on.

However, [RFC5357] does not attempt to describe such operational actions, and it is likely that different TWAMP implementations could support different sets of operational commands, with different restrictions. Therefore, this document considers it the responsibility of the individual implementation to define its corresponding TWAMP operational commands data model.

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

INTERNET-DRAFT N. Elkins
Inside Products
R. Hamilton
Chemical Abstracts Service
M. Ackermann
BCBS Michigan
Intended Status: Proposed Standard
Expires: December 28, 2017
June 26, 2017

IPv6 Performance and Diagnostic Metrics (PDM) Destination Option
draft-ietf-ippm-6man-pdm-option-13

Abstract

To assess performance problems, this document describes optional headers embedded in each packet that provide sequence numbers and timing information as a basis for measurements. Such measurements may be interpreted in real-time or after the fact. This document specifies the Performance and Diagnostic Metrics (PDM) destination options extension header. The field limits, calculations, and usage in measurement of PDM are included in this document.

Table of Contents

1 Background 5
1.1 Terminology 5
1.2 Rationale for defined solution 5
1.3 IPv6 Transition Technologies 6
2 Measurement Information Derived from PDM 6
2.1 Round-Trip Delay 6
2.2 Server Delay 7
3 Performance and Diagnostic Metrics Destination Option Layout . . 7
3.1 Destination Options Header 7
3.2 Performance and Diagnostic Metrics Destination Option . . . 7
3.2.1 PDM Layout 7
3.2.2 Base Unit for Time Measurement 10
3.3 Header Placement 11
3.4 Header Placement Using IPSec ESP Mode 11
3.4.1 Using ESP Transport Mode 11
3.4.2 Using ESP Tunnel Mode 12
3.5 Implementation Considerations 12
3.5.1 PDM Activation 12
3.5.2 PDM Timestamps 12
3.6 Dynamic Configuration Options 12
3.7 Information Access and Storage 13
4 Security Considerations 13
4.1 Resource Consumption and Resource Consumption Attacks . . . 13
4.2 Pervasive monitoring 13
4.3 PDM as a Covert Channel 14
4.4 Timing Attacks 14
5 IANA Considerations 15
6 References 15
6.1 Normative References 15
6.2 Informative References 16
Appendix A: Context for PDM 16
A.1 End User Quality of Service (QoS) 16
A.2 Need for a Packet Sequence Number (PSN) 17
A.3 Rationale for Defined Solution 17
A.4 Use PDM with Other Headers 17
Appendix B : Timing Considerations 19
B.1 Timing Differential Calculations 19
B.2 Considerations of this time-differential representation . . 20
B.2.1 Limitations with this encoding method 20
B.2.2 Loss of precision induced by timer value truncation . . 21
Appendix C: Sample Packet Flows 22
C.1 PDM Flow - Simple Client Server 22
C.1.1 Step 1 23
C.1.2 Step 2 23
C.1.3 Step 3 24
C.1.4 Step 4 25

C.1.5 Step 5 26
C.2 Other Flows 26
C.2.1 PDM Flow – One Way Traffic 26
C.2.2 PDM Flow – Multiple Send Traffic 28
C.2.3 PDM Flow – Multiple Send with Errors 29
Appendix D: Potential Overhead Considerations 30
Acknowledgments 31
Authors' Addresses 32

1 Background

To assess performance problems, measurements based on optional sequence numbers and timing may be embedded in each packet. Such measurements may be interpreted in real-time or after the fact.

As defined in RFC2460 [RFC2460], destination options are carried by the IPv6 Destination Options extension header. Destination options include optional information that need be examined only by the IPv6 node given as the destination address in the IPv6 header, not by routers or other "middle boxes". This document specifies the Performance and Diagnostic Metrics (PDM) destination option. The field limits, calculations, and usage in measurement of the PDM destination options extension header are included in this document.

1.1 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 RFC 2119 [RFC2119].

1.2 Rationale for defined solution

The current IPv6 specification does not provide timing nor a similar field in the IPv6 main header or in any extension header. The IPv6 Performance and Diagnostic Metrics destination option (PDM) provides such fields.

Advantages include:

1. Real measure of actual transactions.

2. Ability to span organizational boundaries with consistent instrumentation.

3. No time synchronization needed between session partners

4. Ability to handle all transport protocols (TCP, UDP, SCTP, etc) in a uniform way

The PDM provides the ability to determine quickly if the (latency) problem is in the network or in the server (application). That is, it is a fast way to do triage. For more information on background and usage of PDM, see Appendix A.

1.3 IPv6 Transition Technologies

In the path to full implementation of IPv6, transition technologies such as translation or tunneling may be employed. It is possible that an IPv6 packet containing PDM may be dropped if using IPv6 transition technologies. For example, an implementation using a translation technique (IPv6 to IPv4) which does not support or recognize the IPv6 Destination Options extension header may simply drop the packet rather than translating it without the extension header.

It is also possible that some devices in the network may not correctly handle multiple IPv6 Extension Headers, including the IPv6 Destination Option. For example, adding the PDM header to a packet may push the layer 4 information to a point in the packet where it is not visible to filtering logic, and may be dropped. This kind of situation is expected to become rare over time.

2 Measurement Information Derived from PDM

Each packet contains information about the sender and receiver. In IP protocol, the identifying information is called a "5-tuple".

The 5-tuple consists of:

```
SADDR : IP address of the sender
SPORT : Port for sender
DADDR : IP address of the destination
DPORT : Port for destination
PROTC : Protocol for upper layer (ex. TCP, UDP, ICMP, etc.)
```

The PDM contains the following base fields:

```
PSNTP    : Packet Sequence Number This Packet
PSNLR    : Packet Sequence Number Last Received
DELTATLR : Delta Time Last Received
DELTATLS : Delta Time Last Sent
```

Other fields for storing time scaling factors are also in the PDM and will be described in section 3.

This information, combined with the 5-tuple, allows the measurement of the following metrics:

1. Round-trip delay
2. Server delay

2.1 Round-Trip Delay

Round-trip *Network* delay is the delay for packet transfer from a source host to a destination host and then back to the source host. This measurement has been defined, and the advantages and disadvantages discussed in "A Round-trip Delay Metric for IPPM" [RFC2681].

2.2 Server Delay

Server delay is the interval between when a packet is received by a device and the first corresponding packet is sent back in response. This may be "Server Processing Time". It may also be a delay caused by acknowledgements. Server processing time includes the time taken by the combination of the stack and application to return the response. The stack delay may be related to network performance. If this aggregate time is seen as a problem, and there is a need to make a clear distinction between application processing time and stack delay, including that caused by the network, then more client based measurements are needed.

3 Performance and Diagnostic Metrics Destination Option Layout

3.1 Destination Options Header

The IPv6 Destination Options Header is used to carry optional information that needs to be examined only by a packet's destination node(s). The Destination Options Header is identified by a Next Header value of 60 in the immediately preceding header and is defined in RFC2460 [RFC2460]. The IPv6 Performance and Diagnostic Metrics Destination Option (PDM) is implemented as an IPv6 Option carried in the Destination Options Header. The PDM does not require time synchronization.

3.2 Performance and Diagnostic Metrics Destination Option

3.2.1 PDM Layout

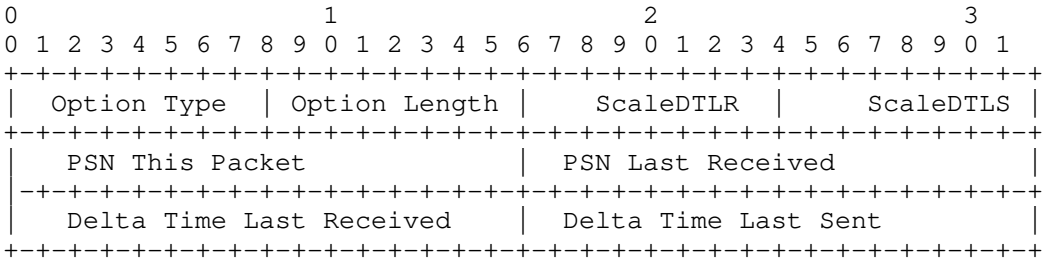
The IPv6 Performance and Diagnostic Metrics Destination Option (PDM) contains the following fields:

```
SCALEDTLR: Scale for Delta Time Last Received
SCALEDTLS: Scale for Delta Time Last Sent
PSNTP    : Packet Sequence Number This Packet
PSNLR    : Packet Sequence Number Last Received
DELTATLR : Delta Time Last Received
DELTATLS : Delta Time Last Sent
```

PDM has alignment requirements. Following the convention in IPv6, these options are aligned in a packet so that multi-octet values

within the Option Data field of each option fall on natural boundaries (i.e., fields of width n octets are placed at an integer multiple of n octets from the start of the header, for n = 1, 2, 4, or 8) [RFC2460].

The PDM destination option is encoded in type-length-value (TLV) format as follows:



Option Type

TBD = 0xXX (TBD) [To be assigned by IANA] [RFC2780]

In keeping with RFC2460[RFC2460], the two high order bits of the Option Type field are encoded to indicate specific processing of the option; for the PDM destination option, these two bits MUST be set to 00.

The third high order bit of the Option Type specifies whether or not the Option Data of that option can change en-route to the packet's final destination.

In the PDM, the value of the third high order bit MUST be 0.

Option Length

8-bit unsigned integer. Length of the option, in octets, excluding the Option Type and Option Length fields. This field MUST be set to 10.

Scale Delta Time Last Received (SCALEDTLR)

8-bit unsigned integer. This is the scaling value for the Delta Time Last Received (DELTATLR) field. The possible values are from 0-255. See Section 4 for further discussion on Timing Considerations and formatting of the scaling values.

Scale Delta Time Last Sent (SCALEDTLS)

8-bit signed integer. This is the scaling value for the Delta Time Last Sent (DELTATLS) field. The possible values are from 0 to 255.

Packet Sequence Number This Packet (PSNTP)

16-bit unsigned integer. This field will wrap. It is intended for use while analyzing packet traces.

Initialized at a random number and incremented monotonically for each packet of the session flow of the 5-tuple. The random number initialization is intended to make it harder to spoof and insert such packets.

Operating systems MUST implement a separate packet sequence number counter per 5-tuple.

Packet Sequence Number Last Received (PSNLR)

16-bit unsigned integer. This is the PSNTP of the packet last received on the 5-tuple.

This field is initialized to 0.

Delta Time Last Received (DELTATLR)

A 16-bit unsigned integer field. The value is set according to the scale in SCALEDTLR.

Delta Time Last Received = (Send time packet n - Receive time packet n-1)

Delta Time Last Sent (DELTATLS)

A 16-bit unsigned integer field. The value is set according to the scale in SCALEDTLS.

Delta Time Last Sent = (Receive time packet n - Send time packet n-1)

3.2.2 Base Unit for Time Measurement

A time differential is always a whole number in a CPU; it is the unit specification -- hours, seconds, nanoseconds -- that determine what the numeric value means. For PDM, the base time unit is 1 attosecond (asec). This allows for a common unit and scaling of the time differential among all IP stacks and hardware implementations.

Note that PDM provides the ability to measure both time differentials that are extremely small, and time differentials in a Delay/Disruption Tolerant Networking (DTN) environment where the

delays may be very great. To store a time differential in just 16 bits that must range in this way will require some scaling of the time differential value.

One issue is the conversion from the native time base in the CPU hardware of whatever device is in use to some number of attoseconds. It might seem this will be an astronomical number, but the conversion is straightforward. It involves multiplication by an appropriate power of 10 to change the value into a number of attoseconds. Then, to scale the value so that it fits into DELTATLR or DELTATLS, the value is shifted by of a number of bits, retaining the 16 high-order or most significant bits. The number of bits shifted becomes the scaling factor, stored as SCALEDTLR or SCALEDTLS, respectively. For additional information of this process, including examples, please see Appendix A.

3.3 Header Placement

The PDM Destination Option is placed as defined in RFC2460 [RFC2460]. There may be a choice of where to place the Destination Options header. If using ESP mode, please see section 3.4 of this document for placement of the PDM Destination Options header.

For each IPv6 packet header, the PDM MUST NOT appear more than once. However, an encapsulated packet MAY contain a separate PDM associated with each encapsulated IPv6 header.

3.4 Header Placement Using IPSec ESP Mode

IPSec Encapsulating Security Payload (ESP) is defined in [RFC4303] and is widely used. Section 3.1.1 of [RFC4303] discusses placement of Destination Options Headers.

The placement of PDM is different depending on if ESP is used in tunnel or transport mode.

In ESP case, no 5-tuple is available, as there are no port numbers. ESP flow should be identified only by using SADDR, DADDR and PROTOC. The SPI numbers SHOULD be ignored when considering the flow over which PDM information is measured.

3.4.1 Using ESP Transport Mode

Note that Destination Options may be placed before or after ESP or both. If using PDM in ESP transport mode, PDM MUST be placed after the ESP header so as not to leak information.

3.4.2 Using ESP Tunnel Mode

Note that Destination Options may be placed before or after ESP or both in both the outer set of IP headers and the inner set of IP headers. A tunnel endpoint that creates a new packet may decide to use PDM independent of the use of PDM of the original packet to enable delay measurements between the two tunnel endpoints.

3.5 Implementation Considerations

3.5.1 PDM Activation

An implementation should provide an interface to enable or disable the use of PDM. This specification recommends having PDM off by default.

PDM MUST NOT be turned on merely if a packet is received with a PDM header. The received packet could be spoofed by another device.

3.5.2 PDM Timestamps

The PDM timestamps are intended to isolate wire time from server or host time, but may necessarily attribute some host processing time to network latency.

RFC2330 [RFC2330] "Framework for IP Performance Metrics" describes two notions of wire time in section 10.2. These notions are only defined in terms of an Internet host H observing an Internet link L at a particular location:

+ For a given IP packet P, the 'wire arrival time' of P at H on L is the first time T at which any bit of P has appeared at H's observational position on L.

+ For a given IP packet P, the 'wire exit time' of P at H on L is the first time T at which all the bits of P have appeared at H's observational position on L.

This specification does not define the exact H's observing position on L. That is left for the deployment setups to define. However, the position where PDM timestamps are taken SHOULD be as close to the physical network interface as possible. Not all implementations will be able to achieve the ideal level of measurement.

3.6 Dynamic Configuration Options

If the PDM destination options extension header is used, then it MAY be turned on for all packets flowing through the host, applied to an upper-layer protocol (TCP, UDP, SCTP, etc), a local port, or IP address only. These are at the discretion of the implementation.

3.7 Information Access and Storage

Measurement information provided by PDM may be made accessible for higher layers or the user itself. Similar to activating the use of PDM, the implementation may also provide an interface to indicate if received

PDM information may be stored, if desired. If a packet with PDM information is received and the information should be stored, the upper layers may be notified. Furthermore, the implementation should define a configurable maximum lifetime after which the information can be removed as well as a configurable maximum amount of memory that should be allocated for PDM information.

4 Security Considerations

PDM may introduce some new security weaknesses.

4.1 Resource Consumption and Resource Consumption Attacks

PDM needs to calculate the deltas for time and keep track of the sequence numbers. This means that control blocks which reside in memory may be kept at the end hosts per 5-tuple.

A limit on how much memory is being used SHOULD be implemented.

Without a memory limit, any time a control block is kept in memory, an attacker can try to misuse the control blocks to cause excessive resource consumption. This could be used to compromise the end host.

PDM is used only at the end hosts and memory is used only at the end host and not at routers or middle boxes.

4.2 Pervasive monitoring

Since PDM passes in the clear, a concern arises as to whether the data can be used to fingerprint the system or somehow obtain information about the contents of the payload.

Let us discuss fingerprinting of the end host first. It is possible that seeing the pattern of deltas or the absolute values could give some information as to the speed of the end host - that is, if it is a very fast system or an older, slow device. This may be useful to

the attacker. However, if the attacker has access to PDM, the attacker also has access to the entire packet and could make such a deduction based merely on the time frames elapsed between packets WITHOUT PDM.

As far as deducing the content of the payload, in terms of the application level information such as web page, user name, user password and so on, it appears to us that PDM is quite unhelpful in this regard. Having said that, the ability to separate wire-time from processing time may potentially provide an attacker with additional information. It is conceivable that an attacker could attempt to deduce the type of application in use by noting the server time and payload length. Some encryption algorithms attempt to obfuscate the packet length to avoid just such vulnerabilities. In the future, encryption algorithms may wish to obfuscate the server time as well.

4.3 PDM as a Covert Channel

PDM provides a set of fields in the packet which could be used to leak data. But, there is no real reason to suspect that PDM would be chosen rather than another part of the payload or another Extension Header.

A firewall or another device could sanity check the fields within the PDM but randomly assigned sequence numbers and delta times might be expected to vary widely. The biggest problem though is how an attacker would get access to PDM in the first place to leak data. The attacker would have to either compromise the end host or have Man in the Middle (MitM). It is possible that either one could change the fields. But, then the other end host would get sequence numbers and deltas that don’t make any sense.

It is conceivable that someone could compromise an end host and make it start sending packets with PDM without the knowledge of the host. But, again, the bigger problem is the compromise of the end host. Once that is done, the attacker probably has better ways to leak data.

Having said that, if a PDM aware middle box or an implementation (destination host) detects some number of "nonsensical" sequence numbers or timing information, it could take action to block, discard, or alert on this traffic.

4.4 Timing Attacks

The fact that PDM can help in the separation of node processing time

from network latency brings value to performance monitoring. Yet, it is this very characteristic of PDM which may be misused to make certain new type of timing attacks against protocols and implementations possible.

Depending on the nature of the cryptographic protocol used, it may be possible to leak the credentials of the device. For example, if an attacker can see that PDM is being used, then the attacker might use PDM to launch a timing attack against the keying material used by the cryptographic protocol.

An implementation may want to be sure that PDM is enabled only for certain ip addresses, or only for some ports. Additionally, the implementation SHOULD require an explicit restart of monitoring after a certain time period (for example for 1 hour), to make sure that PDM is not accidentally left on after debugging has been done etc.

Even so, if using PDM, a user "Consent to be Measured" SHOULD be a pre-requisite for using PDM. Consent is common in enterprises and with some subscription services. The actual content of "Consent to be Measured" will differ by site but it SHOULD make clear that the traffic is being measured for quality of service and to assist in diagnostics as well as to make clear that there may be potential risks of certain vulnerabilities if the traffic is captured during a diagnostic session.

5 IANA Considerations

This draft requests an Destination Option Type assignment with the act bits set to 00 and the chg bit set to 0 from the Destination Options and Hop-by-Hop Options sub-registry of Internet Protocol Version 6 (IPv6) Parameters [ref to RFCs and URL below].

http://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml#ipv6-parameters-2

Hex Value	Binary Value act chg rest	Description	Reference
TBD	TBD	Performance and Diagnostic Metrics (PDM)	[This draft]

6 References

6.1 Normative References

[RFC1122] Braden, R., "Requirements for Internet Hosts -- Communication Layers", RFC 1122, October 1989.

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

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC2681] Almes, G., Kalidindi, S., and M. Zekauskas, "A Round-trip Delay Metric for IPPM", RFC 2681, September 1999.

[RFC2780] Bradner, S. and V. Paxson, "IANA Allocation Guidelines For Values In the Internet Protocol and Related Headers", BCP 37, RFC 2780, March 2000.

[RFC4303] Kent, S, "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005.

6.2 Informative References

[RFC2330] Paxson, V., Almes, G., Mahdavi, J., and M. Mathis, "Framework for IP Performance Metrics", RFC 2330, May 1998.

[TRAM-TCPM] Trammel, B., "Encoding of Time Intervals for the TCP Timestamp Option-01", Internet Draft, July 2013. [Work in Progress]

Appendix A: Context for PDM

A.1 End User Quality of Service (QoS)

The timing values in the PDM embedded in the packet will be used to estimate QoS as experienced by an end user device.

For many applications, the key user performance indicator is response time. When the end user is an individual, he is generally indifferent to what is happening along the network; what he really cares about is how long it takes to get a response back. But this is not just a matter of individuals' personal convenience. In many cases, rapid response is critical to the business being conducted.

Low, reliable and acceptable response times are not just "nice to have". On many networks, the impact can be financial hardship or can endanger human life. In some cities, the emergency police contact

system operates over IP; law enforcement, at all levels, use IP networks; transactions on our stock exchanges are settled using IP networks. The critical nature of such activities to our daily lives and financial well-being demand a simple solution to support response time measurements.

A.2 Need for a Packet Sequence Number (PSN)

While performing network diagnostics of an end-to-end connection, it often becomes necessary to isolate the factors along the network path responsible for problems. Diagnostic data may be collected at multiple places along the path (if possible), or at the source and destination. Then, in post-collection processing, the diagnostic data corresponding to each packet at different observation points must be matched for proper measurements. A sequence number in each packet provides sufficient basis for the matching process. If need be, the timing fields may be used along with the sequence number to ensure uniqueness.

This method of data collection along the path is of special use to determine where packet loss or packet corruption is happening.

The packet sequence number needs to be unique in the context of the session (5-tuple).

A.3 Rationale for Defined Solution

One of the important functions of PDM is to allow you to quickly dispatch the right set of diagnosticians. Within network or server latency, there may be many components. The job of the diagnostician is to rule each one out until the culprit is found.

How PDM fits into this diagnostic picture is that PDM will quickly tell you how to escalate. PDM will point to either the network area or the server area. Within the server latency, PDM does not tell you if the bottleneck is in the IP stack or the application or buffer allocation. Within the network latency, PDM does not tell you which of the network segments or middle boxes is at fault.

What PDM does tell you is whether the problem is in the network or the server.

A.4 Use PDM with Other Headers

For diagnostics, one my want to use PDM with other headers (L2, L3, etc). For example, if PDM is used is by a technician (or tool) looking at a packet capture, within the packet capture, they would have available to them the layer 2 header, IP header (v6 or v4), TCP,

UCP, ICMP, SCTP or other headers. All information would be looked at together to make sense of the packet flow. The technician or processing tool could analyze, report or ignore the data from PDM, as necessary.

For an example of how PDM can help with TCP retransmit problems, please look at Appendix C.

Appendix B : Timing Considerations

B.1 Timing Differential Calculations

The time counter in a CPU is a binary whole number, representing a number of milliseconds (msec), microseconds (usec) or even picoseconds (psec). Representing one of these values as attoseconds (asec) means multiplying by 10 raised to some exponent. Refer to this table of equalities:

Base value	= # of sec	= # of asec	1000s of asec
1 second	1 sec	10**18 asec	1000**6 asec
1 millisecond	10**-3 sec	10**15 asec	1000**5 asec
1 microsecond	10**-6 sec	10**12 asec	1000**4 asec
1 nanosecond	10**-9 sec	10**9 asec	1000**3 asec
1 picosecond	10**-12 sec	10**6 asec	1000**2 asec
1 femtosecond	10**-15 sec	10**3 asec	1000**1 asec

For example, if you have a time differential expressed in microseconds, since each microsecond is 10**12 asec, you would multiply your time value by 10**12 to obtain the number of attoseconds. If you time differential is expressed in nanoseconds, you would multiply by 10**9 to get the number of attoseconds.

The result is a binary value that will need to be shortened by a number of bits so it will fit into the 16-bit PDM DELTA field.

The next step is to divide by 2 until the value is contained in just 16 significant bits. The exponent of the value in the last column of of the table is useful here; the initial scaling factor is that exponent multiplied by 10. This is the minimum number of low-order bits to be shifted-out or discarded. It represents dividing the time value by 1024 raised to that exponent.

The resulting value may still be too large to fit into 16 bits, but can be normalized by shifting out more bits (dividing by 2) until the value fits into the 16-bit DELTA field. The number of extra bits shifted out is then added to the scaling factor. The scaling factor, the total number of low-order bits dropped, is the SCALEDTL value.

For example: say an application has these start and finish timer values (hexadecimal values, in microseconds):

Finish:	27C849234 usec	(02:57:58.997556)
-Start:	27C83F696 usec	(02:57:58.957718)
==========	=========	==========
Difference	9B9E usec	00.039838 sec or 39838 usec

To convert this differential value to binary attoseconds, multiply the number of microseconds by 10**12. Divide by 1024**4, or simply discard 40 bits from the right. The result is 36232, or 8D88 in hex, with a scaling factor or SCALEDTL value of 40.

For another example, presume the time differential is larger, say 32.311072 seconds, which is 32311072 usec. Each microsecond is 10**12 asec, so multiply by 10**12, giving the hexadecimal value 1C067FCCAE8120000. Using the initial scaling factor of 40, drop the last 10 characters (40 bits) from that string, giving 1C067FC. This will not fit into a DELTA field, as it is 25 bits long. Shifting the value to the right another 9 bits results in a DELTA value of E033, with a resulting scaling factor of 49.

When the time differential value is a small number, regardless of the time unit, the exponent trick given above is not useful in determining the proper scaling value. For example, if the time differential is 3 seconds and you want to convert that directly, you would follow this path:

```
3 seconds = 3*10**18 asec (decimal)
          = 29A2241AF62C0000 asec (hexadecimal)
```

If you just truncate the last 60 bits, you end up with a delta value of 2 and a scaling factor of 60, when what you really wanted was a delta value with more significant digits. The most precision with which you can store this value in 16 bits is A688, with a scaling factor of 46.

B.2 Considerations of this time-differential representation

There are a few considerations to be taken into account with this representation of a time differential. The first is whether there are any limitations on the maximum or minimum time differential that can be expressed using method of a delta value and a scaling factor. The second is the amount of imprecision introduced by this method.

B.2.1 Limitations with this encoding method

The DELTATLS and DELTATLR fields store only the 16 most-significant bits of the time differential value. Thus the range, excluding the scaling factor, is from 0 to 65535, or a maximum of 2**16-1. This method is further described in [TRAM-TCPM].

The actual magnitude of the time differential is determined by the scaling factor. SCALEDTLR and SCALEDTLS are 8-bit unsigned integers, so the scaling factor ranges from 0 to 255. The smallest number that can be represented would have a value of 1 in the delta field and a

value of 0 in the associated scale field. This is the representation for 1 attosecond. Clearly this allows PDM to measure extremely small time differentials.

On the other end of the scale, the maximum delta value is 65535, or FFFF in hexadecimal. If the maximum scale value of 255 is used, the time differential represented is 65535*2**255, which is over 3*10**55 years, essentially, forever. So there appears to be no real limitation to the time differential that can be represented.

B.2.2 Loss of precision induced by timer value truncation

As PDM specifies the DELTATLR and DELTATLS values as 16-bit unsigned integers, any time the precision is greater than those 16 bits, there will be truncation of the trailing bits, with an accompanying loss of precision in the value.

Any time differential value smaller than 65536 asec can be stored exactly in DELTATLR or DELTATLS, because the representation of this value requires at most 16 bits.

Since the time differential values in PDM are measured in attoseconds, the range of values that would be truncated to the same encoded value is 2**(Scale)-1 asec.

For example, the smallest time differential that would be truncated to fit into a delta field is

1 0000 0000 0000 0000 = 65536 asec

This value would be encoded as a delta value of 8000 (hexadecimal) with a scaling factor of 1. The value

1 0000 0000 0000 0001 = 65537 asec

would also be encoded as a delta value of 8000 with a scaling factor of 1. This actually is the largest value that would be truncated to that same encoded value. When the scale value is 1, the value range is calculated as 2**1 - 1, or 1 asec, which you can see is the difference between these minimum and maximum values.

The scaling factor is defined as the number of low-order bits truncated to reduce the size of the resulting value so it fits into a 16-bit delta field. If, for example, you had to truncate 12 bits, the loss of precision would depend on the bits you truncated. The range of these values would be

0000 0000 0000 = 0 asec

to
1111 1111 1111 = 4095 asec

So the minimum loss of precision would be 0 asec, where the delta value exactly represents the time differential, and the maximum loss of precision would be 4095 asec. As stated above, the scaling factor of 12 means the maximum loss of precision is 2**12-1 asec, or 4095 asec.

Compare this loss of precision to the actual time differential. The range of actual time differential values that would incur this loss of precision is from

1000 0000 0000 0000 0000 0000 0000 = 2**27 asec or 134217728 asec
to
1111 1111 1111 1111 1111 1111 1111 = 2**28-1 asec or 268435455 asec

Granted, these are small values, but the point is, any value between these two values will have a maximum loss of precision of 4095 asec, or about 0.00305% for the first value, as encoded, and at most 0.001526% for the second. These maximum-loss percentages are consistent for all scaling values.

Appendix C: Sample Packet Flows

C.1 PDM Flow - Simple Client Server

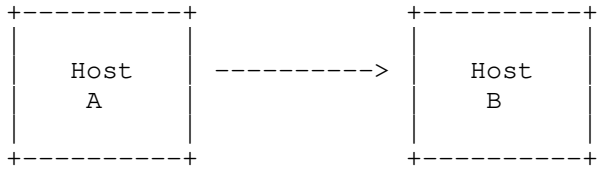
Following is a sample simple flow for the PDM with one packet sent from Host A and one packet received by Host B. The PDM does not require time synchronization between Host A and Host B. The calculations to derive meaningful metrics for network diagnostics are shown below each packet sent or received.

C.1.1 Step 1

Packet 1 is sent from Host A to Host B. The time for Host A is set initially to 10:00AM.

The time and packet sequence number are saved by the sender internally. The packet sequence number and delta times are sent in the packet.

Packet 1



PDM Contents:

```
PSNTP    : Packet Sequence Number This Packet:     25
PSNLR    : Packet Sequence Number Last Received:   -
DELTATLR : Delta Time Last Received:               -
SCALEDTLR: Scale of Delta Time Last Received:      0
DELTATLS : Delta Time Last Sent:                   -
SCALEDTLS: Scale of Delta Time Last Sent:          0
```

Internally, within the sender, Host A, it must keep:

```
Packet Sequence Number of the last packet sent:     25
Time the last packet was sent:                10:00:00
```

Note, the initial PSNTP from Host A starts at a random number. In this case, 25. The time in these examples is shown in seconds for the sake of simplicity.

C.1.2 Step 2

Packet 1 is received at Host B. Its time is set to one hour later than Host A. In this case, 11:00AM

Internally, within the receiver, Host B, it must note:

```
Packet Sequence Number of the last packet received:    25
Time the last packet was received                 :    11:00:03
```

Note, this timestamp is in Host B time. It has nothing whatsoever to do with Host A time. The Packet Sequence Number of the last packet

received will become PSNLR which will be sent out in the packet sent by Host B in the next step. The time last received will be used to calculate the DELTALR value to be sent out in the packet sent by Host B in the next step.

C.1.3 Step 3

Packet 2 is sent by Host B to Host A. Note, the initial packet sequence number (PSNTP) from Host B starts at a random number. In this case, 12. Before sending the packet, Host B does a calculation of deltas. Since Host B knows when it is sending the packet, and it knows when it received the previous packet, it can do the following calculation:

Sending time : packet 2 - receive time : packet 1

The result of this calculation is called: Delta Time Last Received (DELTATLR)

Note, both sending time and receive time are saved internally in Host B. They do not travel in the packet. Only the Delta is in the packet.

Assume that within Host B is the following:

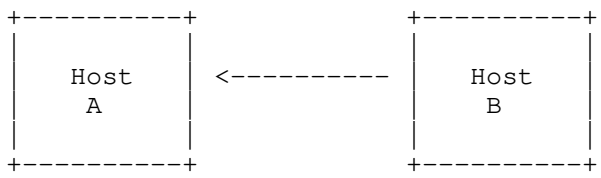
Packet Sequence Number of the last packet received:	25
Time the last packet was received:	11:00:03
Packet Sequence Number of this packet:	12
Time this packet is being sent:	11:00:07

Now a delta value to be sent out in the packet can be calculated. DELTATLR becomes:

4 seconds = 11:00:07 - 11:00:03 = 3782DACE9D900000 asec

This is the derived metric: Server Delay. The time and scaling factor must be converted; in this case, the time differential is DE0B, and the scaling factor is 2E, or 46 in decimal. Then, these values, along with the packet sequence numbers will be sent to Host A as follows:

Packet 2



PDM Contents:

```
PSNTP    : Packet Sequence Number This Packet:    12
PSNLR    : Packet Sequence Number Last Received:  25
DELTATLR : Delta Time Last Received:              DE0B (4 seconds)
SCALEDTLR: Scale of Delta Time Last Received:     2E (46 decimal)
DELTATLS : Delta Time Last Sent:                   -
SCALEDTLS: Scale of Delta Time Last Sent:          0
```

The metric left to be calculated is the Round-Trip Delay. This will be calculated by Host A when it receives Packet 2.

C.1.4 Step 4

Packet 2 is received at Host A. Remember, its time is set to one hour earlier than Host B. Internally, it must note:

```
Packet Sequence Number of the last packet received:    12
Time the last packet was received                :    10:00:12
```

Note, this timestamp is in Host A time. It has nothing whatsoever to do with Host B time.

So, now, Host A can calculate total end-to-end time. That is:

End-to-End Time = Time Last Received - Time Last Sent

For example, packet 25 was sent by Host A at 10:00:00. Packet 12 was received by Host A at 10:00:12 so:

End-to-End time = 10:00:12 - 10:00:00 or 12 (Server and Network RT delay combined). This time may also be called total Overall Round-Trip Time (RTT) which includes Network RTT and Host Response Time.

This derived metric we will call Delta Time Last Sent (DELTATLS)

Round trip delay can now be calculated. The formula is:

Round trip delay = (Delta Time Last Sent - Delta Time Last Received)

Or:

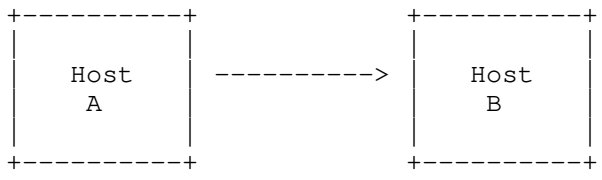
Round trip delay = 12 - 4 or 8

Now, the only problem is that at this point all metrics are in Host A only and not exposed in a packet. To do that, we need a third packet.

Note: this simple example assumes one send and one receive. That is done only for purposes of explaining the function of the PDM. In cases where there are multiple packets returned, one would take the time in the last packet in the sequence. The calculations of such timings and intelligent processing is the function of post-processing of the data.

C.1.5 Step 5

Packet 3 is sent from Host A to Host B.



PDM Contents:

```
PSNTP    : Packet Sequence Number This Packet:    26
PSNLR    : Packet Sequence Number Last Received:  12
DELTATLR : Delta Time Last Received:               0
SCALEDTLS: Scale of Delta Time Last Received       0
DELTATLS : Delta Time Last Sent:                 A688 (scaled value)
SCALEDTLR: Scale of Delta Time Last Received:     30 (48 decimal)
```

To calculate Two-Way Delay, any packet capture device may look at these packets and do what is necessary.

C.2 Other Flows

What has been discussed so far is a simple flow with one packet sent and one returned. Let's look at how PDM may be useful in other types of flows.

C.2.1 PDM Flow - One Way Traffic

The flow on a particular session may not be a send-receive paradigm. Let us consider some other situations. In the case of a one-way flow, one might see the following:

Note: The time is expressed in generic units for simplicity. That is, these values do not represent a number of attoseconds, microseconds or any other real units of time.

Packet	Sender	PSN This Packet	PSN Last Recvd	Delta Time Last Recvd	Delta Time Last Sent
1	Server	1	0	0	0
2	Server	2	0	0	5
3	Server	3	0	0	12
4	Server	4	0	0	20

What does this mean and how is it useful?

In a one-way flow, only the Delta Time Last Sent will be seen as used. Recall, Delta Time Last Sent is the difference between the send of one packet from a device and the next. This is a measure of throughput for the sender - according to the sender's point of view. That is, it is a measure of how fast is the application itself (with stack time included) able to send packets.

How might this be useful? If one is having a performance issue at the client and sees that packet 2, for example, is sent after 5 time units from the server but takes 10 times that long to arrive at the destination, then one may safely conclude that there are delays in the path other than at the server which may be causing the delivery issue of that packet. Such delays may include the network links, middle-boxes, etc.

Now, true one-way traffic is quite rare. What people often mean by "one-way" traffic is an application such as FTP where a group of packets (for example, a TCP window size worth) is sent, then the sender waits for acknowledgment. This type of flow would actually fall into the "multiple-send" traffic model.

C.2.2 PDM Flow - Multiple Send Traffic

Assume that two packets are sent for each ACK from the server. For example, a TCP flow will do this, per RFC1122 [RFC1122] Section-4.2.3.

Packet	Sender	PSN This Packet	PSN Last Recvd	Delta Time Last Recvd	Delta Time Last Sent
1	Server	1	0	0	0
2	Server	2	0	0	5
3	Client	1	2	20	0
4	Server	3	1	10	15

How might this be used?

Notice that in packet 3, the client has a value of Delta Time Last received of 20. Recall that Delta Time Last Received is the Send time of packet 3 - receive time of packet 2. So, what does one know now? In this case, Delta Time Last Received is the processing time for the Client to send the next packet.

How to interpret this depends on what is actually being sent. Remember, PDM is not being used in isolation, but to supplement the fields found in other headers. Let's take some examples:

1. Client is sending a standalone TCP ACK. One would find this by looking at the payload length in the IPv6 header and the TCP Acknowledgement field in the TCP header. So, in this case, the client is taking 20 units to send back the ACK. This may or may not be interesting.

2. Client is sending data with the packet. Again, one would find this by looking at the payload length in the IPv6 header and the TCP Acknowledgement field in the TCP header. So, in this case, the client is taking 20 units to send back data. This may represent "User Think Time". Again, this may or may not be interesting, in isolation. But, if there is a performance problem receiving data at the server, then taken in conjunction with RTT or other packet timing information, this information may be quite interesting.

Of course, one also needs to look at the PSN Last Received field to make sure of the interpretation of this data. That is, to make sure that the Delta Last Received corresponds to the packet of interest.

The benefits of PDM are that such information is now available in a uniform manner for all applications and all protocols without

extensive changes required to applications.

C.2.3 PDM Flow - Multiple Send with Errors

Let us now look at a case of how PDM may be able to help in a case of TCP retransmission and add to the information that is sent in the TCP header.

Assume that three packets are sent with each send from the server.

From the server, this is what is seen.

Pkt	Sender	PSN This Pkt	PSN LastRecvd	Delta Time LastRecvd	Delta Time LastSent	TCP SEQ	Data Bytes
1	Server	1	0	0	0	123	100
2	Server	2	0	0	5	223	100
3	Server	3	0	0	5	333	100

The client, however, does not receive all the packets. From the client, this is what is seen for the packets sent from the server.

Pkt	Sender	PSN This Pkt	PSN LastRecvd	Delta Time LastRecvd	Delta Time LastSent	TCP SEQ	Data Bytes
1	Server	1	0	0	0	123	100
2	Server	3	0	0	5	333	100

Let's assume that the server now retransmits the packet. (Obviously, a duplicate acknowledgment sequence for fast retransmit or a retransmit timeout would occur. To illustrate the point, these packets are being left out.)

So, then if a TCP retransmission is done, then from the client, this is what is seen for the packets sent from the server.

Pkt	Sender	PSN This Pkt	PSN LastRecvd	Delta Time LastRecvd	Delta Time LastSent	TCP SEQ	Data Bytes
1	Server	4	0	0	30	223	100

The server has resent the old packet 2 with TCP sequence number of 223. The retransmitted packet now has a PSN This Packet value of 4.

The Delta Last Sent is 30 - the time between sending the packet with PSN of 3 and this current packet.

Let's say that packet 4 is lost again. Then, after some amount of time (RTO) then the packet with TCP sequence number of 223 is resent.

From the client, this is what is seen for the packets sent from the server.

Pkt	Sender	PSN This Pkt	PSN LastRecvd	Delta Time LastRecvd	Delta Time LastSent	TCP SEQ	Data Bytes
1	Server	5	0	0	60	223	100

If now, this packet arrives at the destination, one has a very good idea that packets exist which are being sent from the server as retransmissions and not arriving at the client. This is because the PSN of the resent packet from the server is 5 rather than 4. If we had used TCP sequence number alone, we would never have seen this situation. The TCP sequence number in all situations is 223.

This situation would be experienced by the user of the application (the human being actually sitting somewhere) as a "hangs" or long delay between packets. On large networks, to diagnose problems such as these where packets are lost somewhere on the network, one has to take multiple traces to find out exactly where.

The first thing is to start with doing a trace at the client and the server. So, we can see if the server sent a particular packet and the client received it. If the client did not receive it, then we start tracking back to trace points at the router right after the server and the router right before the client. Did they get these packets which the server has sent? This is a time consuming activity.

With PDM, we can speed up the diagnostic time because we may be able to use only the trace taken at the client to see what the server is sending.

Appendix D: Potential Overhead Considerations

One might wonder as to the potential overhead of PDM. First, PDM is entirely optional. That is, a site may choose to implement PDM or not as they wish. If they are happy with the costs of PDM vs. the benefits, then the choice should be theirs.

Below is a table outlining the potential overhead in terms of additional time to deliver the response to the end user for various assumed RTTs.

Bytes in Packet	RTT	Bytes Per Millisec	Bytes in PDM	New RTT	Overhead
1000	1000 milli	1	16	1016.000	16.000 milli
1000	100 milli	10	16	101.600	1.600 milli
1000	10 milli	100	16	10.160	.160 milli
1000	1 milli	1000	16	1.016	.016 milli

Below are some examples of actual RTTs for packets traversing large enterprise networks. The first example is for packets going to multiple business partners.

Bytes in Packet	RTT	Bytes Per Millisec	Bytes in PDM	New RTT	Overhead
1000	17 milli	58	16	17.360	.360 milli

The second example is for packets at a large enterprise customer within a data center. Notice that the scale is now in microseconds rather than milliseconds.

Bytes in Packet	RTT	Bytes Per Microsec	Bytes in PDM	New RTT	Overhead
1000	20 micro	50	16	20.320	.320 micro

As with other diagnostic tools, such as packet traces, a certain amount of processing time will be required to create and process PDM. Since PDM is lightweight (has only a few variables), we expect the processing time to be minimal.

Acknowledgments

The authors would like to thank Keven Haining, Al Morton, Brian Trammel, David Boyes, Bill Jouris, Richard Scheffenegger, and Rick Troth for their comments and assistance. We would also like to thank Tero Kivinen and Jouni Korhonen for their detailed and perceptive reviews.

Authors' Addresses

Nalini Elkins
Inside Products, Inc.
36A Upper Circle
Carmel Valley, CA 93924
United States
Phone: +1 831 659 8360
Email: nalini.elkins@insidethestack.com
http://www.insidethestack.com

Robert M. Hamilton
Chemical Abstracts Service
A Division of the American Chemical Society
2540 Olentangy River Road
Columbus, Ohio 43202
United States
Phone: +1 614 447 3600 x2517
Email: rhamilton@cas.org
http://www.cas.org

Michael S. Ackermann
Blue Cross Blue Shield of Michigan
P.O. Box 2888
Detroit, Michigan 48231
United States
Phone: +1 310 460 4080
Email: mackermann@bcbsm.com
http://www.bcbsm.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: July 24, 2016

A. Morton
AT&T Labs

January 21, 2016

Active and Passive Metrics and Methods (and everything in-between, or Hybrid)
draft-ietf-ippm-active-passive-06

Abstract

This memo provides clear definitions for Active and Passive performance assessment. The construction of Metrics and Methods can be described as Active or Passive. Some methods may use a subset of both active and passive attributes, and we refer to these as Hybrid Methods. This memo also describes multiple dimensions to help evaluate new methods as they emerge.

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 July 24, 2016.

Table of Contents

1. Introduction 2
1.1. Requirements Language 3
2. Purpose and Scope 3
3. Terms and Definitions 3
3.1. Performance Metric 4
3.2. Method of Measurement 4
3.3. Observation Point 4
3.4. Active Methods 4
3.5. Active Metric 5
3.6. Passive Methods 5
3.7. Passive Metric 6
3.8. Hybrid Methods and Metrics 6
4. Discussion 8
4.1. Graphical Representation 8
4.2. Discussion of PDM 10
4.3. Discussion of "Coloring" Method 11
4.4. Brief Discussion of OAM Methods 11
5. Security considerations 12
6. IANA Considerations 12
7. Acknowledgements 12
8. References 13
8.1. Normative References 13
8.2. Informative References 14
Author's Address 15

1. Introduction

The adjectives "active" and "passive" have been used for many years to distinguish two different classes of Internet performance assessment. The first Passive and Active Measurement (PAM) Conference was held in 2000, but the earliest proceedings available on-line are from the second PAM conference in 2001 [https://www.ripe.net/ripe/meetings/pam-2001].

The notions of "active" and "passive" are well-established. In general:

> An Active metric or method depends on a dedicated measurement packet stream and observations of the stream.
>
> A Passive metric or method depends *solely* on observation of one or more existing packet streams. The streams only serve

measurement when they are observed for that purpose, and are present whether measurements take place or not.

As new techniques for assessment emerge it is helpful to have clear definitions of these notions. This memo provides more detailed definitions, defines a new category for combinations of traditional active and passive techniques, and discusses dimensions to evaluate new techniques as they emerge.

This memo provides definitions for Active and Passive Metrics and Methods based on long usage in the Internet measurement community, and especially the Internet Engineering Task Force. This memo also describes the combination of fundamental Active and Passive categories, which are called Hybrid Methods and Metrics.

1.1. Requirements Language

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 RFC 2119 [RFC2119].

2. Purpose and Scope

The scope of this memo is to define and describe Active and Passive versions of metrics and methods which are consistent with the long-time usage of these adjectives in the Internet measurement community and especially the Internet Engineering Task Force. Since the science of measurement is expanding, we provide a category for combinations of the traditional extremes, treating Active and Passive as a continuum and designating combinations of their attributes as Hybrid methods.

Further, this memo's purpose includes describing multiple dimensions to evaluate new methods as they emerge.

3. Terms and Definitions

This section defines the key terms of the memo. Some definitions use the notion of "stream of interest" which is synonymous with "population of interest" defined in clause 6.1.1 of ITU-T Recommendation Y.1540 [Y.1540]. These definitions will be useful for work-in-progress, such as [I-D.zheng-ippm-framework-passive] (with which there is already good consistency).

3.1. Performance Metric

The standard definition of a quantity, produced in an assessment of performance and/or reliability of the network, which has an intended utility and is carefully specified to convey the exact meaning of a measured value. (This definition is consistent with that of Performance Metric in [RFC2330] and [RFC6390]).

3.2. Method of Measurement

The procedure or set of operations having the object of determining a Measured Value or Measurement Result.

3.3. Observation Point

See section 2 of [RFC7011] for this definition (a location in the network where packets can be observed), and related definitions. The comparable term defined in IETF literature on Active measurement is Measurement Point, see section 4.1 of [RFC5835]. Both of these terms have come into use describing similar actions at the identified point in the network path.

3.4. Active Methods

Active measurement methods have the following attributes:

1. Active methods generate packet streams. Commonly, the packet stream of interest is generated as the basis of measurement. Sometimes, the adjective "synthetic" is used to categorize Active measurement streams [Y.1731]. Accompanying packet stream(s) may be generated to increase overall traffic load, though the loading stream(s) may not be measured.

2. The packets in the stream of interest have fields or field values (or are augmented or modified to include fields or field values) which are dedicated to measurement. Since measurement usually requires determining the corresponding packets at multiple measurement points, a sequence number is the most common information dedicated to measurement, and often combined with a timestamp.

3. The Source and Destination of the packet stream of interest are usually known a priori.

4. The characteristics of the packet stream of interest are known at the Source at least, and may be communicated to Destination as part of the method. Note that some packet characteristics will

normally change during packet forwarding. Other changes along the path are possible, see [I-D.morton-ippm-2330-stdform-typep].

When adding traffic to the network for measurement, Active Methods influence the quantities measured to some degree, and those performing tests should take steps to quantify the effect(s) and/or minimize such effects.

3.5. Active Metric

An Active Metric incorporates one or more of the aspects of Active Methods in the metric definition.

For example, IETF metrics for IP performance (developed according to the [RFC2330] framework) include the Source packet stream characteristics as metric input parameters, and also specify the packet characteristics (Type-P) and Source and Destination IP addresses (with their implications on both stream treatment and interfaces associated with measurement points).

3.6. Passive Methods

Passive measurement methods are

- based solely on observations of undisturbed and unmodified packet stream of interest (in other words, the method of measurement MUST NOT add, change, or remove packets or fields, or change field values anywhere along the path).

- dependent on the existence of one or more packet streams to supply the stream of interest.

- dependent on the presence of the packet stream of interest at one or more designated observation points.

Some passive methods simply observe and collect information on all packets that pass Observation Point(s), while others filter the packets as a first step and only collect information on packets that match the filter criteria, and thereby narrow the stream of interest.

It is common that passive methods are conducted at one or more Observation Points. Passive methods to assess Performance Metrics often require multiple observation points, e.g., to assess latency of packet transfer across a network path between two Observation Points. In this case, the observed packets must include enough information to determine the corresponding packets at different Observation Points.

Communication of the observations (in some form) to a collector is an essential aspect of Passive Methods. In some configurations, the traffic load associated with results export to a collector may influence the network performance. However, the collection of results is not unique to Passive Methods, and the load from management and operations of measurement systems must always be considered for potential effects on the measured values.

3.7. Passive Metric

Passive Metrics apply to observations of packet traffic (traffic flows in [RFC7011]).

Passive performance metrics are assessed independent of the packets or traffic flows, and solely through observation. Some refer to such assessments as "out-of-band".

One example of passive performance metrics for IP packet transfer can be found in ITU-T Recommendation Y.1540 [Y.1540], where the metrics are defined on the basis of reference events generated as packet pass reference points. The metrics are agnostic to the distinction between active and passive when the necessary packet correspondence can be derived from the observed stream of interest as required.

3.8. Hybrid Methods and Metrics

Hybrid Methods are Methods of Measurement which use a combination of Active Methods and Passive Methods, to assess Active Metrics, Passive Metrics, or new metrics derived from the a priori knowledge and observations of the stream of interest. ITU-T Recommendation Y.1540 [Y.1540] defines metrics that are also applicable to the hybrid categories, since packet correspondence at different observation/ reference points could be derived from "fields or field values which are dedicated to measurement", but otherwise the methods are passive.

There are several types of Hybrid methods, as categorized below.

With respect to a *single* stream of interest, Hybrid Type I methods fit in the continuum as follows, in terms of what happens at the Source (or Observation Point nearby):

- If you generate the stream of interest => Active
- If you augment or modify the stream of interest, or employ methods that modify the treatment of the stream => Hybrid Type I
- If you solely observe a stream of interest => Passive

As an example, consider the case where the method generates traffic load stream(s), and observes an existing stream of interest according to the criteria for Passive Methods. Since loading streams are an aspect of Active Methods, the stream of interest is not "solely observed", and the measurements involve a single stream of interest whose treatment has been modified both the presence of the load. Therefore, this is a Hybrid Type I method.

We define Hybrid Type II as follows: Methods that employ two or more different streams of interest with some degree of mutual coordination (e.g., one or more Active streams and one or more undisturbed and unmodified packet streams) to collect both Active and Passive Metrics and enable enhanced characterization from additional joint analysis. [I-D.trammell-ippm-hybrid-ps] presents a problem statement for Hybrid Type II methods and metrics. Note that one or more Hybrid Type I streams could be substituted for the Active streams or undisturbed streams in the mutually coordinated set. It is the Type II Methods where unique Hybrid Metrics are anticipated to emerge.

Methods based on a combination of a single (generated) Active stream and Passive observations applied to the stream of interest at intermediate observation points are also a type of Hybrid Methods. However, [RFC5644] already defines these as Spatial Metrics and Methods. It is possible to replace the Active stream of [RFC5644] with a Hybrid Type I stream and measure Spatial Metrics (but this was un-anticipated when [RFC5644] was developed).

The Table below illustrates the categorization of methods (where "Synthesis" refers to a combination of Active and Passive Method attributes).

	Single Stream of Interest	Multiple Simultaneous Streams of Interest from Different Methods
Single Fundamental Method	Active or Passive	
Synthesis of Fundamental Methods	Hybrid Type I	
Multiple Methods	Spatial Metrics [RFC5644]	Hybrid Type II

There may be circumstances where results measured with Hybrid Methods can be considered equivalent to Passive Methods. Referencing the notion of a "class C" where packets of different Type-P are treated

equally in Section 13 of [RFC2330]and the terminology for paths from Section 5 of [RFC2330]:

Hybrid Methods of Measurement that augment or modify packets of a "class C" in a host should produce equivalent results to Passive Methods of Measurement, when hosts accessing and links transporting these packets along the path (other than those performing augmentation/modification) treat packets from both categories of methods (with and without the augmentation/modification) as the same "class C". The Passive Methods of Measurement represent the Ground Truth for comparisons of results between Passive and Hybrid methods, and this comparison should be conducted to confirm the class C treatment.

4. Discussion

This section illustrates the definitions and presents some examples.

4.1. Graphical Representation

If we compare the Active and Passive Methods, there are at least two dimensions on which methods can be evaluated. This evaluation space may be useful when a method is a combination of the two alternative methods.

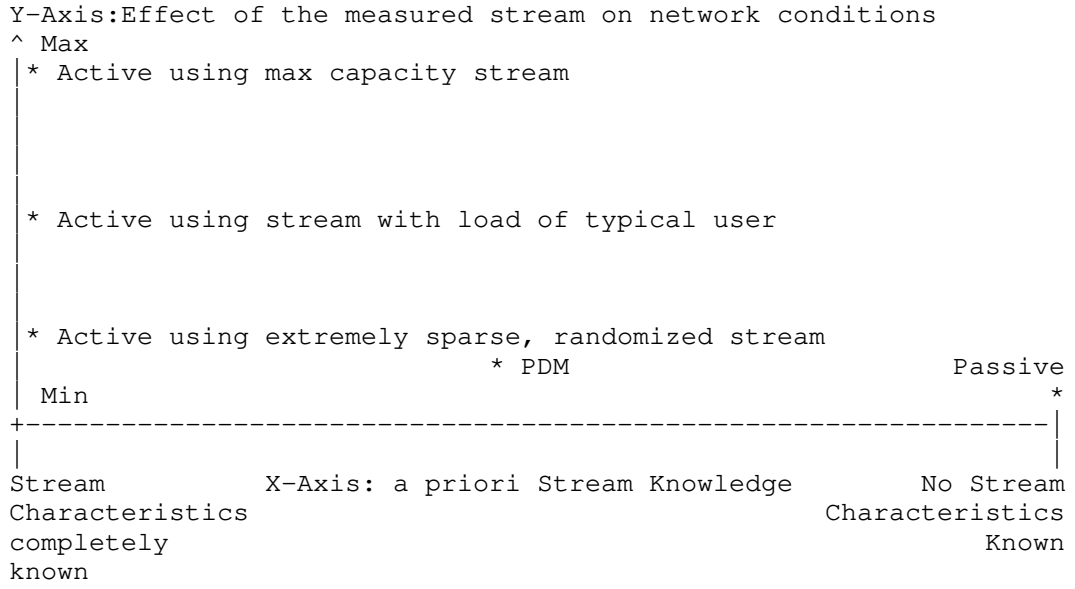
The two dimensions (initially chosen) are:

Y-Axis: "Effect of the measured stream on network conditions." The degree to which the stream of interest biases overall network conditions experienced by that stream and other streams. This is a key dimension for Active measurement error analysis. (Comment: There is also the notion of time averages - a measurement stream may have significant effect while it is present, but the stream is only generated 0.1% of the time. On the other hand, observations alone have no effect on network performance. To keep these dimensions simple, we consider the stream effect only when it is present, but note that reactive networks defined in [RFC7312] may exhibit bias for some time beyond the life of a stream.)

X-Axis: "a priori Stream Knowledge." The degree to which stream characteristics are known a priori. There are methodological advantages of knowing the source stream characteristics, and having complete control of the stream characteristics. For example, knowing the number of packets in a stream allows more efficient operation of the measurement receiver, and so is an asset for active measurement methods. Passive methods (with no sample filter) have few clues available to anticipate what the protocol first packet observed will use or how many packets will

comprise the flow, but once the standard protocol of a flow is known the possibilities narrow (for some compliant flows). Therefore this is a key dimension for Passive measurement error analysis.

There are a few examples we can plot on a two-dimensional space. We can anchor the dimensions with reference point descriptions.



(In the graph above, "PDM" refers to [I-D.ietf-ippm-6man-pdm-option], an IPv6 Option Header for Performance and Diagnostic Measurements, descrived in section 4.2.)

We recognize that method categorization could be based on additional dimensions, but this would require a different graphical approach.

For example, "effect of stream of interest on network conditions" could easily be further qualified into:

1. effect on the performance of the stream of interest itself: for example, choosing a packet marking or Differentiated Services Code Point (DSCP) resulting in domain treatment as a real-time stream (as opposed to default/best-effort marking).

2. effect on unmeasured streams that share the path and/or bottlenecks: for example, an extremely sparse measured stream of minimal size packets typically has little effect on other flows (and itself), while a stream designed to characterize path capacity may affect all other flows passing through the capacity bottleneck (including itself).

3. effect on network conditions resulting in network adaptation: for example, a network monitoring load and congestion conditions might change routing, placing some flows to alternate paths to mitigate the congestion.

We have combined 1 and 2 on the Y-axis, as examination of examples indicates strong correlation of effects in this pair, and network adaptation is not addressed.

It is apparent that different methods of IP network measurement can produce different results, even when measuring the same path at the same time. The two dimensions of the graph help to understand how the results might change with the method chosen. For example, an Active Method to assess throughput adds some amount of traffic to the network which might result in lower throughput for all streams. However, a Passive Method to assess throughput can also err on the low side due to unknown limitations of the hosts providing traffic, competition for host resources, limitations of the network interface, or private sub-networks that are not an intentional part of the path, etc. And Hybrid Methods could easily suffer from both forms of error. Another example of potential errors stems from the pitfalls of using an Active stream with known bias, such as a periodic stream defined in [RFC3432]. The strength of modelling periodic streams (like VoIP) is a potential weakness when extending the measured results to other application whose streams are non-periodic. The solutions are to model the application streams more exactly with an Active Method, or accept the risks and potential errors with the Passive Method discussed above.

4.2. Discussion of PDM

In [I-D.ietf-ippm-6man-pdm-option], an IPv6 Option Header for Performance and Diagnostic Measurements (PDM) is described which (when added to the stream of interest at strategic interfaces) supports performance measurements. This method processes a user traffic stream and adds "fields which are dedicated to measurement" (the measurement intent is made clear in the title of this option). Thus:

- The method intends to have a small effect on the measured stream and other streams in the network. There are conditions where this intent may not be realized.

- The measured stream has unknown characteristics until it is processed to add the PDM Option header. Note that if the packet MTU is exceeded after adding the header, the intent to have small effect will not be realized.

We conclude that this is a Hybrid Type I method, having at least one characteristic of both active and passive methods for a single stream of interest.

4.3. Discussion of "Coloring" Method

Draft [I-D.tempia-opsawg-p3m], proposed to color packets by re-writing a field of the stream at strategic interfaces to support performance measurements (noting that this is a difficult operation at an intermediate point on an encrypted Virtual Private Network). This method processes a user traffic stream and inserts "fields or values which are dedicated to measurement". Thus:

- The method intends to have a small effect on the measured stream and other streams in the network (smaller than PDM above). There are conditions where this intent may not be realized.

- The measured stream has unknown characteristics until it is processed to add the coloring in the header, and the stream could be measured and time-stamped during that process.

We note that [I-D.chen-ippm-coloring-based-ipfpm-framework] proposes a method similar to [I-D.tempia-opsawg-p3m], as ippm-list discussion revealed.

We conclude that this is a Hybrid Type I method, having at least one characteristic of both active and passive methods for a single stream of interest.

4.4. Brief Discussion of OAM Methods

Many Operations, Administration, and Management (OAM) methods exist beyond the IP-layer. For example, [Y.1731] defines several different measurement methods which we would classify as follows:

- Loss Measurement (LM) occasionally injects frames with a count of previous frames since the last LM message. We conclude LM is Hybrid Type I because

A. This method processes a user traffic stream,

B. and augments the stream of interest with frames having "fields which are dedicated to measurement".

- Synthetic Loss Measurement (SLM) and Delay Measurement (DM) methods both inject dedicated measurement frames, so the "stream of interest is generated as the basis of measurement". We conclude that SLM and DM methods are Active Methods.

We also recognize the existence of alternate terminology used in OAM at layers other than IP. Readers are encouraged to consult [RFC6374] for MPLS Loss and Delay measurement terminology, for example.

5. Security considerations

When considering security and privacy of those involved in measurement or those whose traffic is measured, there is sensitive information communicated and observed at observation and measurement points described above, and protocol issues to consider. We refer the reader to the security and privacy considerations described in the Large Scale Measurement of Broadband Performance (LMAP) Framework [RFC7594], which covers active and passive measurement techniques and supporting material on measurement context.

6. IANA Considerations

This memo makes no requests for IANA consideration.

7. Acknowledgements

Thanks to Mike Ackermann for asking the right question, and for several suggestions on terminology. Brian Trammell provided key terms and references for the passive category, and suggested ways to expand the Hybrid description and types. Phil Eardley suggested some hybrid scenarios for categorization as part of his review. Tiziano Ionta reviewed the draft and suggested the classification for the "coloring" method of measurement. Nalini Elkins identified several areas for clarification following her review. Bill Jouris, Stenio Fernandes, and Spencer Dawkins suggested several editorial improvements. Tal Mizrahi, Joachim Fabini, Greg Mirsky and Mike Ackermann raised many key considerations in their WGLC reviews, based on their broad measurement experience.

8. References

8.1. Normative References

[RFC2330] Paxson, V., Almes, G., Mahdavi, J., and M. Mathis, "Framework for IP Performance Metrics", RFC 2330, DOI 10.17487/RFC2330, May 1998, <http://www.rfc-editor.org/info/rfc2330>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3432] Raisanen, V., Grotefeld, G., and A. Morton, "Network performance measurement with periodic streams", RFC 3432, DOI 10.17487/RFC3432, November 2002, <http://www.rfc-editor.org/info/rfc3432>.

[RFC5644] Stephan, E., Liang, L., and A. Morton, "IP Performance Metrics (IPPM): Spatial and Multicast", RFC 5644, DOI 10.17487/RFC5644, October 2009, <http://www.rfc-editor.org/info/rfc5644>.

[RFC5835] Morton, A., Ed. and S. Van den Berghe, Ed., "Framework for Metric Composition", RFC 5835, DOI 10.17487/RFC5835, April 2010, <http://www.rfc-editor.org/info/rfc5835>.

[RFC6390] Clark, A. and B. Claise, "Guidelines for Considering New Performance Metric Development", BCP 170, RFC 6390, DOI 10.17487/RFC6390, October 2011, <http://www.rfc-editor.org/info/rfc6390>.

[RFC7011] Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, DOI 10.17487/RFC7011, September 2013, <http://www.rfc-editor.org/info/rfc7011>.

[RFC7312] Fabini, J. and A. Morton, "Advanced Stream and Sampling Framework for IP Performance Metrics (IPPM)", RFC 7312, DOI 10.17487/RFC7312, August 2014, <http://www.rfc-editor.org/info/rfc7312>.

[RFC7594] Eardley, P., Morton, A., Bagnulo, M., Burbridge, T., Aitken, P., and A. Akhter, "A Framework for Large-Scale Measurement of Broadband Performance (LMAP)", RFC 7594, DOI 10.17487/RFC7594, September 2015, <http://www.rfc-editor.org/info/rfc7594>.

8.2. Informative References

[RFC6374] Frost, D. and S. Bryant, "Packet Loss and Delay Measurement for MPLS Networks", RFC 6374, DOI 10.17487/RFC6374, September 2011, <http://www.rfc-editor.org/info/rfc6374>.

[I-D.morton-ippm-2330-stdform-typep] Morton, A., Fabini, J., Elkins, N., mackermann@bcbsm.com, m., and V. Hegde, "IP Options and IPv6 Updates for IPPM's Active Metric Framework: Packets of Type-P and Standard-Formed Packets", draft-morton-ippm-2330-stdform-typep-02 (work in progress), December 2015.

[I-D.ietf-ippm-6man-pdm-option] Elkins, N. and M. Ackermann, "IPv6 Performance and Diagnostic Metrics (PDM) Destination Option", draft-ietf-ippm-6man-pdm-option-01 (work in progress), October 2015.

[I-D.tempia-opsawg-p3m] Capello, A., Cociglio, M., Castaldelli, L., and A. Bonda, "A packet based method for passive performance monitoring", draft-tempia-opsawg-p3m-04 (work in progress), February 2014.

[I-D.chen-ippm-coloring-based-ipfpm-framework] Chen, M., Zheng, L., Mirsky, G., and G. Fioccola, "IP Flow Performance Measurement Framework", draft-chen-ippm-coloring-based-ipfpm-framework-04 (work in progress), July 2015.

[I-D.zheng-ippm-framework-passive] Zheng, L., Elkins, N., Lingli, D., Ackermann, M., and G. Mirsky, "Framework for IP Passive Performance Measurements", draft-zheng-ippm-framework-passive-03 (work in progress), February 2015.

[I-D.trammell-ippm-hybrid-ps] Trammell, B., Zheng, L., Berenguer, S., and M. Bagnulo, "Hybrid Measurement using IPPM Metrics", draft-trammell-ippm-hybrid-ps-01 (work in progress), February 2014.

[Y.1540] ITU-T Recommendation Y.1540, , "Internet protocol data communication service - IP packet transfer and availability performance parameters", March 2011.

[Y.1731] ITU-T Recommendation Y.1731, , "Operation, administration and management (OAM) functions and mechanisms for Ethernet-based networks", October 2015.

Author's Address

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

Email: acmorton@att.com

IP Performance Working Group
Internet-Draft
Intended status: Experimental
Expires: January 7, 2016

M. Mathis
Google, Inc
A. Morton
AT&T Labs
July 6, 2015

Model Based Metrics for Bulk Transport Capacity
draft-ietf-ippm-model-based-metrics-06.txt

Abstract

We introduce a new class of Model Based Metrics designed to assess if a complete Internet path can be expected to meet a predefined Bulk Transport Performance target by applying a suite of IP diagnostic tests to successive subpaths. The subpath-at-a-time tests can be robustly applied to key infrastructure, such as interconnects or even individual devices, to accurately detect if any part of the infrastructure will prevent any path traversing it from meeting the specified Target Transport Performance.

The IP diagnostic tests consist of precomputed traffic patterns and statistical criteria for evaluating packet delivery. The traffic patterns are precomputed to mimic TCP or other transport protocol over a long path but are constructed in such a way that they are independent of the actual details of the subpath under test, end systems or applications. Likewise the success criteria depends on the packet delivery statistics of the subpath, as evaluated against a protocol model applied to the Target Transport Performance. The success criteria also does not depend on the details of the subpath, end systems or application. This makes the measurements open loop, eliminating most of the difficulties encountered by traditional bulk transport metrics.

Model based metrics exhibit several important new properties not present in other Bulk Capacity Metrics, including the ability to reason about concatenated or overlapping subpaths. The results are vantage independent which is critical for supporting independent validation of tests results from multiple Measurement Points.

This document does not define IP diagnostic tests directly, but provides a framework for designing suites of IP diagnostics tests that are tailored to confirming that infrastructure can meet a predetermined Target Transport Performance.

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 January 7, 2016.

Copyright Notice

Table of Contents

1. Introduction 5
1.1. Version Control 6
2. Overview 7
3. Terminology 9
4. Background 14
4.1. TCP properties 16
4.2. Diagnostic Approach 17
4.3. New requirements relative to RFC 2330 18
5. Common Models and Parameters 18
5.1. Target End-to-end parameters 18
5.2. Common Model Calculations 19
5.3. Parameter Derating 20
5.4. Test Preconditions 21
6. Traffic generating techniques 21
6.1. Paced transmission 21
6.2. Constant window pseudo CBR 23
6.3. Scanned window pseudo CBR 24
6.4. Concurrent or channelized testing 24
7. Interpreting the Results 25
7.1. Test outcomes 25
7.2. Statistical criteria for estimating run_length 27
7.3. Reordering Tolerance 29
8. Diagnostic Tests 29
8.1. Basic Data Rate and Packet Delivery Tests 30
8.1.1. Delivery Statistics at Paced Full Data Rate 30
8.1.2. Delivery Statistics at Full Data Windowed Rate 31
8.1.3. Background Packet Delivery Statistics Tests 31
8.2. Standing Queue Tests 31
8.2.1. Congestion Avoidance 33
8.2.2. Bufferbloat 33
8.2.3. Non excessive loss 33
8.2.4. Duplex Self Interference 34
8.3. Slowstart tests 34
8.3.1. Full Window slowstart test 35
8.3.2. Slowstart AQM test 35
8.4. Sender Rate Burst tests 35
8.5. Combined and Implicit Tests 36
8.5.1. Sustained Bursts Test 36
8.5.2. Streaming Media 37
9. An Example 38
10. Validation 40
11. Security Considerations 41
12. Acknowledgements 41
13. IANA Considerations 42
14. References 42
14.1. Normative References 42

14.2. Informative References 42
Appendix A. Model Derivations 44
A.1. Queueless Reno 45
Appendix B. Complex Queueing 46
Appendix C. Version Control 47
Authors' Addresses 47

1. Introduction

Model Based Metrics (MBM) rely on mathematical models to specify a targeted suite of IP diagnostic tests, designed to assess whether common transport protocols can be expected to meet a predetermined performance target over an Internet path. Each test in the Targeted Diagnostic Suite (TDS) measures some aspect of IP packet transfer that is required to meet the Target Transport Performance. For example a TDS may have separate diagnostic tests to verify that there is: sufficient IP capacity (rate); sufficient queue space to deliver typical transport bursts; and that the background packet loss ratio is small enough not to interfere with congestion control. Unlike other metrics which yield measures of network properties, Model Based Metrics nominally yield pass/fail evaluations of the ability of standard transport protocols to meet a specific performance objective over some network path.

This note describes the modeling framework to derive the IP diagnostic test parameters from the Target Transport Performance specified for TCP Bulk Transport Capacity. Model Based Metrics is an alternative to the approach described in [RFC3148]. In the future, other Model Based Metrics may cover other applications and transports, such as VoIP over RTP. In most cases the IP diagnostic tests can be implemented by combining existing IPPM metrics with additional controls for generating precomputed traffic patterns and statistical criteria for evaluating packet delivery.

This approach, mapping Target Transport Performance to a targeted diagnostic suite (TDS) of IP tests, solves some intrinsic problems with using TCP or other throughput maximizing protocols for measurement. In particular all throughput maximizing protocols (and TCP congestion control in particular) cause some level of congestion in order to detect when they have filled the network. This self inflicted congestion obscures the network properties of interest and introduces non-linear equilibrium behaviors that make any resulting measurements useless as metrics because they have no predictive value for conditions or paths other than that of the measurement itself. These problems are discussed at length in Section 4.

A targeted suite of IP diagnostic tests does not have such difficulties. They can be constructed such that they make strong statistical statements about path properties that are independent of the measurement details, such as vantage and choice of measurement points. Model Based Metrics bridge the gap between empirical IP measurements and expected TCP performance.

1.1. Version Control

RFC Editor: Please remove this entire subsection prior to publication.

Please send comments about this draft to ippm@ietf.org. See http://goo.gl/02tkD for more information including: interim drafts, an up to date todo list and information on contributing.

Formatted: Mon Jul 6 13:49:30 PDT 2015

Changes since -05 draft:

- Wordsmithing on sections overhauled in -05 draft.
- Reorganized the document:
  - Relocated subsection "Preconditions".
  - Relocated subsection "New Requirements relative to RFC 2330".
- Addressed nits and not so nits by Ruediger Geib. (Thanks!)
- Substantially tightened the entire definitions section.
- Many terminology changes, to better conform to other docs :
  - IP rate and IP capacity (following RFC 5136) replaces various forms of link data rate.
  - subpath replaces link.
  - target_window_size replaces target_pipe_size.
  - Implied Bottleneck IP Rate replaces effective bottleneck link rate.
  - Packet delivery statistics replaces delivery statistics.

Changes since -04 draft:

- The introduction was heavily overhauled: split into a separate introduction and overview.
- The new shorter introduction:
  - Is a problem statement;
  - This document provides a framework;
  - That it replaces TCP measurement by IP tests;
  - That the results are pass/fail.
- Added a diagram of the framework to the overview
- and introduces all of the elements of the framework.
- Renumbered sections, reducing the depth of some section numbers.
- Updated definitions to better agree with other documents:
  - Reordered section 2
  - Bulk [data] performance -> Bulk Transport Capacity, everywhere including the title.
  - loss rate and loss probability -> packet loss ratio
  - end-to-end path -> complete path
  - [end-to-end][target] performance -> Target Transport Performance

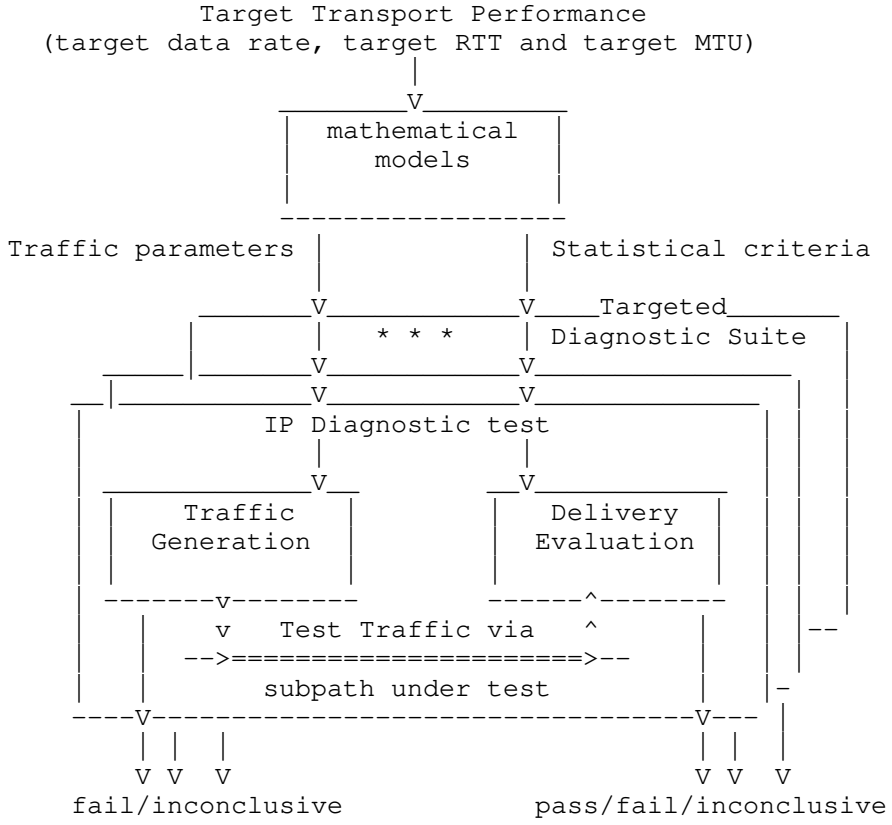
* load test -> capacity test

2. Overview

This document describes a modeling framework for deriving a Targeted Diagnostic Suite from a predetermined Target Transport Performance. It is not a complete specification, and relies on other standards documents to define important details such as packet type-p selection, sampling techniques, vantage selection, etc. We imagine Fully Specified Targeted Diagnostic Suites (FSTDS), that define all of these details. We use Targeted Diagnostic Suite (TDS) to refer to the subset of such a specification that is in scope for this document. This terminology is defined in Section 3.

Section 4 describes some key aspects of TCP behavior and what it implies about the requirements for IP packet delivery. Most of the IP diagnostic tests needed to confirm that the path meets these properties can be built on existing IPPM metrics, with the addition of statistical criteria for evaluating packet delivery and in a few cases, new mechanisms to implement precomputed traffic patterns. (One group of tests, the standing queue tests described in Section 8.2, don't correspond to existing IPPM metrics, but suitable metrics can be patterned after existing tools.)

Figure 1 shows the MBM modeling and measurement framework. The Target Transport Performance, at the top of the figure, is determined by the needs of the user or application, outside the scope of this document. For Bulk Transport Capacity, the main performance parameter of interest is the target data rate. However, since TCP's ability to compensate for less than ideal network conditions is fundamentally affected by the Round Trip Time (RTT) and the Maximum Transmission Unit (MTU) of the complete path, these parameters must also be specified in advance based on knowledge about the intended application setting. They may reflect a specific application over real path through the Internet or an idealized application and hypothetical path representing a typical user community. Section 5 describes the common parameters and models derived from the Target Transport Performance.



Overall Modeling Framework

Figure 1

The mathematical models are used to design traffic patterns that mimic TCP or other bulk transport protocol operating at the target data rate, MTU and RTT over a full range of conditions, including flows that are bursty at multiple time scales. The traffic patterns are generated based on the three target parameters of complete path and independent of the properties of individual subpaths using the techniques described in Section 6. As much as possible the measurement traffic is generated deterministically (precomputed) to minimize the extent to which test methodology, measurement points, measurement vantage or path partitioning affect the details of the measurement traffic.

Section 7 describes packet delivery statistics and methods test them against the bounds provided by the mathematical models. Since these statistics are typically the composition of subpaths of the complete

path [RFC6049] , in situ testing requires that the end-to-end statistical bounds be apportioned as separate bounds for each subpath. Subpaths that are expected to be bottlenecks may be expected to contribute a larger fraction of the total packet loss. In compensation, non-bottlenecked subpaths have to be constrained to contribute less packet loss. The criteria for passing each test of a TDS is an apportioned share of the total bound determined by the mathematical model from the Target Transport Performance.

Section 8 describes the suite of individual tests needed to verify all of required IP delivery properties. A subpath passes if and only if all of the individual IP diagnostics tests pass. Any subpath that fails any test indicates that some users are likely fail to attain their Target Transport Performance under some conditions. In addition to passing or failing, a test can be deemed to be inconclusive for a number of reasons including: the precomputed traffic pattern was not accurately generated; the measurement results were not statistically significant; and others such as failing to meet some required test preconditions. If all tests pass, except some are inconclusive then the entire suite is deemed to be inconclusive.

In Section 9 we present an example TDS that might be representative of HD video, and illustrate how Model Based Metrics can be used to address difficult measurement situations, such as confirming that intercarrier exchanges have sufficient performance and capacity to deliver HD video between ISPs.

Since there is some uncertainty in the modeling process, Section 10 describes a validation procedure to diagnose and minimize false positive and false negative results.

3. 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].

Note that terms containing underscores (rather than spaces) appear in equations in the modeling sections. In some cases both forms are used for aesthetic reasons, they do not have different meanings.

General Terminology:

Target: A general term for any parameter specified by or derived from the user's application or transport performance requirements.

Target Transport Performance: Application or transport performance goals for the complete path. For Bulk Transport Capacity defined in this note the Target Transport Performance includes the target data rate, target RTT and target MTU as described below.

Target Data Rate: The specified application data rate required for an application's proper operation. Conventional BTC metrics are focused on the target data rate, however these metrics had little or no predictive value because they do not consider the effects of the other two parameters of the Target Transport Performance, the RTT and MTU of the complete paths.

Target RTT (Round Trip Time): The specified baseline (minimum) RTT of the longest complete path over which the application expects to be able meet the target performance. TCP and other transport protocol's ability to compensate for path problems is generally proportional to the number of round trips per second. The Target RTT determines both key parameters of the traffic patterns (e.g. burst sizes) and the thresholds on acceptable traffic statistics. The Target RTT must be specified considering appropriate packets sizes: MTU sized packets on the forward path, ACK sized packets (typically header_overhead) on the return path. Note that target RTT is specified and not measured, it determines the applicability MBM evaluations for paths that are different than the measured path.

Target MTU (Maximum Transmission Unit): The specified maximum MTU supported by the complete path the over which the application expects to meet the target performance. Assume 1500 Byte MTU unless otherwise specified. If some subpath forces a smaller MTU, then it becomes the target MTU for the complete path, and all model calculations and subpath tests must use the same smaller MTU.

Targeted Diagnostic Suite (TDS): A set of IP diagnostic tests designed to determine if an otherwise ideal complete path containing the subpath under test can sustain flows at a specific target_data_rate using target_MTU sized packets when the RTT of the complete path is target_RTT.

Fully Specified Targeted Diagnostic Suite: A TDS together with additional specification such as "type-p", etc which are out of scope for this document, but need to be drawn from other standards documents.

Bulk Transport Capacity: Bulk Transport Capacity Metrics evaluate an Internet path's ability to carry bulk data, such as large files, streaming (non-real time) video, and under some conditions, web images and other content. Prior efforts to define BTC metrics have been based on [RFC3148], which predates our understanding of TCP ant the requirements described in Section 4

IP diagnostic tests: Measurements or diagnostic tests to determine if packet delivery statistics meet some precomputed target.

traffic patterns: The temporal patterns or statistics of traffic generated by applications over transport protocols such as TCP. There are several mechanisms that cause bursts at various time scales as described in Section 4.1. Our goal here is to mimic the range of common patterns (burst sizes and rates, etc), without tying our applicability to specific applications, implementations or technologies, which are sure to become stale.

packet delivery statistics: Raw, detailed or summary statistics about packet delivery properties of the IP layer including packet losses, ECN marks, reordering, or any other properties that may be germane to transport performance.

packet loss ratio: As defined in [I-D.ietf-ippm-2680-bis].

apportioned: To divide and allocate, for example budgeting packet loss across multiple subpaths such that they will accumulate to less than a specified end-to-end loss ratio.

open loop: A control theory term used to describe a class of techniques where systems that naturally exhibit circular dependencies can be analyzed by suppressing some of the dependencies, such that the resulting dependency graph is acyclic.

Terminology about paths, etc. See [RFC2330] and [RFC7398].

[data] sender: Host sending data and receiving ACKs.

[data] receiver: Host receiving data and sending ACKs.

complete path: The end-to-end path from the data sender to the data receiver.

subpath: A portion of the complete path. Note that there is no requirement that subpaths be non-overlapping. A subpath can be a small as a single device, link or interface.

Measurement Point: Measurement points as described in [RFC7398].

test path: A path between two measurement points that includes a subpath of the complete path under test, and if the measurement points are off path, may include "test leads" between the measurement points and the subpath.

[Dominant] Bottleneck: The Bottleneck that generally dominates packet delivery statistics for the entire path. It typically determines a flow's self clock timing, packet loss and ECN marking rate. See Section 4.1.

front path: The subpath from the data sender to the dominant bottleneck.

back path: The subpath from the dominant bottleneck to the receiver.

return path: The path taken by the ACKs from the data receiver to the data sender.

cross traffic: Other, potentially interfering, traffic competing for network resources (bandwidth and/or queue capacity).

Properties determined by the complete path and application. They are described in more detail in Section 5.1.

Application Data Rate: General term for the data rate as seen by the application above the transport layer in bytes per second. This is the payload data rate, and explicitly excludes transport and lower level headers (TCP/IP or other protocols), retransmissions and other overhead that is not part to the total quantity of data delivered to the application.

IP Rate: The actual number of IP-layer bytes delivered through a subpath, per unit time, including TCP and IP headers, retransmits and other TCP/IP overhead. Follows from IP-type-P Link Usage [RFC5136].

IP Capacity: The maximum number of IP-layer bytes that can be transmitted through a subpath, per unit time, including TCP and IP headers, retransmits and other TCP/IP overhead. Follows from IP-type-P Link Capacity [RFC5136].

Bottleneck IP Rate: This is the IP rate of the data flowing through the dominant bottleneck in the forward path. TCP and other protocols normally derive their self clocks from the timing of this data. See Section 4.1 and Appendix B for more details.

Implied Bottleneck IP Rate: This is the bottleneck IP rate implied by the returning ACKs from the receiver. It is determined by looking at how much application data the ACK stream reports delivered per unit time. If the return path is thinning, batching or otherwise altering ACK timing TCP will derive its clock from the the implied bottleneck IP rate of the ACK stream, which in the short term, might be much different than the actual bottleneck IP rate. In the case of thinned or batched ACKs front path must have sufficient buffering to smooth any data bursts to the IP capacity of the bottleneck. If the return path is not altering the ACK stream, then the Implied Bottleneck IP Rate will be the same as the Bottleneck IP Rate. See Section 4.1 and Appendix B for more details.

[sender | interface] rate: The IP rate which corresponds to the IP Capacity of the data sender's interface. Due to issues of sender efficiency and technologies such as TCP offload engines, nearly all moderns servers deliver data in bursts at full interface link rate. Today 1 or 10 Gb/s are typical.

Header_overhead: The IP and TCP header sizes, which are the portion of each MTU not available for carrying application payload. Without loss of generality this is assumed to be the size for returning acknowledgements (ACKs). For TCP, the Maximum Segment Size (MSS) is the Target MTU minus the header_overhead.

Basic parameters common to models and subpath tests are defined here are described in more detail in Section 5.2. Note that these are mixed between application transport performance (excludes headers) and IP performance (which include TCP headers and retransmissions as part of the payload).

Window: The total quantity of data plus the data represented by ACKs circulating in the network is referred to as the window. See Section 4.1. Sometimes used with other qualifiers (congestion window, cwnd or receiver window) to indicate which mechanism is controlling the window.

pipe size: A general term for number of packets needed in flight (the window size) to exactly fill some network path or subpath. It corresponds to the window size which maximizes network power, the observed data rate divided by the observed RTT. Often used with additional qualifiers to specify which path, or under what conditions, etc.

target_window_size: The average number of packets in flight (the window size) needed to meet the target data rate, for the specified target RTT, and MTU. It implies the scale of the bursts that the network might experience.

run length: A general term for the observed, measured, or specified number of packets that are (expected to be) delivered between losses or ECN marks. Nominally one over the sum of the loss and ECN marking probabilities, if there are independently and identically distributed.

target_run_length: The target_run_length is an estimate of the minimum number of non-congestion marked packets needed between losses or ECN marks necessary to attain the target_data_rate over a path with the specified target_RTT and target_MTU, as computed by a mathematical model of TCP congestion control. A reference calculation is shown in Section 5.2 and alternatives in Appendix A

reference target_run_length: target_run_length computed precisely by the method in Section 5.2. This is likely to be more slightly conservative than required by modern TCP implementations.

Ancillary parameters used for some tests:

derating: Under some conditions the standard models are too conservative. The modeling framework permits some latitude in relaxing or "derating" some test parameters as described in Section 5.3 in exchange for a more stringent TDS validation procedures, described in Section 10.

subpath_IP_capacity: The IP capacity of a specific subpath.

test path: A subpath of a complete path under test.
test_path_RTT: The RTT observed between two measurement points using packet sizes that are consistent with the transport protocol. Generally MTU sized packets of the forward path, header_overhead sized packets on the return path.
test_path_pipe: The pipe size of a test path. Nominally the test path RTT times the test path IP_capacity.
test_window: The window necessary to meet the target_rate over a test path. Typically test_window=target_data_rate*test_path_RTT/ (target_MTU - header_overhead).

The tests described in this note can be grouped according to their applicability.

capacity tests: determine if a network subpath has sufficient capacity to deliver the Target Transport Performance. As long as the test traffic is within the proper envelope for the Target Transport Performance, the average packet losses or ECN marks must be below the threshold computed by the model. As such, capacity tests reflect parameters that can transition from passing to failing as a consequence of cross traffic, additional presented load or the actions of other network users. By definition, capacity tests also consume significant network resources (data capacity and/or queue buffer space), and the test schedules must be balanced by their cost.
Monitoring tests: are designed to capture the most important aspects of a capacity test, but without presenting excessive ongoing load themselves. As such they may miss some details of the network's performance, but can serve as a useful reduced-cost proxy for a capacity test, for example to support ongoing monitoring.
Engineering tests: evaluate how network algorithms (such as AQM and channel allocation) interact with TCP-style self clocked protocols and adaptive congestion control based on packet loss and ECN marks. These tests are likely to have complicated interactions with cross traffic and under some conditions can be inversely sensitive to load. For example a test to verify that an AQM algorithm causes ECN marks or packet drops early enough to limit queue occupancy may experience a false pass result in the presence of cross traffic. It is important that engineering tests be performed under a wide range of conditions, including both in situ and bench testing, and over a wide variety of load conditions. Ongoing monitoring is less likely to be useful for engineering tests, although sparse in situ testing might be appropriate.

4. Background

At the time the IPPM WG was chartered, sound Bulk Transport Capacity

measurement was known to be well beyond our capabilities. Even at the time [RFC3148] was written we knew that we didn't fully understand the problem. Now, by hindsight we understand why BTC is such a hard problem:

- TCP is a control system with circular dependencies - everything affects performance, including components that are explicitly not part of the test.
- Congestion control is an equilibrium process, such that transport protocols change the network statistics (raise the packet loss ratio and/or RTT) to conform to their behavior. By design TCP congestion control keep raising the data rate until the network gives some indication that it is full by dropping or ECN marking packets. If TCP successfully fills the network the packet loss and ECN marks are mostly determined by TCP and how hard TCP drives the network and not by the network itself.
- TCP's ability to compensate for network flaws is directly proportional to the number of roundtrips per second (i.e. inversely proportional to the RTT). As a consequence a flawed subpath may pass a short RTT local test even though it fails when the path is extended by a perfect network to some larger RTT.
- TCP has an extreme form of the Heisenberg problem - Measurement and cross traffic interact in unknown and ill defined ways. The situation is actually worse than the traditional physics problem where you can at least estimate bounds on the relative momentum of the measurement and measured particles. For network measurement you can not in general determine the relative "mass" of either the measurement traffic or the cross traffic, so you can not gauge the relative magnitude of the uncertainty that might be introduced by any interaction.

These properties are a consequence of the equilibrium behavior intrinsic to how all throughput maximizing protocols interact with the Internet. These protocols rely on control systems based on estimated network parameters to regulate the quantity of data traffic sent into the network. The data traffic in turn alters network and the properties observed by the estimators, such that there are circular dependencies between every component and every property. Since some of these properties are nonlinear, the entire system is nonlinear, and any change anywhere causes difficult to predict changes in every parameter.

Model Based Metrics overcome these problems by forcing the measurement system to be open loop: the packet delivery statistics (akin to the network estimators) do not affect the traffic or traffic patterns (bursts), which computed on the basis of the Target Transport Performance. In order for a network to pass, the resulting packet delivery statistics and corresponding network estimators have to be such that they would not cause the control systems slow the

traffic below the target data rate.

4.1. TCP properties

TCP and SCTP are self clocked protocols. The dominant steady state behavior is to have an approximately fixed quantity of data and acknowledgements (ACKs) circulating in the network. The receiver reports arriving data by returning ACKs to the data sender, the data sender typically responds by sending exactly the same quantity of data back into the network. The total quantity of data plus the data represented by ACKs circulating in the network is referred to as the window. The mandatory congestion control algorithms incrementally adjust the window by sending slightly more or less data in response to each ACK. The fundamentally important property of this system is that it is self clocked: The data transmissions are a reflection of the ACKs that were delivered by the network, the ACKs are a reflection of the data arriving from the network.

A number of phenomena can cause bursts of data, even in idealized networks that can be modeled as simple queueing systems.

During slowstart the data rate is doubled on each RTT by sending twice as much data as was delivered to the receiver on the prior RTT. For slowstart to be able to fill such a network the network must be able to tolerate slowstart bursts up to the full pipe size inflated by the anticipated window reduction on the first loss or ECN mark. For example, with classic Reno congestion control, an optimal slowstart has to end with a burst that is twice the bottleneck rate for exactly one RTT in duration. This burst causes a queue which is exactly equal to the pipe size (i.e. the window is exactly twice the pipe size) so when the window is halved in response to the first loss, the new window will be exactly the pipe size.

Note that if the bottleneck data rate is significantly slower than the rest of the path, the slowstart bursts will not cause significant queues anywhere else along the path; they primarily exercise the queue at the dominant bottleneck.

Other sources of bursts include application pauses and channel allocation mechanisms. Appendix B describes the treatment of channel allocation systems. If the application pauses (stops reading or writing data) for some fraction of one RTT, state-of-the-art TCP catches up to the earlier window size by sending a burst of data at the full sender interface rate. To fill such a network with a realistic application, the network has to be able to tolerate interface rate bursts from the data sender large enough to cover application pauses.

Although the interface rate bursts are typically smaller than the last burst of a slowstart, they are at a higher data rate so they potentially exercise queues at arbitrary points along the front path from the data sender up to and including the queue at the dominant bottleneck. There is no model for how frequent or what sizes of sender rate bursts should be tolerated.

To verify that a path can meet a Target Transport Performance, it is necessary to independently confirm that the path can tolerate bursts in the dimensions that can be caused by these mechanisms. Three cases are likely to be sufficient:

- o Slowstart bursts sufficient to get connections started properly.
- o Frequent sender interface rate bursts that are small enough where they can be assumed not to significantly affect packet delivery statistics. (Implicitly derated by limiting the burst size).
- o Infrequent sender interface rate full target_window_size bursts that might affect the packet delivery statistics. (Target_run_length may be derated).

4.2. Diagnostic Approach

A complete path is expected to be able to sustain a Bulk TCP flow of a given data rate, MTU and RTT when all of the following conditions are met:

1. The IP capacity is above the target data rate by sufficient margin to cover all TCP/IP overheads. See Section 8.1 or any number of data rate tests outside of MBM.
2. The observed packet delivery statistics are better than required by a suitable TCP performance model (e.g. fewer losses or ECN marks). See Section 8.1 or any number of low rate packet loss tests outside of MBM.
3. There is sufficient buffering at the dominant bottleneck to absorb a slowstart rate burst large enough to get the flow out of slowstart at a suitable window size. See Section 8.3.
4. There is sufficient buffering in the front path to absorb and smooth sender interface rate bursts at all scales that are likely to be generated by the application, any channel arbitration in the ACK path or any other mechanisms. See Section 8.4.
5. When there is a slowly rising standing queue at the bottleneck the onset of packet loss has to be at an appropriate point (time or queue depth) and progressive. See Section 8.2.
6. When there is a standing queue at a bottleneck for a shared media subpath (e.g. half duplex), there are suitable bounds on how the data and ACKs interact, for example due to the channel arbitration mechanism. See Section 8.2.4.

Note that conditions 1 through 4 require capacity tests for

validation, and thus may need to be monitored on an ongoing basis. Conditions 5 and 6 require engineering tests best performed in controlled environments such as a bench test. They won't generally fail due to load, but may fail in the field due to configuration errors, etc. and should be spot checked.

We are developing a tool that can perform many of the tests described here [MBMSource].

4.3. New requirements relative to RFC 2330

Model Based Metrics are designed to fulfill some additional requirements that were not recognized at the time RFC 2330 was written [RFC2330]. These missing requirements may have significantly contributed to policy difficulties in the IP measurement space. Some additional requirements are:

- IP metrics must be actionable by the ISP - they have to be interpreted in terms of behaviors or properties at the IP or lower layers, that an ISP can test, repair and verify.
- Metrics should be spatially composable, such that measures of concatenated paths should be predictable from subpaths.
- Metrics must be vantage point invariant over a significant range of measurement point choices, including off path measurement points. The only requirements on MP selection should be that the RTT between the MPs is below some reasonable bound, and that the effects of the "test leads" connecting MPs to the subpath under test can be can be calibrated out of the measurements. The latter might be be accomplished if the test leads are effectively ideal or their properties can be deducted from the measurements between the MPs. While many of tests require that the test leads have at least as much IP capacity as the subpath under test, some do not, for example Background Packet Delivery Tests described in Section 8.1.3.
- Metric measurements must be repeatable by multiple parties with no specialized access to MPs or diagnostic infrastructure. It must be possible for different parties to make the same measurement and observe the same results. In particular it is specifically important that both a consumer (or their delegate) and ISP be able to perform the same measurement and get the same result. Note that vantage independence is key to meeting this requirement.

5. Common Models and Parameters

5.1. Target End-to-end parameters

The target end-to-end parameters are the target data rate, target RTT and target MTU as defined in Section 3. These parameters are

determined by the needs of the application or the ultimate end user and the complete Internet path over which the application is expected to operate. The target parameters are in units that make sense to upper layers: payload bytes delivered to the application, above TCP. They exclude overheads associated with TCP and IP headers, retransmits and other protocols (e.g. DNS).

Other end-to-end parameters defined in Section 3 include the effective bottleneck data rate, the sender interface data rate and the TCP and IP header sizes.

The target_data_rate must be smaller than all subpath IP capacities by enough headroom to carry the transport protocol overhead, explicitly including retransmissions and an allowance for fluctuations in TCP's actual data rate. Specifying a target_data_rate with insufficient headroom is likely to result in brittle measurements having little predictive value.

Note that the target parameters can be specified for a hypothetical path, for example to construct TDS designed for bench testing in the absence of a real application; or for a live in situ test of production infrastructure.

The number of concurrent connections is explicitly not a parameter to this model. If a subpath requires multiple connections in order to meet the specified performance, that must be stated explicitly and the procedure described in Section 6.4 applies.

5.2. Common Model Calculations

The Target Transport Performance is used to derive the target_window_size and the reference target_run_length.

The target_window_size, is the average window size in packets needed to meet the target_rate, for the specified target_RTT and target_MTU. It is given by:

target_window_size = ceiling(target_rate * target_RTT / (target_MTU - header_overhead))

Target_run_length is an estimate of the minimum required number of unmarked packets that must be delivered between losses or ECN marks, as computed by a mathematical model of TCP congestion control. The derivation here follows [MSMO97], and by design is quite conservative.

Reference target_run_length is derived as follows: assume the subpath_IP_capacity is infinitesimally larger than the

target_data_rate plus the required header_overhead. Then target_window_size also predicts the onset of queueing. A larger window will cause a standing queue at the bottleneck.

Assume the transport protocol is using standard Reno style Additive Increase, Multiplicative Decrease (AIMD) congestion control [RFC5681] (but not Appropriate Byte Counting [RFC3465]) and the receiver is using standard delayed ACKs. Reno increases the window by one packet every pipe_size worth of ACKs. With delayed ACKs this takes 2 Round Trip Times per increase. To exactly fill the pipe, losses must be no closer than when the peak of the AIMD sawtooth reached exactly twice the target_window_size otherwise the multiplicative window reduction triggered by the loss would cause the network to be underfilled. Following [MSMO97] the number of packets between losses must be the area under the AIMD sawtooth. They must be no more frequent than every 1 in ((3/2)*target_window_size)*(2*target_window_size) packets, which simplifies to:

target_run_length = 3*(target_window_size^2)

Note that this calculation is very conservative and is based on a number of assumptions that may not apply. Appendix A discusses these assumptions and provides some alternative models. If a different model is used, a fully specified TDS or FSTDS MUST document the actual method for computing target_run_length and ratio between alternate target_run_length and the reference target_run_length calculated above, along with a discussion of the rationale for the underlying assumptions.

These two parameters, target_window_size and target_run_length, directly imply most of the individual parameters for the tests in Section 8.

5.3. Parameter Derating

Since some aspects of the models are very conservative, the MBM framework permits some latitude in derating test parameters. Rather than trying to formalize more complicated models we permit some test parameters to be relaxed as long as they meet some additional procedural constraints:

- The TDS or FSTDS MUST document and justify the actual method used to compute the derated metric parameters.
- The validation procedures described in Section 10 must be used to demonstrate the feasibility of meeting the Target Transport Performance with infrastructure that infinitesimally passes the derated tests.

- o The validation process for a FSTDS itself must be documented is such a way that other researchers can duplicate the validation experiments.

Except as noted, all tests below assume no derating. Tests where there is not currently a well established model for the required parameters explicitly include derating as a way to indicate flexibility in the parameters.

5.4. Test Preconditions

Many tests have preconditions which are required to assure their validity. For example the presence or nonpresence of cross traffic on specific subpaths, or appropriate preloading to put reactive network elements into the proper states [RFC7312]. If preconditions are not properly satisfied for some reason, the tests should be considered to be inconclusive. In general it is useful to preserve diagnostic information about why the preconditions were not met, and any test data that was collected even if it is not useful for the intended test. Such diagnostic information and partial test data may be useful for improving the test in the future.

It is important to preserve the record that a test was scheduled, because otherwise precondition enforcement mechanisms can introduce sampling bias. For example, canceling tests due to cross traffic on subscriber access links might introduce sampling bias in tests of the rest of the network by reducing the number of tests during peak network load.

Test preconditions and failure actions MUST be specified in a FSTDS.

6. Traffic generating techniques

6.1. Paced transmission

Paced (burst) transmissions: send bursts of data on a timer to meet a particular target rate and pattern. In all cases the specified data rate can either be the application or IP rates. Header overheads must be included in the calculations as appropriate.

Packet Headway: Time interval between packets, specified from the start of one to the start of the next. e.g. If packets are sent with a 1 mS headway, there will be exactly 1000 packets per second.

Burst Headway: Time interval between bursts, specified from the start of the first packet one burst to the start of the first packet of the next burst. e.g. If 4 packet bursts are sent with a 1 mS burst headway, there will be exactly 4000 packets per second.

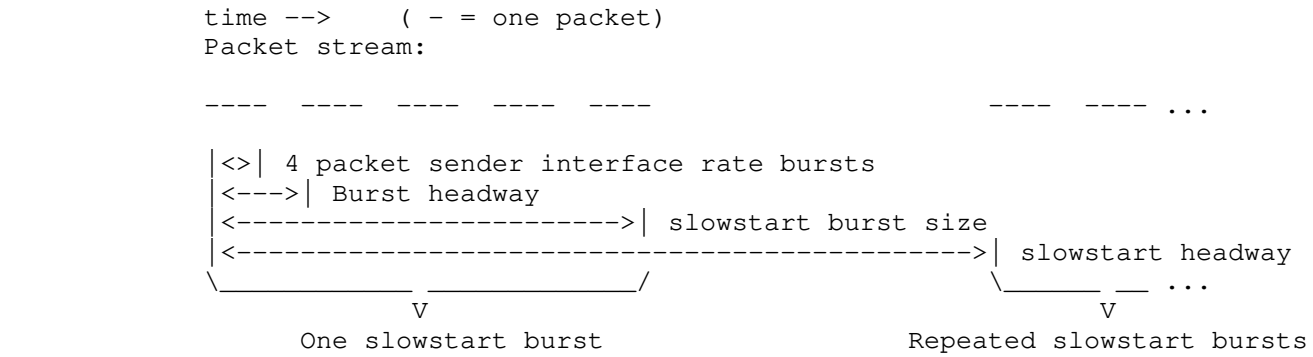
Paced single packets: Send individual packets at the specified rate or packet headway.

Paced Bursts: Send sender interface rate bursts on a timer. Specify any 3 of: average rate, packet size, burst size (number of packets) and burst headway (burst start to start). The packet headway within a burst is typically assumed to be the minimum supported by the tester's interface. i.e. Bursts are normally sent as back-to-back packets. The packet headway within the bursts can also be explicitly specified.

Slowstart burst: Mimic TCP slowstart by sending 4 packet paced bursts at an average data rate equal to twice the implied bottleneck IP rate (but not more than the sender interface rate). If the implied bottleneck IP rate is more than half of the sender interface rate, slowstart rate bursts become sender interface rate bursts. See the discussion and figure below.

Repeated Slowstart bursts: Repeat Slowstart bursts once per target_RTT. For TCP each burst would be twice as large as the prior burst, and the sequence would end at the first ECN mark or lost packet. For measurement, all slowstart bursts would be the same size (nominally target_window_size but other sizes might be specified). See the discussion and figure below.

The slowstart bursts mimic TCP slowstart under a particular set of implementation assumptions. The burst headway shown in Figure 2 reflects the TCP self clock derived from the data passing through the dominant bottleneck. The slow start burst size is nominally target_window_size (so it might end with a bust that is less than 4 packets). The slowstart bursts are repeated every target_RTT. Note that a stream of repeated slowstart bursts has three different average rates, depending on the averaging interval. At the finest time scale (a few packet times at the sender interface) the peak of the average rate is the same as the sender interface rate; at a medium scale (a few packet times at the dominant bottleneck) the peak of the average rate is twice the implied bottleneck IP rate; and at time scales longer than the target_RTT and when the burst size is equal to the target_window_size the average rate is equal to the target_data_rate. This pattern corresponds to repeating the last RTT of TCP slowstart when delayed ACK and sender side byte counting are present but without the limits specified in Appropriate Byte Counting [RFC3465].



Slowstart Burst Structure

Figure 2

Note that in conventional measurement practice, exponentially distributed intervals are often used to eliminate many sorts of correlations. For the procedures above, the correlations are created by the network or protocol elements and accurately reflect their behavior. At some point in the future, it will be desirable to introduce noise sources into the above pacing models, but they are not warranted at this time.

6.2. Constant window pseudo CBR

Implement pseudo constant bit rate by running a standard protocol such as TCP with a fixed window size, such that it is self clocked. Data packets arriving at the receiver trigger acknowledgements (ACKs) which travel back to the sender where they trigger additional transmissions. The window size is computed from the target_data_rate and the actual RTT of the test path. The rate is only maintained in average over each RTT, and is subject to limitations of the transport protocol.

Since the window size is constrained to be an integer number of packets, for small RTTs or low data rates there may not be sufficiently precise control over the data rate. Rounding the window size up (the default) is likely to be result in data rates that are higher than the target rate, but reducing the window by one packet may result in data rates that are too small. Also cross traffic potentially raises the RTT, implicitly reducing the rate. Cross traffic that raises the RTT nearly always makes the test more strenuous. A FSTDS specifying a constant window CBR tests MUST explicitly indicate under what conditions errors in the data cause tests to inconclusive.

Since constant window pseudo CBR testing is sensitive to RTT fluctuations it is less accurate at control the data rate in environments with fluctuating delays.

6.3. Scanned window pseudo CBR

Scanned window pseudo CBR is similar to the constant window CBR described above, except the window is scanned across a range of sizes designed to include two key events, the onset of queueing and the onset of packet loss or ECN marks. The window is scanned by incrementing it by one packet every 2*target_window_size delivered packets. This mimics the additive increase phase of standard TCP congestion avoidance when delayed ACKs are in effect. Normally the window increases separated by intervals slightly longer than twice the target_RTT.

There are two ways to implement this test: one built by applying a window clamp to standard congestion control in a standard protocol such as TCP and the other built by stiffening a non-standard transport protocol. When standard congestion control is in effect, any losses or ECN marks cause the transport to revert to a window smaller than the clamp such that the scanning clamp loses control the window size. The NPAD pathdiag tool is an example of this class of algorithms [Pathdiag].

Alternatively a non-standard congestion control algorithm can respond to losses by transmitting extra data, such that it maintains the specified window size independent of losses or ECN marks. Such a stiffened transport explicitly violates mandatory Internet congestion control and is not suitable for in situ testing. [RFC5681] It is only appropriate for engineering testing under laboratory conditions. The Windowed Ping tool implements such a test [WPING]. The tool described in the paper has been updated.[mpingSource]

The test procedures in Section 8.2 describe how to the partition the scans into regions and how to interpret the results.

6.4. Concurrent or channelized testing

The procedures described in this document are only directly applicable to single stream measurement, e.g. one TCP connection or measurement stream. In an ideal world, we would disallow all performance claims based multiple concurrent streams, but this is not practical due to at least two different issues. First, many very high rate link technologies are channelized and at last partially pin the flow to channel mapping to minimize packet reordering within flows. Second, TCP itself has scaling limits. Although the former problem might be overcome through different design decisions, the

later problem is more deeply rooted.

All congestion control algorithms that are philosophically aligned with the standard [RFC5681] (e.g. claim some level of TCP compatibility, friendliness or fairness) have scaling limits, in the sense that as a long fast network (LFN) with a fixed RTT and MTU gets faster, these congestion control algorithms get less accurate and as a consequence have difficulty filling the network[CCscaling]. These properties are a consequence of the original Reno AIMD congestion control design and the requirement in [RFC5681] that all transport protocols have similar responses to congestion.

There are a number of reasons to want to specify performance in term of multiple concurrent flows, however this approach is not recommended for data rates below several megabits per second, which can be attained with run lengths under 10000 packets on many paths. Since the required run length goes as the square of the data rate, at higher rates the run lengths can be unreasonably large, and multiple flows might be the only feasible approach.

If multiple flows are deemed necessary to meet aggregate performance targets then this MUST be stated both the design of the TDS and in any claims about network performance. The IP diagnostic tests MUST be performed concurrently with the specified number of connections. For the the tests that use bursty traffic, the bursts should be synchronized across flows.

7. Interpreting the Results

7.1. Test outcomes

To perform an exhaustive test of a complete network path, each test of the TDS is applied to each subpath of the complete path. If any subpath fails any test then a standard transport protocol running over the complete path can also be expected to fail to attain the Target Transport Performance under some conditions.

In addition to passing or failing, a test can be deemed to be inconclusive for a number of reasons. Proper instrumentation and treatment of inconclusive outcomes is critical to the accuracy and robustness of Model Based Metrics. Tests can be inconclusive if the precomputed traffic pattern or data rates were not accurately generated; the measurement results were not statistically significant; and others causes such as failing to meet some required preconditions for the test. See Section 5.4

For example consider a test that implements Constant Window Pseudo

CBR (Section 6.2) by adding rate controls and detailed traffic instrumentation to TCP (e.g. [RFC4898]). TCP includes built in control systems which might interfere with the sending data rate. If such a test meets the required packet delivery statistics (e.g. run length) while failing to attain the specified data rate it must be treated as an inconclusive result, because we can not a priori determine if the reduced data rate was caused by a TCP problem or a network problem, or if the reduced data rate had a material effect on the observed packet delivery statistics.

Note that for capacity tests, if the observed packet delivery statistics meet the statistical criteria for failing (accepting hypnosis H1 in Section 7.2), the test can can be considered to have failed because it doesn't really matter that the test didn't attain the required data rate.

The really important new properties of MBM, such as vantage independence, are a direct consequence of opening the control loops in the protocols, such that the test traffic does not depend on network conditions or traffic received. Any mechanism that introduces feedback between the paths measurements and the traffic generation is at risk of introducing nonlinearities that spoil these properties. Any exceptional event that indicates that such feedback has happened should cause the test to be considered inconclusive.

One way to view inconclusive tests is that they reflect situations where a test outcome is ambiguous between limitations of the network and some unknown limitation of the IP diagnostic test itself, which may have been caused by some uncontrolled feedback from the network.

Note that procedures that attempt to sweep the target parameter space to find the limits on some parameter such as target_data_rate are at risk of breaking the location independent properties of Model Based Metrics, if any part of the boundary between passing and inconclusive is sensitive to RTT (which is normally the case).

One of the goals for evolving TDS designs will be to keep sharpening distinction between inconclusive, passing and failing tests. The criteria for for passing, failing and inconclusive tests MUST be explicitly stated for every test in the TDS or FSTDS.

One of the goals of evolving the testing process, procedures, tools and measurement point selection should be to minimize the number of inconclusive tests.

It may be useful to keep raw packet delivery statistics and ancillary metrics [RFC3148] for deeper study of the behavior of the network path and to measure the tools themselves. Raw packet delivery

statistics can help to drive tool evolution. Under some conditions it might be possible to reevaluate the raw data for satisfying alternate Target Transport Performance. However it is important to guard against sampling bias and other implicit feedback which can cause false results and exhibit measurement point vantage sensitivity. Simply applying different delivery criteria based on a different Target Transport Performance is insufficient if the test traffic patterns (bursts, etc) does not match the alternate Target Transport Performance.

7.2. Statistical criteria for estimating run_length

When evaluating the observed run_length, we need to determine appropriate packet stream sizes and acceptable error levels for efficient measurement. In practice, can we compare the empirically estimated packet loss and ECN marking ratios with the targets as the sample size grows? How large a sample is needed to say that the measurements of packet transfer indicate a particular run length is present?

The generalized measurement can be described as recursive testing: send packets (individually or in patterns) and observe the packet delivery performance (packet loss ratio or other metric, any marking we define).

As each packet is sent and measured, we have an ongoing estimate of the performance in terms of the ratio of packet loss or ECN mark to total packets (i.e. an empirical probability). We continue to send until conditions support a conclusion or a maximum sending limit has been reached.

We have a target_mark_probability, 1 mark per target_run_length, where a "mark" is defined as a lost packet, a packet with ECN mark, or other signal. This constitutes the null Hypothesis:

H0: no more than one mark in target_run_length = 3*(target_window_size)^2 packets

and we can stop sending packets if on-going measurements support accepting H0 with the specified Type I error = alpha (= 0.05 for example).

We also have an alternative Hypothesis to evaluate: if performance is significantly lower than the target_mark_probability. Based on analysis of typical values and practical limits on measurement duration, we choose four times the H0 probability:

H1: one or more marks in (target_run_length/4) packets

and we can stop sending packets if measurements support rejecting H0 with the specified Type II error = beta (= 0.05 for example), thus preferring the alternate hypothesis H1.

H0 and H1 constitute the Success and Failure outcomes described elsewhere in the memo, and while the ongoing measurements do not support either hypothesis the current status of measurements is inconclusive.

The problem above is formulated to match the Sequential Probability Ratio Test (SPRT) [StatQC]. Note that as originally framed the events under consideration were all manufacturing defects. In networking, ECN marks and lost packets are not defects but signals, indicating that the transport protocol should slow down.

The Sequential Probability Ratio Test also starts with a pair of hypothesis specified as above:

H0: p0 = one defect in target_run_length
H1: p1 = one defect in target_run_length/4
As packets are sent and measurements collected, the tester evaluates the cumulative defect count against two boundaries representing H0 Acceptance or Rejection (and acceptance of H1):

Acceptance line: Xa = -h1 + s*n
Rejection line: Xr = h2 + s*n
where n increases linearly for each packet sent and

h1 = { log((1-alpha)/beta) }/k
h2 = { log((1-beta)/alpha) }/k
k = log{ (p1(1-p0)) / (p0(1-p1)) }
s = [log{ (1-p0)/(1-p1) }]/k
for p0 and p1 as defined in the null and alternative Hypotheses statements above, and alpha and beta as the Type I and Type II errors.

The SPRT specifies simple stopping rules:

- Xa < defect_count(n) < Xb: continue testing
- defect_count(n) <= Xa: Accept H0
- defect_count(n) >= Xb: Accept H1

The calculations above are implemented in the R-tool for Statistical Analysis [Rtool] , in the add-on package for Cross-Validation via Sequential Testing (CVST) [CVST] .

Using the equations above, we can calculate the minimum number of packets (n) needed to accept H0 when x defects are observed. For example, when x = 0:

Xa = 0 = -h1 + s*n
and n = h1 / s

7.3. Reordering Tolerance

All tests must be instrumented for packet level reordering [RFC4737]. However, there is no consensus for how much reordering should be acceptable. Over the last two decades the general trend has been to make protocols and applications more tolerant to reordering (see for example [RFC4015]), in response to the gradual increase in reordering in the network. This increase has been due to the deployment of technologies such as multi threaded routing lookups and Equal Cost MultiPath (ECMP) routing. These techniques increase parallelism in network and are critical to enabling overall Internet growth to exceed Moore's Law.

Note that transport retransmission strategies can trade off reordering tolerance vs how quickly they can repair losses vs overhead from spurious retransmissions. In advance of new retransmission strategies we propose the following strawman: Transport protocols should be able to adapt to reordering as long as the reordering extent is no more than the maximum of one quarter window or 1 mS, whichever is larger. Within this limit on reorder extent, there should be no bound on reordering density.

By implication, recording which is less than these bounds should not be treated as a network impairment. However [RFC4737] still applies: reordering should be instrumented and the maximum reordering that can be properly characterized by the test (e.g. bound on history buffers) should be recorded with the measurement results.

Reordering tolerance and diagnostic limitations, such as history buffer size, MUST be specified in a FSTDS.

8. Diagnostic Tests

The IP diagnostic tests below are organized by traffic pattern: basic data rate and packet delivery statistics, standing queues, slowstart bursts, and sender rate bursts. We also introduce some combined tests which are more efficient when networks are expected to pass, but conflate diagnostic signatures when they fail.

There are a number of test details which are not fully defined here.

They must be fully specified in a FSTDS. From a standardization perspective, this lack of specificity will weaken this version of Model Based Metrics, however it is anticipated that this it be more than offset by the extent to which MBM suppresses the problems caused by using transport protocols for measurement. e.g. non-specific MBM metrics are likely to have better repeatability than many existing BTC like metrics. Once we have good field experience, the missing details can be fully specified.

8.1. Basic Data Rate and Packet Delivery Tests

We propose several versions of the basic data rate and packet delivery statistics test. All measure the number of packets delivered between losses or ECN marks, using a data stream that is rate controlled at or below the target_data_rate.

The tests below differ in how the data rate is controlled. The data can be paced on a timer, or window controlled at full target data rate. The first two tests implicitly confirm that sub_path has sufficient raw capacity to carry the target_data_rate. They are recommend for relatively infrequent testing, such as an installation or periodic auditing process. The third, background packet delivery statistics, is a low rate test designed for ongoing monitoring for changes in subpath quality.

All rely on the receiver accumulating packet delivery statistics as described in Section 7.2 to score the outcome:

Pass: it is statistically significant that the observed interval between losses or ECN marks is larger than the target_run_length.

Fail: it is statistically significant that the observed interval between losses or ECN marks is smaller than the target_run_length.

A test is considered to be inconclusive if it failed to meet the data rate as specified below, meet the qualifications defined in Section 5.4 or neither run length statistical hypothesis was confirmed in the allotted test duration.

8.1.1. Delivery Statistics at Paced Full Data Rate

Confirm that the observed run length is at least the target_run_length while relying on timer to send data at the target_rate using the procedure described in in Section 6.1 with a burst size of 1 (single packets) or 2 (packet pairs).

The test is considered to be inconclusive if the packet transmission can not be accurately controlled for any reason.

RFC 6673 [RFC6673] is appropriate for measuring packet delivery statistics at full data rate.

8.1.2. Delivery Statistics at Full Data Windowed Rate

Confirm that the observed run length is at least the target_run_length while sending at an average rate approximately equal to the target_data_rate, by controlling (or clamping) the window size of a conventional transport protocol to a fixed value computed from the properties of the test path, typically test_window=target_data_rate*test_path_RTT/target_MTU. Note that if there is any interaction between the forward and return path, test_window may need to be adjusted slightly to compensate for the resulting inflated RTT.

Since losses and ECN marks generally cause transport protocols to at least temporarily reduce their data rates, this test is expected to be less precise about controlling its data rate. It should not be considered inconclusive as long as at least some of the round trips reached the full target_data_rate without incurring losses or ECN marks. To pass this test the network MUST deliver target_window_size packets in target_RTT time without any losses or ECN marks at least once per two target_window_size round trips, in addition to meeting the run length statistical test.

8.1.3. Background Packet Delivery Statistics Tests

The background run length is a low rate version of the target target rate test above, designed for ongoing lightweight monitoring for changes in the observed subpath run length without disrupting users. It should be used in conjunction with one of the above full rate tests because it does not confirm that the subpath can support raw data rate.

RFC 6673 [RFC6673] is appropriate for measuring background packet delivery statistics.

8.2. Standing Queue Tests

These engineering tests confirm that the bottleneck is well behaved across the onset of packet loss, which typically follows after the onset of queueing. Well behaved generally means lossless for transient queues, but once the queue has been sustained for a sufficient period of time (or reaches a sufficient queue depth) there should be a small number of losses to signal to the transport protocol that it should reduce its window. Losses that are too early can prevent the transport from averaging at the target_data_rate. Losses that are too late indicate that the queue might be subject to

bufferbloat [wikiBloat] and inflict excess queuing delays on all flows sharing the bottleneck queue. Excess losses (more than half of the window) at the onset of congestion make loss recovery problematic for the transport protocol. Non-linear, erratic or excessive RTT increases suggest poor interactions between the channel acquisition algorithms and the transport self clock. All of the tests in this section use the same basic scanning algorithm, described here, but score the link or subpath on the basis of how well it avoids each of these problems.

For some technologies the data might not be subject to increasing delays, in which case the data rate will vary with the window size all the way up to the onset of load induced losses or ECN marks. For theses technologies, the discussion of queueing does not apply, but it is still required that the onset of losses or ECN marks be at an appropriate point and progressive.

Use the procedure in Section 6.3 to sweep the window across the onset of queueing and the onset of loss. The tests below all assume that the scan emulates standard additive increase and delayed ACK by incrementing the window by one packet for every 2*target_window_size packets delivered. A scan can typically be divided into three regions: below the onset of queueing, a standing queue, and at or beyond the onset of loss.

Below the onset of queueing the RTT is typically fairly constant, and the data rate varies in proportion to the window size. Once the data rate reaches the subpath IP rate, the data rate becomes fairly constant, and the RTT increases in proportion to the increase in window size. The precise transition across the start of queueing can be identified by the maximum network power, defined to be the ratio data rate over the RTT. The network power can be computed at each window size, and the window with the maximum are taken as the start of the queueing region.

For technologies that do not have conventional queues, start the scan at a window equal to the test_window=target_data_rate*test_path_RTT/ target_MTU, i.e. starting at the target rate, instead of the power point.

If there is random background loss (e.g. bit errors, etc), precise determination of the onset of queue induced packet loss may require multiple scans. Above the onset of queuing loss, all transport protocols are expected to experience periodic losses determined by the interaction between the congestion control and AQM algorithms. For standard congestion control algorithms the periodic losses are likely to be relatively widely spaced and the details are typically dominated by the behavior of the transport protocol itself. For the

stiffened transport protocols case (with non-standard, aggressive congestion control algorithms) the details of periodic losses will be dominated by how the the window increase function responds to loss.

8.2.1. Congestion Avoidance

A subpath passes the congestion avoidance standing queue test if more than target_run_length packets are delivered between the onset of queueing (as determined by the window with the maximum network power) and the first loss or ECN mark. If this test is implemented using a standards congestion control algorithm with a clamp, it can be performed in situ in the production internet as a capacity test. For an example of such a test see [Pathdiag].

For technologies that do not have conventional queues, use the test_window inplace of the onset of queueing. i.e. A subpath passes the congestion avoidance standing queue test if more than target_run_length packets are delivered between start of the scan at test_window and the first loss or ECN mark.

8.2.2. Bufferbloat

This test confirms that there is some mechanism to limit buffer occupancy (e.g. that prevents bufferbloat). Note that this is not strictly a requirement for single stream bulk transport capacity, however if there is no mechanism to limit buffer queue occupancy then a single stream with sufficient data to deliver is likely to cause the problems described in [RFC2309], [I-D.ietf-aqm-recommendation] and [wikiBloat]. This may cause only minor symptoms for the dominant flow, but has the potential to make the subpath unusable for other flows and applications.

Pass if the onset of loss occurs before a standing queue has introduced more delay than than twice target_RTT, or other well defined and specified limit. Note that there is not yet a model for how much standing queue is acceptable. The factor of two chosen here reflects a rule of thumb. In conjunction with the previous test, this test implies that the first loss should occur at a queueing delay which is between one and two times the target_RTT.

Specified RTT limits that are larger than twice the target_RTT must be fully justified in the FSTDS.

8.2.3. Non excessive loss

This test confirm that the onset of loss is not excessive. Pass if losses are equal or less than the increase in the cross traffic plus the test traffic window increase on the previous RTT. This could be

restated as non-decreasing subpath throughput at the onset of loss, which is easy to meet as long as discarding packets is not more expensive than delivering them. (Note when there is a transient drop in subpath throughput, outside of a standing queue test, a subpath that passes other queue tests in this document will have sufficient queue space to hold one RTT worth of data).

Note that conventional Internet traffic policers will not pass this test, which is correct. TCP often fails to come into equilibrium at more than a small fraction of the available capacity, if the capacity is enforced by a policer. [Citation Pending].

8.2.4. Duplex Self Interference

This engineering test confirms a bound on the interactions between the forward data path and the ACK return path.

Some historical half duplex technologies had the property that each direction held the channel until it completely drained its queue. When a self clocked transport protocol, such as TCP, has data and ACKs passing in opposite directions through such a link, the behavior often reverts to stop-and-wait. Each additional packet added to the window raises the observed RTT by two forward path packet times, once as it passes through the data path, and once for the additional delay incurred by the ACK waiting on the return path.

The duplex self interference test fails if the RTT rises by more than some fixed bound above the expected queueing time computed from trom the excess window divided by the subpath IP Capacity. This bound must be smaller than target_RTT/2 to avoid reverting to stop and wait behavior. (e.g. Data packets and ACKs have to be released at least twice per RTT.)

8.3. Slowstart tests

These tests mimic slowstart: data is sent at twice the effective bottleneck rate to exercise the queue at the dominant bottleneck.

In general they are deemed inconclusive if the elapsed time to send the data burst is not less than half of the time to receive the ACKs. (i.e. sending data too fast is ok, but sending it slower than twice the actual bottleneck rate as indicated by the ACKs is deemed inconclusive). Space the bursts such that the average data rate is equal to the target_data_rate.

8.3.1. Full Window slowstart test

This is a capacity test to confirm that slowstart is not likely to exit prematurely. Send slowstart bursts that are target_window_size total packets.

Accumulate packet delivery statistics as described in Section 7.2 to score the outcome. Pass if it is statistically significant that the observed number of good packets delivered between losses or ECN marks is larger than the target_run_length. Fail if it is statistically significant that the observed interval between losses or ECN marks is smaller than the target_run_length.

Note that these are the same parameters as the Sender Full Window burst test, except the burst rate is at slowestart rate, rather than sender interface rate.

8.3.2. Slowstart AQM test

Do a continuous slowstart (send data continuously at slowstart_rate), until the first loss, stop, allow the network to drain and repeat, gathering statistics on the last packet delivered before the loss, the loss pattern, maximum observed RTT and window size. Justify the results. There is not currently sufficient theory justifying requiring any particular result, however design decisions that affect the outcome of this tests also affect how the network balances between long and short flows (the "mice and elephants" problem). The queue at the time of the first loss should be at least one half of the target_RTT.

This is an engineering test: It would be best performed on a quiescent network or testbed, since cross traffic has the potential to change the results.

8.4. Sender Rate Burst tests

These tests determine how well the network can deliver bursts sent at sender's interface rate. Note that this test most heavily exercises the front path, and is likely to include infrastructure may be out of scope for an access ISP, even though the bursts might be caused by ACK compression, thinning or channel arbitration in the access ISP. See Appendix B.

Also, there are a several details that are not precisely defined. For starters there is not a standard server interface rate. 1 Gb/s and 10 Gb/s are very common today, but higher rates will become cost effective and can be expected to be dominant some time in the future.

Current standards permit TCP to send a full window bursts following an application pause. (Congestion Window Validation [RFC2861], is not required, but even if was, it does not take effect until an application pause is longer than an RTO.) Since full window bursts are consistent with standard behavior, it is desirable that the network be able to deliver such bursts, otherwise application pauses will cause unwarranted losses. Note that the AIMD sawtooth requires a peak window that is twice target_window_size, so the worst case burst may be 2*target_window_size.

It is also understood in the application and serving community that interface rate bursts have a cost to the network that has to be balanced against other costs in the servers themselves. For example TCP Segmentation Offload (TSO) reduces server CPU in exchange for larger network bursts, which increase the stress on network buffer memory.

There is not yet theory to unify these costs or to provide a framework for trying to optimize global efficiency. We do not yet have a model for how much the network should tolerate server rate bursts. Some bursts must be tolerated by the network, but it is probably unreasonable to expect the network to be able to efficiently deliver all data as a series of bursts.

For this reason, this is the only test for which we encourage derating. A TDS could include a table of pairs of derating parameters: what burst size to use as a fraction of the target_window_size, and how much each burst size is permitted to reduce the run length, relative to to the target_run_length.

8.5. Combined and Implicit Tests

Combined tests efficiently confirm multiple network properties in a single test, possibly as a side effect of normal content delivery. They require less measurement traffic than other testing strategies at the cost of conflating diagnostic signatures when they fail. These are by far the most efficient for monitoring networks that are nominally expected to pass all tests.

8.5.1. Sustained Bursts Test

The sustained burst test implements a combined worst case version of all of the capacity tests above. It is simply:

Send target_window_size bursts of packets at server interface rate with target_RTT burst headway (burst start to burst start). Verify that the observed packet delivery statistics meets the target_run_length.

Key observations:

- The subpath under test is expected to go idle for some fraction of the time: (subpath_IP_capacity-target_rate/ (target_MTU-header_overhead)*target_MTU)/subpath_IP_capacity. Failing to do so indicates a problem with the procedure and an inconclusive test result.
- The burst sensitivity can be derated by sending smaller bursts more frequently. E.g. send target_window_size*derate packet bursts every target_RTT*derate.
- When not derated, this test is the most strenuous capacity test.
- A subpath that passes this test is likely to be able to sustain higher rates (close to subpath_IP_capacity) for paths with RTTs significantly smaller than the target_RTT.
- This test can be implemented with instrumented TCP [RFC4898], using a specialized measurement application at one end [MBMSource] and a minimal service at the other end [RFC0863] [RFC0864].
- This test is efficient to implement, since it does not require per-packet timers, and can make use of TSO in modern NIC hardware.
- This test by itself is not sufficient: the standing window engineering tests are also needed to ensure that the subpath is well behaved at and beyond the onset of congestion.
- Assuming the subpath passes relevant standing window engineering tests (particularly that it has a progressive onset of loss at an appropriate queue depth) the passing sustained burst test is (believed to be) a sufficient verify that the subpath will not impair stream at the target performance under all conditions. Proving this statement will be subject of ongoing research.

Note that this test is clearly independent of the subpath RTT, or other details of the measurement infrastructure, as long as the measurement infrastructure can accurately and reliably deliver the required bursts to the subpath under test.

8.5.2. Streaming Media

Model Based Metrics can be implicitly implemented as a side effect of serving any non-throughput maximizing traffic, such as streaming media, with some additional controls and instrumentation in the servers. The essential requirement is that the traffic be constrained such that even with arbitrary application pauses, bursts and data rate fluctuations, the traffic stays within the envelope defined by the individual tests described above.

If the application's serving_data_rate is less than or equal to the target_data_rate and the serving_RTT (the RTT between the sender and client) is less than the target_RTT, this constraint is most easily implemented by clamping the transport window size to be no larger than:

serving_window_clamp=target_data_rate*serving_RTT/
(target_MTU-header_overhead)

Under the above constraints the serving_window_clamp will limit the both the serving data rate and burst sizes to be no larger than the procedures in Section 8.1.2 and Section 8.4 or Section 8.5.1. Since the serving RTT is smaller than the target_RTT, the worst case bursts that might be generated under these conditions will be smaller than called for by Section 8.4 and the sender rate burst sizes are implicitly derated by the serving_window_clamp divided by the target_window_size at the very least. (Depending on the application behavior, the data traffic might be significantly smoother than specified by any of the burst tests.)

In an alternative implementation the data rate and bursts might be explicitly controlled by a host shaper or pacing at the sender. This would provide better control over transmissions but it is substantially more complicated to implement and would be likely to have a higher CPU overhead.

Note that these techniques can be applied to any content delivery that can be subjected to a reduced data rate in order to inhibit TCP equilibrium behavior.

9. An Example

In this section a we illustrate a TDS designed to confirm that an access ISP can reliably deliver HD video from multiple content providers to all of their customers. With modern codecs, minimal HD video (720p) generally fits in 2.5 Mb/s. Due to their geographical size, network topology and modem designs the ISP determines that most content is within a 50 mS RTT from their users (This is a sufficient to cover continental Europe or either US coast from a single serving site.)

2.5 Mb/s over a 50 ms path

End-to-End Parameter	value	units
target_rate	2.5	Mb/s
target_RTT	50	ms
target_MTU	1500	bytes
header_overhead	64	bytes
target_window_size	11	packets
target_run_length	363	packets

Table 1

Table 1 shows the default TCP model with no derating, and as such is quite conservative. The simplest TDS would be to use the sustained burst test, described in Section 8.5.1. Such a test would send 11 packet bursts every 50mS, and confirming that there was no more than 1 packet loss per 33 bursts (363 total packets in 1.650 seconds).

Since this number represents is the entire end-to-end loss budget, independent subpath tests could be implemented by apportioning the packet loss ratio across subpaths. For example 50% of the losses might be allocated to the access or last mile link to the user, 40% to the interconnects with other ISPs and 1% to each internal hop (assuming no more than 10 internal hops). Then all of the subpaths can be tested independently, and the spatial composition of passing subpaths would be expected to be within the end-to-end loss budget.

Testing interconnects has generally been problematic: conventional performance tests run between Measurement Points adjacent to either side of the interconnect, are not generally useful. Unconstrained TCP tests, such as iperf [iperf] are usually overly aggressive because the RTT is so small (often less than 1 mS). With a short RTT these tools are likely to report inflated numbers because for short RTTs these tools can tolerate very high packet loss ratios and can push other cross traffic off of the network. As a consequence they are useless for predicting actual user performance, and may themselves be quite disruptive. Model Based Metrics solves this problem. The same test pattern as used on other subpaths can be applied to the interconnect. For our example, when apportioned 40% of the losses, 11 packet bursts sent every 50mS should have fewer than one loss per 82 bursts (902 packets).

10. Validation

Since some aspects of the models are likely to be too conservative, Section 5.2 permits alternate protocol models and Section 5.3 permits test parameter derating. If either of these techniques are used, we require demonstrations that such a TDS can robustly detect subpaths that will prevent authentic applications using state-of-the-art protocol implementations from meeting the specified Target Transport Performance. This correctness criteria is potentially difficult to prove, because it implicitly requires validating a TDS against all possible subpaths and subpaths. The procedures described here are still experimental.

We suggest two approaches, both of which should be applied: first, publish a fully open description of the TDS, including what assumptions were used and and how it was derived, such that the research community can evaluate the design decisions, test them and comment on their applicability; and second, demonstrate that an applications running over an infinitessimally passing testbed do meet the performance targets.

An infinitessimally passing testbed resembles a epsilon-delta proof in calculus. Construct a test network such that all of the individual tests of the TDS pass by only small (infinitesimal) margins, and demonstrate that a variety of authentic applications running over real TCP implementations (or other protocol as appropriate) meets the Target Transport Performance over such a network. The workloads should include multiple types of streaming media and transaction oriented short flows (e.g. synthetic web traffic).

For example, for the HD streaming video TDS described in Section 9, the IP capacity should be exactly the header overhead above 2.5 Mb/s, the per packet random background loss ratio should be 1/363, for a run length of 363 packets, the bottleneck queue should be 11 packets and the front path should have just enough buffering to withstand 11 packet interface rate bursts. We want every one of the TDS tests to fail if we slightly increase the relevant test parameter, so for example sending a 12 packet bursts should cause excess (possibly deterministic) packet drops at the dominant queue at the bottleneck. On this infinitessimally passing network it should be possible for a real application using a stock TCP implementation in the vendor's default configuration to attain 2.5 Mb/s over an 50 mS path.

The most difficult part of setting up such a testbed is arranging for it to infinitesimally pass the individual tests. Two approaches: constraining the network devices not to use all available resources (e.g. by limiting available buffer space or data rate); and

preloading subpaths with cross traffic. Note that is it important that a single environment be constructed which infinitessimally passes all tests at the same time, otherwise there is a chance that TCP can exploit extra latitude in some parameters (such as data rate) to partially compensate for constraints in other parameters (queue space, or viceversa).

To the extent that a TDS is used to inform public dialog it should be fully publicly documented, including the details of the tests, what assumptions were used and how it was derived. All of the details of the validation experiment should also be published with sufficient detail for the experiments to be replicated by other researchers. All components should either be open source of fully described proprietary implementations that are available to the research community.

11. Security Considerations

Measurement is often used to inform business and policy decisions, and as a consequence is potentially subject to manipulation. Model Based Metrics are expected to be a huge step forward because equivalent measurements can be performed from multiple vantage points, such that performance claims can be independently validated by multiple parties.

Much of the acrimony in the Net Neutrality debate is due by the historical lack of any effective vantage independent tools to characterize network performance. Traditional methods for measuring Bulk Transport Capacity are sensitive to RTT and as a consequence often yield very different results when run local to an ISP or internconnect and when run over a customer's complete path. Neither the ISP nor customer can repeat the other's measurements, leading to high levels of distrust and acrimony. Model Based Metrics are expected to greatly improve this situation.

This document only describes a framework for designing Fully Specified Targeted Diagnostic Suite. Each FSTDS MUST include its own security section.

12. Acknowledgements

Ganga Maguluri suggested the statistical test for measuring loss probability in the target run length. Alex Gilgur for helping with the statistics.

Meredith Whittaker for improving the clarity of the communications.

Ruediger Geib provided feedback which greatly improved the document.

This work was inspired by Measurement Lab: open tools running on an open platform, using open tools to collect open data. See http://www.measurementlab.net/

13. IANA Considerations

This document has no actions for IANA.

14. References

14.1. Normative References

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

14.2. Informative References

[RFC0863] Postel, J., "Discard Protocol", STD 21, RFC 863, May 1983.

[RFC0864] Postel, J., "Character Generator Protocol", STD 22, RFC 864, May 1983.

[RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.

[RFC2330] Paxson, V., Almes, G., Mahdavi, J., and M. Mathis, "Framework for IP Performance Metrics", RFC 2330, May 1998.

[RFC2861] Handley, M., Padhye, J., and S. Floyd, "TCP Congestion Window Validation", RFC 2861, June 2000.

[RFC3148] Mathis, M. and M. Allman, "A Framework for Defining Empirical Bulk Transfer Capacity Metrics", RFC 3148, July 2001.

[RFC3465] Allman, M., "TCP Congestion Control with Appropriate Byte Counting (ABC)", RFC 3465, February 2003.

[RFC4015] Ludwig, R. and A. Gurtov, "The Eifel Response Algorithm

for TCP", RFC 4015, February 2005.

[RFC4737] Morton, A., Ciavattone, L., Ramachandran, G., Shalunov, S., and J. Perser, "Packet Reordering Metrics", RFC 4737, November 2006.

[RFC4898] Mathis, M., Heffner, J., and R. Raghunarayan, "TCP Extended Statistics MIB", RFC 4898, May 2007.

[RFC5136] Chimento, P. and J. Ishac, "Defining Network Capacity", RFC 5136, February 2008.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.

[RFC6049] Morton, A. and E. Stephan, "Spatial Composition of Metrics", RFC 6049, January 2011.

[RFC6673] Morton, A., "Round-Trip Packet Loss Metrics", RFC 6673, August 2012.

[RFC7312] Fabini, J. and A. Morton, "Advanced Stream and Sampling Framework for IP Performance Metrics (IPPM)", RFC 7312, August 2014.

[RFC7398] Bagnulo, M., Burbridge, T., Crawford, S., Eardley, P., and A. Morton, "A Reference Path and Measurement Points for Large-Scale Measurement of Broadband Performance", RFC 7398, February 2015.

[I-D.ietf-ippm-2680-bis]
Almes, G., Kalidindi, S., Zekauskas, M., and A. Morton, "A One-Way Loss Metric for IPPM", draft-ietf-ippm-2680-bis-02 (work in progress), June 2015.

[I-D.ietf-aqm-recommendation]
Baker, F. and G. Fairhurst, "IETF Recommendations Regarding Active Queue Management", draft-ietf-aqm-recommendation-11 (work in progress), February 2015.

[MSMO97] Mathis, M., Semke, J., Mahdavi, J., and T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", Computer Communications Review volume 27, number3, July 1997.

[WPING] Mathis, M., "Windowed Ping: An IP Level Performance Diagnostic", INET 94, June 1994.

[mpingSource]
Fan, X., Mathis, M., and D. Hamon, "Git Repository for mping: An IP Level Performance Diagnostic", Sept 2013, <https://github.com/m-lab/mping>.

[MBMSource]
Hamon, D., Stuart, S., and H. Chen, "Git Repository for Model Based Metrics", Sept 2013, <https://github.com/m-lab/MBM>.

[Pathdiag]
Mathis, M., Heffner, J., O'Neil, P., and P. Siemsen, "Pathdiag: Automated TCP Diagnosis", Passive and Active Measurement , June 2008.

[iperf] Wikipedia Contributors, "iPerf", Wikipedia, The Free Encyclopedia , cited March 2015, <http://en.wikipedia.org/w/index.php?title=Iperf&oldid=649720021>.

[StatQC] Montgomery, D., "Introduction to Statistical Quality Control - 2nd ed.", ISBN 0-471-51988-X, 1990.

[Rtool] R Development Core Team, "R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/", , 2011.

[CVST] Krueger, T. and M. Braun, "R package: Fast Cross-Validation via Sequential Testing", version 0.1, 11 2012.

[AFD] Pan, R., Breslau, L., Prabhakar, B., and S. Shenker, "Approximate fairness through differential dropping", SIGCOMM Comput. Commun. Rev. 33, 2, April 2003.

[wikiBloat]
Wikipedia, "Bufferbloat", http://en.wikipedia.org/w/index.php?title=Bufferbloat&oldid=608805474, March 2015.

[CCscaling]
Fernando, F., Doyle, J., and S. Steven, "Scalable laws for stable network congestion control", Proceedings of Conference on Decision and Control, http://www.ee.ucla.edu/~paganini, December 2001.

Appendix A. Model Derivations

The reference target_run_length described in Section 5.2 is based on

very conservative assumptions: that all window above target_window_size contributes to a standing queue that raises the RTT, and that classic Reno congestion control with delayed ACKs are in effect. In this section we provide two alternative calculations using different assumptions.

It may seem out of place to allow such latitude in a measurement standard, but this section provides offsetting requirements.

The estimates provided by these models make the most sense if network performance is viewed logarithmically. In the operational Internet, data rates span more than 8 orders of magnitude, RTT spans more than 3 orders of magnitude, and packet loss ratio spans at least 8 orders of magnitude if not more. When viewed logarithmically (as in decibels), these correspond to 80 dB of dynamic range. On an 80 dB scale, a 3 dB error is less than 4% of the scale, even though it represents a factor of 2 in untransformed parameter.

This document gives a lot of latitude for calculating target_run_length, however people designing a TDS should consider the effect of their choices on the ongoing tussle about the relevance of "TCP friendliness" as an appropriate model for Internet capacity allocation. Choosing a target_run_length that is substantially smaller than the reference target_run_length specified in Section 5.2 strengthens the argument that it may be appropriate to abandon "TCP friendliness" as the Internet fairness model. This gives developers incentive and permission to develop even more aggressive applications and protocols, for example by increasing the number of connections that they open concurrently.

A.1. Queueless Reno

In Section 5.2 it was assumed that the subpath IP rate matches the target rate plus overhead, such that the excess window needed for the AIMD sawtooth causes a fluctuating queue at the bottleneck.

An alternate situation would be bottleneck where there is no significant queue and losses are caused by some mechanism that does not involve extra delay, for example by the use of a virtual queue as in Approximate Fair Dropping [AFD]. A flow controlled by such a bottleneck would have a constant RTT and a data rate that fluctuates in a sawtooth due to AIMD congestion control. Assume the losses are being controlled to make the average data rate meet some goal which is equal or greater than the target_rate. The necessary run length can be computed as follows:

For some value of Wmin, the window will sweep from Wmin packets to 2*Wmin packets in 2*Wmin RTT (due to delayed ACK). Unlike the

queueing case where Wmin = target_window_size, we want the average of Wmin and 2*Wmin to be the target_window_size, so the average rate is the target rate. Thus we want Wmin = (2/3)*target_window_size.

Between losses each sawtooth delivers (1/2)(Wmin+2*Wmin)(2Wmin) packets in 2*Wmin round trip times.

Substituting these together we get:

target_run_length = (4/3)(target_window_size^2)

Note that this is 44% of the reference_run_length computed earlier. This makes sense because under the assumptions in Section 5.2 the AMID sawtooth caused a queue at the bottleneck, which raised the effective RTT by 50%.

Appendix B. Complex Queueing

For many network technologies simple queueing models don't apply: the network schedules, thins or otherwise alters the timing of ACKs and data, generally to raise the efficiency of the channel allocation when confronted with relatively widely spaced small ACKs. These efficiency strategies are ubiquitous for half duplex, wireless and broadcast media.

Altering the ACK stream generally has two consequences: it raises the implied bottleneck IP capacity, making slowstart burst at higher rates (possibly as high as the sender's interface rate) and it effectively raises the RTT by the average time that the ACKs and data were delayed. The first effect can be partially mitigated by reclocking ACKs once they are beyond the bottleneck on the return path to the sender, however this further raises the effective RTT.

The most extreme example of this sort of behavior would be a half duplex channel that is not released as long as end point currently holding the channel has more traffic (data or ACKs) to send. Such environments cause self clocked protocols under full load to revert to extremely inefficient stop and wait behavior, where they send an entire window of data as a single burst of the forward path, followed by the entire window of ACKs on the return path. It is important to note that due to self clocking, ill conceived channel allocation mechanisms can increase the stress on upstream subpaths in a long path: they cause large and faster bursts.

If a particular return path contains a subpath or device that alters the ACK stream, then the entire path from the sender up to the bottleneck must be tested at the burst parameters implied by the ACK

scheduling algorithm. The most important parameter is the Implied Bottleneck IP Capacity, which is the average rate at which the ACKs advance snd.una. Note that thinning the ACKs (relying on the cumulative nature of seg.ack to permit discarding some ACKs) is implies an effectively infinite Implied Bottleneck IP Capacity.

Holding data or ACKs for channel allocation or other reasons (such as forward error correction) always raises the effective RTT relative to the minimum delay for the path. Therefore it may be necessary to replace target_RTT in the calculation in Section 5.2 by an effective_RTT, which includes the target_RTT plus a term to account for the extra delays introduced by these mechanisms.

Appendix C. Version Control

This section to be removed prior to publication.

Formatted: Mon Jul 6 13:49:30 PDT 2015

Authors' Addresses

Matt Mathis
Google, Inc
1600 Amphitheater Parkway
Mountain View, California 94043
USA

Email: mattmathis@google.com

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

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 12, 2016

G. Mirsky
Ericsson
I. Meilik
Broadcom

February 9, 2016

Support of IEEE-1588 time stamp format in Two-Way Active Measurement Protocol (TWAMP) draft-mirsky-ippm-time-format-03

Abstract

This document describes an OPTIONAL feature for active performance measurement protocols allowing use of time stamp format defined in IEEE-1588v2-2008.

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 August 12, 2016.

Copyright Notice

Table of Contents

1. Introduction 2
1.1. Conventions used in this document 3
1.1.1. Terminology 3
1.1.2. Requirements Language 3
2. OWAMP and TWAMP Extensions 3
2.1. Timestamp Format Negotiation in Setting Up Connection in OWAMP 4
2.2. Timestamp Format Negotiation in Setting Up Connection in TWAMP 5
2.3. OWAMP-Test and TWAMP-Test Update 5
2.3.1. Consideration for TWAMP Light mode 6
3. IANA Considerations 6
4. Security Considerations 6
5. Acknowledgements 6
6. Normative References 7
Authors' Addresses 7

1. Introduction

One-Way Active Measurement Protocol (OWAMP) [RFC4656] defines that only the NTP [RFC5905] format of a time stamp can be used in OWAMP-Test protocol. Two-Way Active Measurement Protocol (TWAMP) [RFC5357] adopted the OWAMP-Test packet format and extended it by adding a format for a reflected test packet. Both the sender's and reflector's packets time stamps are expected to follow the 64-bit long NTP format [RFC5905]. NTP, when used over Internet, typically achieves clock accuracy of about 5ms to 100ms. Surveys conducted recently suggest that 90% devices achieve accuracy of better than 100 ms and 99% - better than 1 sec. It should be noted that NTP synchronizes clocks on the control plane, not on data plane. Distribution of clock wiitin a node may be supported by independent NTP domain or via interprocess communication in multiprocessor distributed system. And of mentioned solutions will be subject to additional queuing delays that negatively affect data plane clock accuracy.

Precision Time Protocol (PTP) [IEEE.1588.2008] has gained wide support since the development of OWAMP and TWAMP. PTP, using on-path support and other mechanisms, allows sub-microsecond clock accuracy. PTP is now supported in multiple implementations of fast forwarding engines and thus accuracy achieved by PTP is the accuracy of clock in data plane. Thus providing option to use more accurate clock as source of time stamps for IP performance measurement is one of advantages this proposal helps to achive. Another advantage realized by simplification of hardware in data plane. To support OWAMP or TWAMP test protocol time stamps must be converted from PTP to NTP.

That requires resources, use of micro-code or additional processing elements, that are always limited. To address this, this document proposes optional extensions to Control and Test protocols to support use of IEEE-1588v2 time stamp format as optional alternative to the NTP time stamp format.

One of the goals of this proposal is not only allow end-points of a test session to use other than NTP timestamp but to support backwards compatibility with nodes that do not yet support this extension.

1.1. Conventions used in this document

1.1.1. Terminology

IPPM: IP Performance Measurement

NTP: Network Time Protocol

PTP: Precision Time Protocol

TWAMP: Two-Way Active Measurement Protocol

OWAMP: One-Way Active Measurement Protocol

1.1.2. Requirements Language

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

2. OWAMP and TWAMP Extensions

OWAMP connection establishment follows the procedure defined in Section 3.1 of [RFC4656] and additional steps in TWAMP described in Section 3.1 of [RFC5357]. In these procedures the Modes field been used to identify and select specific communication capabilities. At the same time the Modes field been recognized and used as extension mechanism [RFC6038]. The new feature requires one bit position for Server and Control-Client to negotiate which timestamp format can be used in some or all test sessions invoked with this control connection. The end-point of the test session, Session-Sender and Session-Receiver or Session-Reflector, that supports this extension MUST be capable to interpret NTP and PTPv2 timestamp formats. If the end-point does not support this extension, then the value of PTPv2 Timestamp flag MUST be 0 because it is in Must Be Zero field. If value of PTPv2 Timestamp flags is 0, then the advertising node can use and interpret only NTP timestamp format.

Use of PTPv2 Timestamp flags discussed in the following sub-sections. For details on the assigned values and bit positions see the Section 3.

2.1. Timestamp Format Negotiation in Setting Up Connection in OWAMP

In OWAMP-Test [RFC4656] it is the Session-Receiver and/or Fetch-Client that are interpreting collected timestamps. Thus announced by a Server in the Modes field timestamp format indicates which formats the Session-Receiver is capable to interpret. The Control-Client inspects values set by the Server for timestamp formats and sets values in the Modes field of the Set-Up-Response message according to timestamp formats Session-Sender is capable of using. The rules of setting timestamp flags in Modes field in server greeting and Set-Up-Response messages and interpreting them are as follows:

- The Server that establishes test sessions for Session-Receiver that supports this extension MUST set PTPv2 Timestamp flag to 1 in the server greeting message according to the requirement listed in Section 2.

- If PTPv2 Timestamp flag of the server greeting message that the Control-Client receives has value 0, then the Session-Sender MUST use NTP format for timestamp in the test session and Control-Client SHOULD set PTPv2 Timestamp flag to 0 in accordance with [RFC4656]. If the Session-Sender cannot use NTP timestamps, then the Control-Client SHOULD close the TCP connection associated with the OWAMP-Control session.

- If the Session-Sender can set timestamp in PTPv2 format, then the Control-Client MUST set the PTPv2 Timestamp flag to 1in Modes field in the Set-Up-Response message and the Session-Sender MUST set timestamp in PTPv2 timestamp format. Otherwise the Control-Client MUST set the PTPv2 Timestamp flag in the Set-Up-Response message to 0.

- Otherwise, if the Session-Sender can set timestamp in NTP format, then the Session-Sender MUST set timestamp in NTP timestamp format. Otherwise the Control-Client SHOULD close the TCP connection associated with the OWAMP-Control session..

If values of both NTP and PTPv2 Timestamp flags in the Set-Up-Response message are equal to 0, then that indicates that the Control-Client can set timestamp only in NTP format.

If OWAMP-Control uses Fetch-Session commands, then selection and use of one or another timestamp format is local decision for both Session-Sender and Session-Receiver.

2.2. Timestamp Format Negotiation in Setting Up Connection in TWAMP

In TWAMP-Test [RFC5357] it is the Session-Sender that is interpreting collected timestamps. Hence, in the Modes field a Server advertises timestamp formats that the Session-Reflector can use in TWAMP-Test message. The choice of the timestamp format to be used by the Session-Sender is a local decision. The Control-Client inspects the Modes field and sets timestamp flags values to indicate which format will be used by the Session-Reflector. The rules of setting and interpreting flag values are as follows:

- Server MUST set to 1 value of PTPv2 Timestamp flag in its greeting message if Session-Reflector can set timestamp in PTPv2 format. Otherwise the PTPv2 Timestamp flag MUST be set to 0.

- If value of the PTPv2 Timestamp flag in received server greeting message equals 0, then Session-Reflector does not support this extension and will use NTP timestamp format. Control-Client SHOULD set PTPv2 Timestamp flag to 0 in Set-Up-Response message in accordance with [RFC5357].

- Control-Client MUST set PTPv2 Timestamp flag value to 1 in Modes field in the Set-Up-Response message if Server advertised ability of the Session-Reflector to use PTPv2 format for timestamps. Otherwise the flag MUST be set to 0.

- If the values of PTPv2 Timestamp flag in the Set-Up-Response message equals 0, then that means that Session-Sender can only interpret NTP timestamp format. Then the Session-Reflector MUST use NTP timestamp format. If the Session-Reflector does not support NTP format for timestamps then Server and SHOULD close the TCP connection associated with the TWAMP-Control session.

2.3. OWAMP-Test and TWAMP-Test Update

Participants of a test session need to indicate which timestamp format being used. The proposal is to use Z field in Error Estimate defined in Section 4.1.2 of [RFC4656]. The new interpretation of the Error Estimate is in addition to it specifying error estimate and synchronization, Error Estimate indicates format of a collected timestamp. And this proposal changes the semantics of the Z bit field, the one between S and Scale fields, to be referred as Timestamp format and value MUST be set according to the following:

- 0 - NTP 64 bit format of a timestamp;

- 1 - PTPv2 truncated format of a timestamp.

As result of this value of the Z field from Error Estimate, Sender Error Estimate or Send Error Estimate and Receive Error Estimate SHOULD NOT be ignored and MUST be used when calculating delay and delay variation metrics based on collected timestamps.

2.3.1. Consideration for TWAMP Light mode

This document does not specify how Session-Sender and Session-Reflector in TWAMP Light mode are informed of timestamp format to be used. It is assumed that, for example, configuration could be used to direct Session-Sender and Session-Reflector respectively to use timestamp format according to their capabilities and rules listed in Section 2.2.

3. IANA Considerations

The TWAMP-Modes registry defined in [RFC5618].

IANA is requested to reserve a new PTPv2 Timestamp as follows:

Value	Description	Semantics	Reference
TBA1 (proposed 256)	PTPv2 Timestamp Capability	bit position TBA2 (proposed 8)	This document

Table 1: New Timestamp Capability

4. Security Considerations

Use of particular format of a timestamp in test session does not appear to introduce any additional security threat to hosts that communicate with OWAMP and/or TWAMP as defined in [RFC4656], [RFC5357] respectively. The security considerations that apply to any active measurement of live networks are relevant here as well. See the Security Considerations sections in [RFC4656] and [RFC5357].

5. Acknowledgements

The authors would like to thank Lakshmikanthan and Suchit Bansal for their insightful suggestions. The authors would like to thank David Allan for his thorough review and thoughtful comments.

6. Normative References

[IEEE.1588.2008]
"Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems", IEEE Standard 1588, March 2008.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC4656] Shalunov, S., Teitelbaum, B., Karp, A., Boote, J., and M. Zekauskas, "A One-way Active Measurement Protocol (OWAMP)", RFC 4656, DOI 10.17487/RFC4656, September 2006, <http://www.rfc-editor.org/info/rfc4656>.

[RFC5357] Hedayat, K., Krzanowski, R., Morton, A., Yum, K., and J. Babiarz, "A Two-Way Active Measurement Protocol (TWAMP)", RFC 5357, DOI 10.17487/RFC5357, October 2008, <http://www.rfc-editor.org/info/rfc5357>.

[RFC5618] Morton, A. and K. Hedayat, "Mixed Security Mode for the Two-Way Active Measurement Protocol (TWAMP)", RFC 5618, DOI 10.17487/RFC5618, August 2009, <http://www.rfc-editor.org/info/rfc5618>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <http://www.rfc-editor.org/info/rfc5905>.

[RFC6038] Morton, A. and L. Ciavattone, "Two-Way Active Measurement Protocol (TWAMP) Reflect Octets and Symmetrical Size Features", RFC 6038, DOI 10.17487/RFC6038, October 2010, <http://www.rfc-editor.org/info/rfc6038>.

Authors' Addresses

Greg Mirsky
Ericsson

Email: gregory.mirsky@ericsson.com

Israel Meilik
Broadcom

Email: israel@broadcom.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 15, 2017

G. Mirsky
X. Min
ZTE Corp.
A. Pan
W. Luo
Ericsson

June 13, 2017

Two-Way Active Measurement Protocol (TWAMP) Light Data Model draft-mirsky-ippm-twamp-light-yang-09

Abstract

This document specifies the data model for implementations of Session-Sender and Session-Reflector for Two-Way Active Measurement Protocol (TWAMP) Light mode 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 December 15, 2017.

Copyright Notice

Table of Contents

1. Introduction 2
1.1. Conventions used in this document 2
1.1.1. Requirements Language 2
2. Scope, Model, and Applicability 3
2.1. Data Model Parameters 3
2.1.1. Session-Sender 3
2.1.2. Session-Reflector 4
3. Data Model 4
3.1. Tree Diagram 5
3.2. YANG Module 9
4. IANA Considerations 27
5. Security Considerations 28
6. References 28
6.1. Normative References 28
6.2. Informative References 29
Appendix A. Acknowledgements 29
Authors' Addresses 29

1. Introduction

The Two-Way Active Measurement Protocol (TWAMP) [RFC5357] can be used to measure performance parameters of IP networks such as latency, jitter, and packet loss by sending test packets and monitoring their experience in the network. The [RFC5357] defines two protocols, TWAMP Control and TWAMP Test, and a profile of TWAMP Test, TWAMP Light. The TWAMP Light is known to have many implementations though no common management framework being defined, thus leaving some aspects of test packet processing to interpretation. The goal of this document is to collect analyze these variations; describe common model while allowing for extensions in the future. This document defines such a TWAMP data model and specifies it formally using the YANG data modeling language [RFC6020].

1.1. Conventions used in this document

1.1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Scope, Model, and Applicability

The scope of this document includes model of the TWAMP Light as defined in Appendix I of [RFC5357]. This mode of TWAMP Light will be referred in this document as Stateless. Another mode, where the Session-Reflector is aware of the state of the TWAMP test session and thus can independently count reflected test packets, referred as Stateful. This document benefits from earlier attempt to define TWAMP MIB in [I-D.elteto-ippm-twamp-mib] and from TWAMP YANG model defined in [I-D.ietf-ippm-twamp-yang].

Figure 1 updates TWAMP-Light reference model presented in Appendix I [RFC5357] for the scenario when instantiation of a TWAMP-Test session between Session-Sender and Session-Reflector controlled by communication between a Configuration Client as a manager and Configuration Servers as agents of the configuration session.

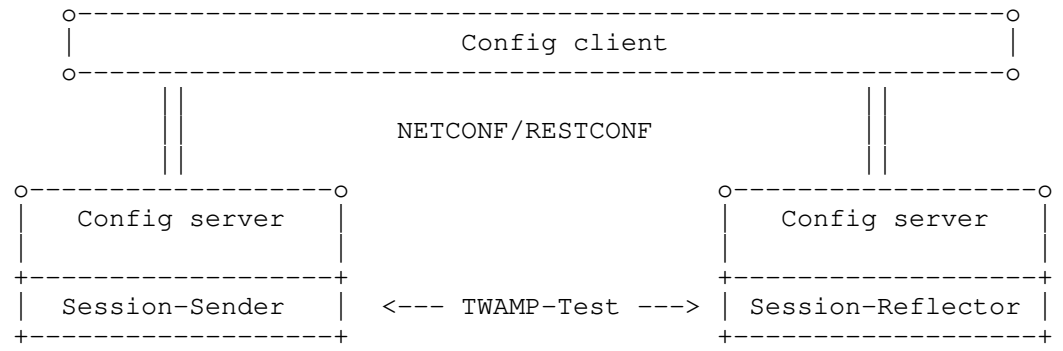


Figure 1: TWAMP Light Reference Model

2.1. Data Model Parameters

This section describes all the parameters of the the TWAMP-Light data model.

2.1.1. Session-Sender

The twamp-light-session-sender container holds items that are related to the configuration of the TWAMP-Light Session-Sender logical entity.

The twamp-light-session-sender-state container holds information about the state of the particular TWAMP-Light test session.

RPCs twamp-sender-start and twamp-sender-stop respectively start and stop the referenced by session-id TWAMP-Light test session.

2.1.1.1. Controls for Test Session and Preforrmance Metric Calculation

The data model supports several scenarios for Session-Sender to execute test sessions and calculate performance metrics:

The test mode in which the test packets are sent unbound in time at defined by the parameter 'interval' in the twamp-light-session-sender container frequency is referred as continuous mode. Performance metrics in the continuous mode are calculated at period defined by the parameter 'measurement-interval'.

The test mode that has specific number of the test packets configured for the test session in the 'number-of-packets' parameter is referred as periodic mode. The test session may be repeated by the Session-Sender with the same parameters. The 'repeat' parameter defines number of tests and the 'repeat-interval' - the interval between the consecuitive tests. The performance metrics are calculated after each test session when the interval defined by the 'session-timeout' expires.

2.1.2. Session-Reflector

The twamp-light-session-reflector container holds items that are related to the configuration of the TWAMP-Light Session-Reflector logical entity.

The twamp-light-session-refl-state container holds Session-Reflector state data for the particular TWAMP-Light test session.

3. Data Model

Creating TWAMP-Light data model presents number of challenges and among them is identification of a test-session at Session-Reflector. A Session-Reflector MAY require only as little as its IP and UDP port number in received TWAMP-Test packet to spawn new test session. More so, to test processing of Class-of-Service along the same route in Equal Cost Multi-Path environment Session-Sender may run TWAMP test sessions concurrently using the same source IP address, source UDP port number, destination IP address, and destination UDP port number. Thus the only parameter that can be used to differentiate these test sessions would be DSCP value. The DSCP field may get re-marked along the path and without use of [RFC7750] that will go undetected, but by using five-tuple instead of four-tuple as a key we can ensure that TWAMP test packets that are considered as different test sessions follow the same path even in ECMP environments.

3.1. Tree Diagram

```
module: ietf-twamp-light
    +--rw twamp-light
    |  +--rw twamp-light-session-sender {session-sender-light}?
    |  |  +--rw sender-light-enable?   enable
    |  |  +--rw test-session* [session-id]
    |  |     +--rw session-id                      uint32
    |  |     +--rw test-session-enable?            enable
    |  |     +--rw number-of-packets?              union
    |  |     +--rw packet-padding-size?            uint32
    |  |     +--rw interval?                       uint32
    |  |     +--rw session-timeout?                uint32
    |  |     +--rw measurement-interval?           uint32
    |  |     +--rw repeat?                         union
    |  |     +--rw repeat-interval?                uint32
    |  |     +--rw dscp-value?                     inet:dscp
    |  |     +--rw test-session-reflector-mode?    session-reflector-mode
    |  |     +--rw sender-ip                       inet:ip-address
    |  |     +--rw sender-udp-port                 inet:port-number
    |  |     +--rw reflector-ip                    inet:ip-address
    |  |     +--rw reflector-udp-port              inet:port-number
    |  |     +--rw authentication-params! {twamp-light-authentication}?
    |  |     |  +--rw key-chain?   kc:key-chain-ref
    |  |     +--rw first-percentile?               percentile
    |  |     +--rw second-percentile?              percentile
    |  |     +--rw third-percentile?               percentile
    |  +--rw twamp-light-session-reflector {session-reflector-light}?
    |     +--rw reflector-light-enable?       enable
    |     +--rw ref-wait?                     uint32
    |     +--rw reflector-light-mode-state?   session-reflector-mode
    |     +--rw test-session* [session-id]
    |        +--rw session-id                uint32
    |        +--rw dscp-handling-mode?       session-dscp-mode
    |        +--rw dscp-value?               inet:dscp
    |        +--rw sender-ip                 inet:ip-address
    |        +--rw sender-udp-port           inet:port-number
    |        +--rw reflector-ip              inet:ip-address
    |        +--rw reflector-udp-port        inet:port-number
    |        +--rw authentication-params! {twamp-light-authentication}?
    |           +--rw key-chain?   kc:key-chain-ref
    +--ro twamp-light-state
       +--ro twamp-light-session-sender-state {session-sender-light}?
       |  +--ro test-session-state* [session-id]
       |     +--ro session-id              uint32
       |     +--ro sender-session-state?   enumeration
       |     +--ro current-stats
```

```
        |       |  +--ro start-time                yang:date-and-time
        |       |  +--ro packet-padding-size?      uint32
        |       |  +--ro interval?                 uint32
        |       |  +--ro duplicate-packets?        uint32
        |       |  +--ro reordered-packets?        uint32
        |       |  +--ro sender-ip                 inet:ip-address
        |       |  +--ro sender-udp-port           inet:port-number
        |       |  +--ro reflector-ip              inet:ip-address
        |       |  +--ro reflector-udp-port        inet:port-number
        |       |  +--ro dscp?                     inet:dscp
        |       |  +--ro sent-packets?             uint32
        |       |  +--ro rcv-packets?              uint32
        |       |  +--ro sent-packets-error?       uint32
        |       |  +--ro rcv-packets-error?        uint32
        |       |  +--ro last-sent-seq?            uint32
        |       |  +--ro last-rcv-seq?             uint32
        |       |  +--ro two-way-delay
        |       |  |  +--ro delay
        |       |  |  |  +--ro min?   yang:gauge32
        |       |  |  |  +--ro max?   yang:gauge32
        |       |  |  |  +--ro avg?   yang:gauge32
        |       |  |  +--ro delay-variation
        |       |  |     +--ro min?   uint32
        |       |  |     +--ro max?   uint32
        |       |  |     +--ro avg?   uint32
        |       |  +--ro one-way-delay-far-end
        |       |  |  +--ro delay
        |       |  |  |  +--ro min?   yang:gauge32
        |       |  |  |  +--ro max?   yang:gauge32
        |       |  |  |  +--ro avg?   yang:gauge32
        |       |  |  +--ro delay-variation
        |       |  |     +--ro min?   uint32
        |       |  |     +--ro max?   uint32
        |       |  |     +--ro avg?   uint32
        |       |  +--ro one-way-delay-near-end
        |       |  |  +--ro delay
        |       |  |  |  +--ro min?   yang:gauge32
        |       |  |  |  +--ro max?   yang:gauge32
        |       |  |  |  +--ro avg?   yang:gauge32
        |       |  |  +--ro delay-variation
        |       |  |     +--ro min?   uint32
        |       |  |     +--ro max?   uint32
        |       |  |     +--ro avg?   uint32
        |       |  +--ro low-percentile
        |       |  |  +--ro delay-percentile
        |       |  |  |  +--ro rtt-delay?        percentile
        |       |  |  |  +--ro near-end-delay?   percentile
        |       |  |  |  +--ro far-end-delay?    percentile
```

```
|      |  |  +--ro jitter-percentile
|      |  |     +--ro rtt-jitter?        percentile
|      |  |     +--ro near-end-jitter?   percentile
|      |  |     +--ro far-end-jitter?    percentile
|      |  +--ro mid-percentile
|      |  |  +--ro delay-percentile
|      |  |  |  +--ro rtt-delay?        percentile
|      |  |  |  +--ro near-end-delay?   percentile
|      |  |  |  +--ro far-end-delay?    percentile
|      |  |  +--ro jitter-percentile
|      |  |     +--ro rtt-jitter?        percentile
|      |  |     +--ro near-end-jitter?   percentile
|      |  |     +--ro far-end-jitter?    percentile
|      |  +--ro high-percentile
|      |  |  +--ro delay-percentile
|      |  |  |  +--ro rtt-delay?        percentile
|      |  |  |  +--ro near-end-delay?   percentile
|      |  |  |  +--ro far-end-delay?    percentile
|      |  |  +--ro jitter-percentile
|      |  |     +--ro rtt-jitter?        percentile
|      |  |     +--ro near-end-jitter?   percentile
|      |  |     +--ro far-end-jitter?    percentile
|      |  +--ro two-way-loss
|      |  |  +--ro loss-count?         int32
|      |  |  +--ro loss-ratio?         percentage
|      |  |  +--ro loss-burst-max?     int32
|      |  |  +--ro loss-burst-min?     int32
|      |  |  +--ro loss-burst-count?   int32
|      |  +--ro one-way-loss-far-end
|      |  |  +--ro loss-count?         int32
|      |  |  +--ro loss-ratio?         percentage
|      |  |  +--ro loss-burst-max?     int32
|      |  |  +--ro loss-burst-min?     int32
|      |  |  +--ro loss-burst-count?   int32
|      |  +--ro one-way-loss-near-end
|      |     +--ro loss-count?         int32
|      |     +--ro loss-ratio?         percentage
|      |     +--ro loss-burst-max?     int32
|      |     +--ro loss-burst-min?     int32
|      |     +--ro loss-burst-count?   int32
|      +--ro history-stats* [id]
|         +--ro id                        uint32
|         +--ro end-time                  yang:date-and-time
|         +--ro number-of-packets?        uint32
|         +--ro packet-padding-size?      uint32
|         +--ro interval?                 uint32
|         +--ro duplicate-packets?        uint32
|         +--ro reordered-packets?        uint32
```

```
|         +--ro loss-packets?             uint32
|         +--ro sender-ip                 inet:ip-address
|         +--ro sender-udp-port           inet:port-number
|         +--ro reflector-ip              inet:ip-address
|         +--ro reflector-udp-port        inet:port-number
|         +--ro dscp?                     inet:dscp
|         +--ro sent-packets?             uint32
|         +--ro rcv-packets?              uint32
|         +--ro sent-packets-error?       uint32
|         +--ro rcv-packets-error?        uint32
|         +--ro last-sent-seq?            uint32
|         +--ro last-rcv-seq?             uint32
|         +--ro two-way-delay
|         |  +--ro delay
|         |  |  +--ro min?   yang:gauge32
|         |  |  +--ro max?   yang:gauge32
|         |  |  +--ro avg?   yang:gauge32
|         |  +--ro delay-variation
|         |     +--ro min?   uint32
|         |     +--ro max?   uint32
|         |     +--ro avg?   uint32
|         +--ro one-way-delay-far-end
|         |  +--ro delay
|         |  |  +--ro min?   yang:gauge32
|         |  |  +--ro max?   yang:gauge32
|         |  |  +--ro avg?   yang:gauge32
|         |  +--ro delay-variation
|         |     +--ro min?   uint32
|         |     +--ro max?   uint32
|         |     +--ro avg?   uint32
|         +--ro one-way-delay-near-end
|            +--ro delay
|            |  +--ro min?   yang:gauge32
|            |  +--ro max?   yang:gauge32
|            |  +--ro avg?   yang:gauge32
|            +--ro delay-variation
|               +--ro min?   uint32
|               +--ro max?   uint32
|               +--ro avg?   uint32
|
+--ro twamp-light-session-refl-state {session-reflector-light}?
   +--ro reflector-light-admin-status    boolean
   +--ro test-session-state* [session-id]
      +--ro session-id            uint32
      +--ro sent-packets?         uint32
      +--ro rcv-packets?          uint32
      +--ro sent-packets-error?   uint32
      +--ro rcv-packets-error?    uint32
      +--ro last-sent-seq?        uint32
```

```
             +--ro last-rcv-seq?         uint32
             +--ro sender-ip             inet:ip-address
             +--ro sender-udp-port       inet:port-number
             +--ro reflector-ip          inet:ip-address
             +--ro reflector-udp-port    inet:port-number

  rpcs:
    +---x twamp-sender-start
    |  +---w input
    |     +---w session-id    uint32
    +---x twamp-sender-stop
       +---w input
          +---w session-id    uint32
```

3.2. YANG Module

<CODE BEGINS> file "ietf-twamp-light@2017-06-13.yang"

```
module ietf-twamp-light {
   namespace "urn:ietf:params:xml:ns:yang:ietf-twamp-light";
    //namespace need to be assigned by IANA
   prefix "ietf-twamp-light";

   import ietf-inet-types {
     prefix inet;
   }
   import ietf-yang-types {
     prefix yang;
   }
   import ietf-key-chain {
     prefix kc;
   }

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

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

   description "TWAMP Light Data Model";

   revision "2017-06-13" {
     description
     "08 version. Appendix I RFC 5357 is covered.";
     reference "RFC 5357";
   }
```

```
   feature session-sender-light {
     description
     "This feature relates to the device functions as the
     TWAMP Light Session-Sender";
   }

   feature session-reflector-light {
     description
     "This feature relates to the device functions as the
     TWAMP Light Session-Reflector";
   }

   feature twamp-light-authentication {
     description
     "TWAMP Light authentication supported";
   }

   typedef enable {
      type boolean;
      description "enable";
   }

   typedef session-reflector-mode {
        type enumeration {
                enum stateful {
                        description
                        "When the Session-Reflector is stateful,
                        i.e. is aware of TWAMP-Test session state.";
                }
                enum stateless {
                        description
                        "When the Session-Reflector is stateless,
                        i.e. is not aware of the state of
                        TWAMP-Test session.";
                }
        }
        description "State of the Session-Reflector";
   }

   typedef session-dscp-mode {
        type enumeration {
                enum copy-received-value {
                        description
                        "Use DSCP value copied from received
                        TWAMP test packet of the test session.";
                }
                enum use-configured-value {
                        description
```

```
                        "Use DSCP value configured for this
                        test session on the Session-Reflector.";
                }
        }
        description
        "DSCP handling mode by Session-Reflector.";
    }

    typedef percentage {
        type decimal64 {
                fraction-digits 5;
        }
        description "Percentage";
    }

    typedef percentile {
        type decimal64 {
                fraction-digits 2;
        }
        description
        "Percentile is a measure used in statistics
        indicating the value below which a given
        percentage of observations in a group of
        observations fall.";
    }

    grouping maintenance-statistics {
      description "Maintenance statistics grouping";
      leaf sent-packets {
        type uint32;
        description "Packets sent";
      }
      leaf rcv-packets {
        type uint32;
        description "Packets received";
      }
      leaf sent-packets-error {
        type uint32;
        description "Packets sent error";
      }
      leaf rcv-packets-error {
        type uint32;
        description "Packets received error";
      }
      leaf last-sent-seq {
        type uint32;
        description "Last sent sequence number";
      }
```

```
     leaf last-rcv-seq {
       type uint32;
       description "Last received sequence number";
     }
   }

grouping twamp-session-percentile {
   description "Percentile grouping";
   leaf first-percentile {
      type percentile;
      default 95.00;
      description
      "First percentile to report";
   }
   leaf second-percentile {
      type percentile;
      default 99.00;
      description
      "Second percentile to report";
   }
   leaf third-percentile {
      type percentile;
      default 99.90;
      description
      "Third percentile to report";
   }
}

   grouping delay-statistics {
     description "Delay statistics grouping";
     container delay {
       description "Packets transmitted delay";
                 leaf min {
                   type yang:gauge32;
                   units microseconds;
                   description
                   "Min of Packets transmitted delay";
                 }
                 leaf max {
                    type yang:gauge32;
                    units microseconds;
                    description
                    "Max of Packets transmitted delay";
                 }
                 leaf avg {
                    type yang:gauge32;
                    units microseconds;
                    description
```

```
                   "Avg of Packets transmitted delay";
                 }
     }

          container delay-variation {
                  description
                  "Packets transmitted delay variation";
                  leaf min {
                    type uint32;
                    units microseconds;
                    description
                    "Min of Packets transmitted
                    delay variation";
                  }
                  leaf max {
                    type uint32;
                    units microseconds;
                    description
                    "Max of Packets transmitted
                    delay variation";
                  }
                  leaf avg {
                    type uint32;
                    units microseconds;
                    description
                    "Avg of Packets transmitted
                    delay variation";
                  }
           }
   }
     grouping time-percentile-report {
        description "Delay percentile report grouping";
        container delay-percentile {
                description
                "Report round-trip, near- and far-end delay";
                leaf rtt-delay {
                        type percentile;
                        description
                        "Percentile of round-trip delay";
                }
                leaf near-end-delay {
                        type percentile;
                        description
                        "Percentile of near-end delay";
                }
                leaf far-end-delay {
                        type percentile;
                        description
```

```
                        "Percentile of far-end delay";
                }
        }
        container jitter-percentile {
                description
                "Report round-trip, near- and far-end jitter";
                leaf rtt-jitter {
                        type percentile;
                        description
                        "Percentile of round-trip jitter";
                }
                leaf near-end-jitter {
                        type percentile;
                        description
                        "Percentile of near-end jitter";
                }
                leaf far-end-jitter {
                        type percentile;
                        description
                        "Percentile of far-end jitter";
                }
        }
    }

    grouping packet-loss-statistics {
        description
        "Grouping for Packet Loss statistics";
        leaf loss-count {
                type int32;
                description
                "Number of lost packets
                during the test interval.";
        }
        leaf loss-ratio {
                type percentage;
                description
                "Ratio of packets lost to packets
                sent during the test interval.";
        }
        leaf loss-burst-max {
                type int32;
                description
                "Maximum number of consequtively
                lost packets during the test interval.";
        }
        leaf loss-burst-min {
                type int32;
                description
```

```
                "Minimum number of consequtively
                lost packets during the test interval.";
        }
                leaf loss-burst-count {
                type int32;
                description
                "Number of occasions with packet
                loss during the test interval.";
                }
    }

   grouping session-light-parameters {
     description
     "Parameters common among
     Session-Sender and Session-Reflector";
     leaf sender-ip {
       type inet:ip-address;
       mandatory true;
       description "Sender IP address";
     }
     leaf sender-udp-port {
       type inet:port-number {
         range "49152..65535";
       }
       mandatory true;
       description "Sender UDP port number";
     }
     leaf reflector-ip {
       type inet:ip-address;
       mandatory true;
       description "Reflector IP address";
     }
     leaf reflector-udp-port {
       type inet:port-number{
         range "49152..65535";
       }
       mandatory true;
       description "Reflector UDP port number";
     }
   }

   grouping session-light-auth-params {
      description
      "Grouping for TWAMP Light authentication parameters";
      container authentication-params {
         if-feature twamp-light-authentication;
         presence "Enables TWAMP Light authentication";
         description
```

```
          "Parameters for TWAMP Light authentication";
          leaf key-chain {
             type kc:key-chain-ref;
             description "Name of key-chain";
          }
       }
    }

    /*Configuration Data*/
    container twamp-light {
      description
      "Top level container for TWAMP-Light configuration";

     container twamp-light-session-sender {
        if-feature session-sender-light;
        description "TWAMP-Light Session-Sender container";

        leaf sender-light-enable {
          type enable;
          default "true";
          description
          "Whether this network element is enabled to
          act as TWAMP-Light Sender";
        }

        list test-session {
          key "session-id";
                unique "sender-ip sender-udp-port reflector-ip"
          +" reflector-udp-port dscp-value";
          description
          "This structure is a container of test session
          managed objects";

          leaf session-id {
            type uint32;
            description "Session ID";
          }

          leaf test-session-enable {
            type enable;
            default "true";
            description
            "Whether this TWAMP Test session is enabled";
          }

          leaf number-of-packets {
            type union {
                 type uint32 {
```

```
                        range 1..4294967294 {
                                description
                                "The overall number of UDP test packet
                                to be transmitted by the sender for this
                                test session";
                        }
                }
                type enumeration {
                        enum forever {
                                description
                                "Indicates that the test session SHALL
                                be run *forever*.";
                        }
                }
          }
          default 10;
          description
          "This value determines if the TWAMP-Test session is
             bound by number of test packets or not.";
        }

        leaf packet-padding-size {
          type uint32;
          default 27;
          description
          "Size of the Packet Padding. Suggested to run
          Path MTU Discovery to avoid packet fragmentation in
          IPv4 and packet blackholing in IPv6";
        }

        leaf interval  {
          type uint32;
          units microseconds;
          description
          "Time interval between transmission of two
          consecutive packets in the test session in
          microseconds";
        }

           leaf session-timeout {
             when "../number-of-packets != 'forever'" {
               description
               "Test session timeout only valid if the
               test mode is periodic.";
          }
          type uint32;
          units "seconds";
          default 900;
```

```
          description
          "The timeout value for the Session-Sender to
          collect outstanding reflected packets.";
        }

        leaf measurement-interval {
          when "../number-of-packets = 'forever'" {
               description
               "Valid only when the test to run forever,
               i.e. continuously.";
          }
          type uint32;
          units "seconds";
          default 60;
          description
          "Interval to calculate performance metric when
                the test mode is 'continuous'.";
           }

           leaf repeat {
             type union {
               type uint32 {
                       range 0..4294967294;
               }
               type enumeration {
                       enum forever {
                               description
                               "Indicates that the test session SHALL
                               be repeated *forever* using the
                               information in repeat-interval
                               parameter, and SHALL NOT decrement
                               the value.";
                       }
               }
             }
             default 0;
             description
             "This value determines if the TWAMP-Test session must
             be repeated. When a test session has completed, the
             repeat parameter is checked. The default value
             of 0 indicates that the session MUST NOT be repeated.
             If the repeat value is 1 through 4,294,967,294
             then the test session SHALL be repeated using the
             information in repeat-interval parameter.
             The implementation MUST decrement the value of repeat
             after determining a repeated session is expected.";
           }
```

```
            leaf repeat-interval {
                when "../repeat != '0'";
                type uint32;
                units seconds;
                default 0;
                description
                "This parameter determines the timing of repeated
                TWAMP-Test sessions when repeat is more than 0.";
            }

            leaf dscp-value {
              type inet:dscp;
              default 0;
              description
              "DSCP value to be set in the test packet.";
            }

            leaf test-session-reflector-mode {
                 type session-reflector-mode;
                 default "stateless";
                 description
                 "The mode of TWAMP-Reflector for the test session.";
            }

          uses session-light-parameters;
          uses session-light-auth-params;
          uses twamp-session-percentile;
        }
      }

      container twamp-light-session-reflector {
        if-feature session-reflector-light;
        description
        "TWAMP-Light Session-Reflector container";
          leaf reflector-light-enable {
            type enable;
            default "true";
            description
            "Whether this network element is enabled to
            act as TWAMP-Light Reflector";
          }

          leaf ref-wait {
          type uint32 {
            range 1..604800;
          }
          units seconds;
          default 900;
```

```
        description
        "REFWAIT(TWAMP test session timeout in seconds),
        the default value is 900";
      }

      leaf reflector-light-mode-state {
        type session-reflector-mode;
        default stateless;
        description
        "The state of the mode of the TWAMP-Light
        Session-Reflector";
       }

       list test-session {
         key "session-id";
               unique "sender-ip sender-udp-port reflector-ip"
         +" reflector-udp-port";
         description
         "This structure is a container of test session
         managed objects";

       leaf session-id {
         type uint32;
         description "Session ID";
       }

       leaf dscp-handling-mode {
         type session-dscp-mode;
         default copy-received-value;
         description
         "Session-Reflector handling of DSCP:
                - use value copied from received TWAMP-Test packet;
                - use value explicitly configured";
          }

          leaf dscp-value {
            when "../dscp-handling-mode = 'use-configured-value'";
            type inet:dscp;
            default 0;
            description
            "DSCP value to be set in the reflected packet
               if dscp-handling-mode is set to use-configured-value.";
          }

        uses session-light-parameters;
        uses session-light-auth-params;
      }
    }
```

```
   }

 /*Operational state data nodes*/
 container twamp-light-state{
   config "false";
   description
   "Top level container for TWAMP-Light state data";

   container twamp-light-session-sender-state {
     if-feature session-sender-light;
     description
     "Session-Sender container for state data";
     list test-session-state{
       key "session-id";
       description
       "This structure is a container of test session
       managed objects";

       leaf session-id {
         type uint32;
         description "Session ID";
       }

       leaf sender-session-state {
         type enumeration {
           enum active {
             description "Test session is active";
           }
           enum ready {
             description "Test session is idle";
           }
         }
         description
         "State of the particular TWAMP-Light test
         session at the sender";
       }

       container current-stats {
         description
         "This container contains the results for the current
          Measurement Interval in a Measurement session ";
          leaf start-time {
            type yang:date-and-time;
            mandatory true;
            description
            "The time that the current Measurement Interval started";
          }
```

```
          leaf packet-padding-size {
            type uint32;
            default 27;
            description
            "Size of the Packet Padding. Suggested to run
            Path MTU Discovery to avoid packet fragmentation
            in IPv4 and packet backholing in IPv6";
          }

          leaf interval  {
            type uint32;
            units microseconds;
           description
           "Time interval between transmission of two
           consecutive packets in the test session";
          }

          leaf duplicate-packets  {
            type uint32;
           description "Duplicate packets";
          }
          leaf reordered-packets  {
            type uint32;
           description "Reordered packets";
          }

          uses session-light-parameters;
          leaf dscp {
            type inet:dscp;
            description
            "The DSCP value that was placed in the header of
            TWAMP UDP test packets by the Session-Sender.";
          }
          uses maintenance-statistics;

          container two-way-delay {
            description
            "two way delay result of the test session";
            uses delay-statistics;
          }

          container one-way-delay-far-end {
            description
            "one way delay far-end of the test session";
            uses delay-statistics;
          }

          container one-way-delay-near-end {
```

```
            description
            "one way delay near-end of the test session";
            uses delay-statistics;
          }

          container low-percentile {
                when "/twamp-light/twamp-light-session-sender/"
                +"test-session[session-id]/"
                +"first-percentile != '0.00'" {
                        description
                        "Only valid if the
                        the first-percentile is not NULL";
                }
                description
                "Low percentile report";
                uses time-percentile-report;
          }

          container mid-percentile {
                when "/twamp-light/twamp-light-session-sender/"
                +"test-session[session-id]/"
                +"second-percentile != '0.00'" {
                        description
                        "Only valid if the
                        the first-percentile is not NULL";
                }
                description
                "Mid percentile report";
                uses time-percentile-report;
          }

          container high-percentile {
                when "/twamp-light/twamp-light-session-sender/"
                +"test-session[session-id]/"
                +"third-percentile != '0.00'" {
                        description
                        "Only valid if the
                        the first-percentile is not NULL";
                }
                description
                "High percentile report";
                uses time-percentile-report;
          }

          container two-way-loss {
            description
            "two way loss count and ratio result of
            the test session";
```

```
          uses packet-loss-statistics;
        }
        container one-way-loss-far-end {
          when "/twamp-light/twamp-light-session-sender/"
          +"test-session[session-id]/"
          +"test-session-reflector-mode = 'stateful'" {
              description
              "One-way statistic is only valid if the
              session-reflector is in stateful mode.";
          }
          description
          "one way loss count and ratio far-end of
          the test session";
          uses packet-loss-statistics;
        }
        container one-way-loss-near-end {
          when "/twamp-light/twamp-light-session-sender/"
          +"test-session[session-id]/"
          +"test-session-reflector-mode = 'stateful'" {
              description
              "One-way statistic is only valid if the
              session-reflector is in stateful mode.";
          }
          description
          "one way loss count and ratio near-end of
          the test session";
          uses packet-loss-statistics;
        }
      }

      list history-stats {
        key id;
        description
        "This container contains the results for the history
         Measurement Interval in a Measurement session ";
         leaf id {
           type uint32;
           description
           "The identifier for the Measurement Interval
           within this session";
         }
         leaf end-time {
           type yang:date-and-time;
           mandatory true;
           description
           "The time that the Measurement Interval ended";
         }
         leaf number-of-packets {
```

```
            type uint32;
            description
            "The overall number of UDP test packets to be
            transmitted by the sender for this test session";
          }

          leaf packet-padding-size {
            type uint32;
            default 27;
            description
            "Size of the Packet Padding. Suggested to run
            Path MTU Discovery to avoid packet fragmentation
            in IPv4 and packet blackholing in IPv6";
          }

          leaf interval  {
            type uint32;
            units microseconds;
           description
           "Time interval between transmission of two
           consecutive packets in the test session";
          }
          leaf duplicate-packets  {
            type uint32;
           description "Duplicate packets";
          }
          leaf reordered-packets  {
            type uint32;
           description "Reordered packets";
          }
          leaf loss-packets  {
            type uint32;
           description "Loss packets";
          }

          uses session-light-parameters;
          leaf dscp {
            type inet:dscp;
            description
            "The DSCP value that was placed in the header of
            TWAMP UDP test packets by the Session-Sender.";
          }
          uses maintenance-statistics;

          container two-way-delay{
            description
            "two way delay result of the test session";
            uses delay-statistics;
```

```
          }
          container one-way-delay-far-end{
            description
            "one way delay far end of the test session";
            uses delay-statistics;
          }
          container one-way-delay-near-end{
            description
            "one way delay near end of the test session";
            uses delay-statistics;
          }
        }
      }
    }

     container twamp-light-session-refl-state {
       if-feature session-reflector-light;
       description
       "TWAMP-Light Session-Reflector container for
       state data";
       leaf reflector-light-admin-status {
         type boolean;
         mandatory "true";
         description
         "Whether this network element is enabled to
         act as TWAMP-Light Reflector";
       }

      list test-session-state {
        key "session-id";
        description
        "This structure is a container of test session
        managed objects";

        leaf session-id {
          type uint32;
          description "Session ID";
        }

        uses maintenance-statistics;
        uses session-light-parameters;
      }
    }
  }

  rpc twamp-sender-start {
    description
          "start the configured sender session";
```

```
   input {
         leaf session-id {
           type uint32;
           mandatory true;
           description
                 "The session to be started";
         }
   }
 }

 rpc twamp-sender-stop {
   description
         "stop the configured sender session";
   input {
         leaf session-id {
           type uint32;
           mandatory true;
           description
                 "The session to be stopped";
         }
   }
 }
}
```

<CODE ENDS>

4. 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-light

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-light

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

prefix: twamp

reference: RFC XXXX

5. Security Considerations

The configuration, state, action data defined in this document may be accessed via the NETCONF protocol [RFC6241]. SSH [RFC6242] is mandatory secure transport that is the lowest NETCONF layer. The NETCONF access control model [RFC6536] provides means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

But, in general, this TWAMP Light YANG module does not change any underlying security issues that already may exist in [I-D.elteto-ippm-twamp-mib].

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

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

[RFC5357] Hedayat, K., Krzanowski, R., Morton, A., Yum, K., and J. Babiarz, "A Two-Way Active Measurement Protocol (TWAMP)", RFC 5357, DOI 10.17487/RFC5357, October 2008, <http://www.rfc-editor.org/info/rfc5357>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc6536>.

[RFC7750] Hedin, J., Mirsky, G., and S. Baillargeon, "Differentiated Service Code Point and Explicit Congestion Notification Monitoring in the Two-Way Active Measurement Protocol (TWAMP)", RFC 7750, DOI 10.17487/RFC7750, February 2016, <http://www.rfc-editor.org/info/rfc7750>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <http://www.rfc-editor.org/info/rfc8174>.

6.2. 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-twamp-yang]
Civil, R., Morton, A., Rahman, R., Jethanandani, M., and K. Pentikousis, "Two-Way Active Measurement Protocol (TWAMP) Data Model", draft-ietf-ippm-twamp-yang-03 (work in progress), February 2017.

Appendix A. Acknowledgements

TBD

Authors' Addresses

Greg Mirsky
ZTE Corp.

Email: gregimirsky@gmail.com

Xiao Min
ZTE Corp.

Email: xiao.min2@zte.com.cn

Adrian Pan
Ericsson

Email: adrian.pan@ericsson.com

Wei S Luo
Ericsson

Email: wei.s.luo@ericsson.com

Network Working Group
Internet-Draft
Updates: 4656 (if approved)
Intended status: Standards Track
Expires: January 7, 2016

A. Morton
AT&T Labs

July 6, 2015

Registries for the One-Way Active Measurement Protocol - OWAMP
draft-morton-ippm-owamp-registry-01

Abstract

This memo describes the registries for OWAMP - the One-Way Active Measurement Protocol. The registries allow assignment of MODE bit positions and OWAMP Command numbers. The memo also requests that IANA establish the registries for new features, called the OWAMP-Modes registry and the OWAMP Control Command Number registry.

Requirements Language

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 RFC 2119 [RFC2119].

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 January 7, 2016.

Copyright Notice

Table of Contents

1. Introduction 2
2. Purpose and Scope 3
3. IANA Considerations for OWAMP Control Registries 3
3.1. Control Command Number Registry 3
3.1.1. Registry Specification 3
3.1.2. Registry Management 3
3.1.3. Experimental Numbers 4
3.1.4. OWAMP-Control Command Numbers Initial Contents . . . 4
3.2. OWAMP-Modes 4
3.2.1. Registry Specification 4
3.2.2. Registry Management 4
3.2.3. Experimental Numbers 5
3.2.4. OWAMP-Modes Initial Contents 5
4. Security Considerations 6
5. Acknowledgements 6
6. References 6
6.1. Normative References 6
6.2. Informative References 6
Author's Address 7

1. Introduction

The One-way Active Measurement Protocol, OWAMP [RFC4656] was prepared to support measurements of metrics specified by the IP Performance Metrics (IPPM) working group in the IETF. The Two-Way Active Measurement Protocol, TWAMP [RFC5357] is an extension of OWAMP. The

TWAMP specification gathered wide review as it approached completion, and the by-products were several recommendations for new features in TWAMP. As a result, a registry of new features was established for TWAMP. However, there were no new features proposed for OWAMP until recently.

This memo establishes the needed registries for OWAMP, and updates [RFC4656].

2. Purpose and Scope

The purpose and scope of this memo is to describe and request the establishment of registries for future OWAMP [RFC4656] extensions. IANA already administrates the "Two-way Active Measurement Protocol (TWAMP) Parameters", and this request follows a similar form (with one exception identified below).

This memo also provides the initial contents for the registries.

3. IANA Considerations for OWAMP Control Registries

OWAMP-Control protocol coordinates the measurement capability. All OWAMP-Control messages follow specifications defined in section 3 of [RFC4656].

3.1. Control Command Number Registry

IANA is requested to create a OWAMP-Control Command Number registry.

OWAMP-Control Commands follow specifications defined in section 3.4 of [RFC4656].

3.1.1. Registry Specification

OWAMP-Control Commands Numbers are specified in the first octet of OWAMP-Control-Client command messages consistent with section 3 of [RFC4656]. There are a maximum of 256 command numbers.

3.1.2. Registry Management

Because the "OWAMP-Control Command Numbers" registry can contain only 256 values, and because OWAMP is an IETF protocol, these registries must be updated only by "IETF Consensus" as specified in [RFC5226] (an RFC that documents registry use and is approved by the IESG).

3.1.3. Experimental Numbers

One experimental value is currently assigned in the Command Numbers Registry, as indicated in the initial contents below.

3.1.4. OWAMP-Control Command Numbers Initial Contents

OWAMP-Control Commands follows the procedure defined in section 3.5 of [RFC4656] (and in the remainder of section 3).

The complete set of OWAMP-Control Command Numbers are as follows (including one reserved bit position):

OWAMP-Control Command Numbers Registry

Value	Description	Semantics Definition
0	Reserved	
1	Request-Session	RFC 4656, Section 3.5
2	Start-Sessions	RFC 4656, Section 3.7
3	Stop-Sessions	RFC 4656, Section 3.8
4	Fetch-Sessions	RFC 4656, Section 3.9
5	Experimentation	This Memo
6-255	Unassigned	

3.2. OWAMP-Modes

IANA is requested to create a OWAMP-Modes registry.

3.2.1. Registry Specification

OWAMP-Modes are specified in OWAMP Server Greeting messages and Set-up Response messages consistent with section 3.1 of [RFC4656]. Modes are currently indicated by setting single bits in the 32-bit Modes Field. However, more complex encoding may be used in the future.

3.2.2. Registry Management

Because the "OWAMP-Modes" are based on only 32 bit positions with each position conveying a unique feature, and because TWAMP is an IETF protocol, these registries must be updated only by "IETF Consensus" as specified in [RFC5226] (an RFC that documents registry use and is approved by the IESG).

3.2.3. Experimental Numbers

No experimental bit positions are currently assigned in the Modes Registry, as indicated in the initial contents below.

3.2.4. OWAMP-Modes Initial Contents

OWAMP-Control connection establishment follows the procedure defined in section 3.1 of [RFC4656].

In the OWAMP-Modes registry, assignments are straightforward on the basis of bit positions, and there are no references to values - this is a difference from the comparable TWAMP registry (and a topic for improvement in the TWAMP-Modes registry).

An Extension of the OWAMP-Modes is proposed in [I-D.ietf-ippm-ipsec]. With this extension, the complete set of OWAMP Mode bit positions are as follows (including one reserved bit position):

OWAMP-Modes Registry

Bit Posit.	Description	Reference/Explanation
0	Unauthenticated	RFC4656, Section 3.1
1	Authenticated	RFC4656, Section 3.1
2	Encrypted	RFC4656, Section 3.1
3	Reserved	bit position (3)
4	IKEv2-derived Shared Secret Key	RFC_TBD and this memo new bit position (4)
5-31	Unassigned	

In the original OWAMP and TWAMP Modes field, setting bit position 0, 1 or 2 indicated the security mode of the Control protocol, and the Test protocol inherited the same mode (see section 4 of [RFC4656]).

The value of the Modes Field sent by the Server in the Server-Greeting message is the bit-wise OR of the modes (bit positions) that it is willing to support during this session. Thus, the last four bits of the Modes 32-bit Field are used. When no other features are activated, the first 28 bits MUST be zero. A client conforming to this extension of [RFC5357] MAY ignore the values in the first 28 bits of the Modes Field, or it MAY support other features that are communicated in these bit positions.

OWAMP and TWAMP registries for Modes may grow to contain different features and functions due to the inherent differences in one-way and two-way measurement configurations and the metrics they measure. No

attempt will be made to coordinate them unnecessarily, except the Reserved bit position 3 above. This is available for assignment if a mixed security mode [RFC5618] is defined for OWAMP, and would allow alignment with the comparable TWAMP feature.

4. Security Considerations

As this memo simply requests creation of a registry, it presents no new security or privacy issues for the Internet.

The security considerations that apply to any active measurement of live networks are relevant here as well. See [RFC4656] and [RFC5357].

Privacy considerations for measurement systems, particularly when Internet users participate in the tests in some way, are described in [I-D.ietf-lmap-framework].

5. Acknowledgements

The author would like to thank Kostas Pentikousis, Nalini Elkins, and Mike Ackermann for insightful reviews and comments.

6. References

6.1. Normative References

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

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

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

6.2. Informative References

[I-D.ietf-ippm-ipsec] Pentikousis, K., Zhang, E., and Y. Cui, "IKEv2-derived Shared Secret Key for O/TWAMP", draft-ietf-ippm-ipsec-10 (work in progress), May 2015.

[I-D.ietf-lmap-framework]
Eardley, P., Morton, A., Bagnulo, M., Burbridge, T., Aitken, P., and A. Akhter, "A framework for Large-Scale Measurement of Broadband Performance (LMAP)", draft-ietf-lmap-framework-14 (work in progress), April 2015.

[RFC5618] Morton, A. and K. Hedayat, "Mixed Security Mode for the Two-Way Active Measurement Protocol (TWAMP)", RFC 5618, August 2009.

Author's Address

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/

Network Working Group
Internet-Draft
Intended status: Informational
Expires: December 31, 2015

Srivathsa Sarangapani
Peyush Gupta
Juniper Networks
June 29, 2015

Monitoring Service KPIs using TWAMP
draft-sp-ippm-monitor-services-kpi-00

Abstract

We are introducing a new method to calculate services KPIs and metrics in the network using TWAMP protocol. The services here ranging from subscriber aware services to security application, Traffic load balancing, content delivery, real time streaming and like.The KPIs discussed in this draft include Service Latency, Serviced Packets Count, Serviced Subscriber Count, Application Liveliness and Session load per Service. KPIs monitoring of these services therefore, play a vital role in optimum usage of network resources such as capacity and throughput. Once we have the attributes like service latency, the network topology can be chosen to provide better quality of experience to the end user. For different services, the attributes may vary and our design takes care of supporting different KPIs for different services model. Additionally, liveliness of application and servers can be monitored using this proposed solution.

Table of Contents

1. Introduction 3
1.1. Conventions used in this document 3
1.1.1. Requirements Language 3
1.2. Terminology 3
2. Purpose and Scope 4
3. Logical Model 5
4. TWAMP Extensions 6
4.1. TWAMP-Control Extensions 7
4.1.1. Connection Setup with Services KPIs Monitoring . . . 7
4.1.2. Services KPI-Monitor-REQ Command 7
4.1.3. Services KPI-Monitor-RSP Command 8
4.1.4. Services KPI-Monitor-IND Command 9
4.1.5. Services KPI-Monitor-ACK Command 10
4.1.6. Request-TW-Session Command Format 11
4.2. TWAMP-Test Extension 12
5. Services KPIs 17
5.1. Services Keepalive Monitoring 17
5.2. Service Latency 17
5.3. Serviced Packets Count 18
5.4. Serviced Bytes Count 19
5.5. Serviced Subscriber Count 19
6. Acknowledgements 21
7. IANA Considerations 21
8. Security Considerations 23
9. Normative References 23
Authors' Addresses 23

1. Introduction

The TWAMP-Test runs between a Session-Sender and a Session-Reflector RFC 5357 [RFC5357]. The existing TWAMP-Test packet format has existing padding octets that are currently not used (either set to zero or pseudo-random values). These octets can be used to carry additional information between the Session-Sender and the Session-Reflector. The proposed extension uses these padding octets and provide a method to monitor services KPIs in the network. This feature is termed as Services KPI Monitoring using TWAMP.

The services here refers to Layer 4 to Layer 7 services like DPI, SFW, CGNAT, TDF and so on. Additionally these services can refer to applications like DNS, HTTP, FTP and so on. The KPIs MAY include service latency, service load monitoring in terms of number of flows, number of sessions, number of subscribers, number of octets, liveliness of a service.

For instance, Services KPI Monitoring using TWAMP MAY be used to measure service latency of DPI, number of CGNAT flows, number of TDF subscribers and so on. Similarly this MAY be used to monitor the liveliness of the DNS Server, HTTP Server and so on.

As per the proposed extension, both the TWAMP-Control and the TWAMP-Test packet formats are modified. One TWAMP-Test session SHALL be used to monitor KPIs for a specific service. But there can be multiple KPIs monitored using a single test session for a specific service. A single TWAMP-Control connection MAY establish multiple TWAMP-Test sessions that measure KPIs for multiple services in the network.

This extension can be used to monitor KPIs for standalone service or a set of services.

1.1. Conventions used in this document

1.1.1. Requirements Language

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 RFC 2119 [RFC2119].

1.2. Terminology

TWAMP: Two-Way Active Measurement Protocol

KPI: Key Performance Indicator

DPI: Deep Packet Inspection

CGNAT: Carrier Grade Network Address Translation

SFW: Stateful Firewall

TDF: Traffic Detection Function

DNS: Domain Name Server

HTTP: Hyper Text Transfer Protocol

FTP: File Transfer Protocol

SKMC: Services KPI Monitoring Command

PDU: Protocol Data Unit

2. Purpose and Scope

The purpose of this extension is to provide a method to describe an additional optional feature for TWAMP RFC 5357 [RFC5357].

The scope of the extension is limited to specifications of the following features:

1. Extension of the modes of operation through assignment of a new value in the Modes field to communicate feature capability.

2. Addition of new command types to identify, negotiate and monitor the supported services and supported KPIs for each service between Control-Client and TWAMP Server.

3. Use of existing padding octets of TWAMP-Test session to carry the services KPI data that is being monitored as part of the TWAMP-Test session.

The purpose for this feature is to enhance TWAMP protocol to monitor and calculate service-related KPIs in real-time that helps in network performance analysis and optimization.

The actual method to calculate the KPIs is not discussed and depends on implementation.

3. Logical Model

The set of messages that are exchanged between the Control-Client and TWAMP Server to negotiate and monitor the services KPI is referred to as Service Block (Fig 1.) Service Block MAY be a part of the same network element or can be a different entity.

Services KPI-Monitor-REQ is sent from Control-Client to TWAMP Server to get the list of supported services and the KPIs that can be monitored for each service. Once TWAMP Server receives this request, Services KPI-Monitor-RSP is sent with the number of services that can be monitored on this Control-Client connection.

This message is followed by Services KPI-Monitor-IND message from the Session-Reflector which contain a service ID to identify the service and the list of KPIs that are supported for this service.The client replies with the Services KPI-Monitor-ACK message that include the list of KPIs the client is interested in monitoring.These two sets of messages will be repeated for each of the services that Server can monitor.

Then the client will initiate Request-TW-Session Message that contain the service ID for a specific service. Once Server replies with the Accept-Session Message, the client SHALL start sending test packets that MAY contain Service PDU.

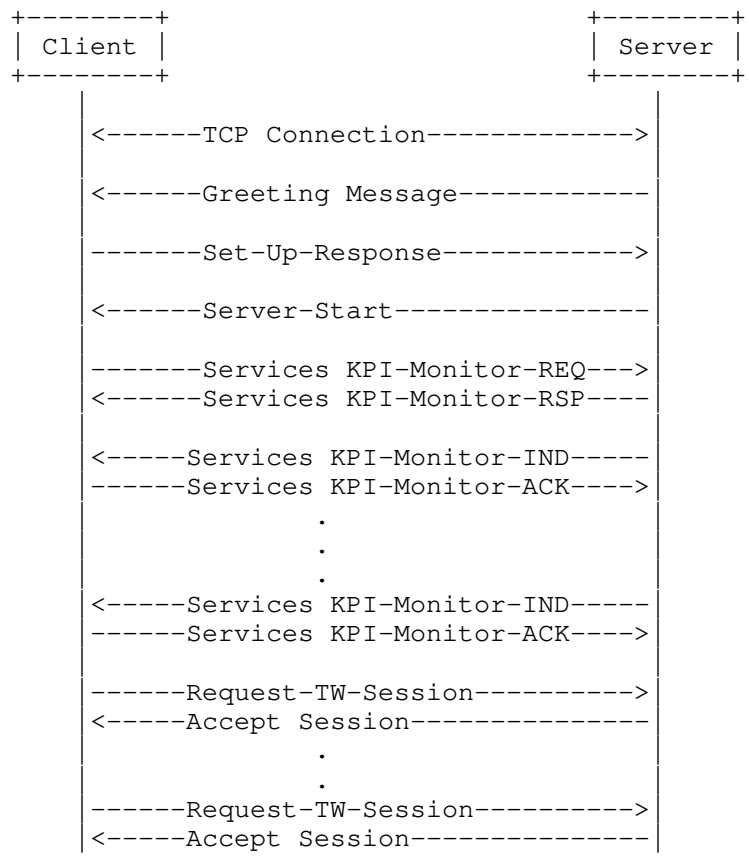


Figure 1

4. TWAMP Extensions

The TWAMP connection establishment follows the procedure defined in Section 3.1 of OWAMP [RFC4656] and Section 3.1 of TWAMP [RFC5357] where the Modes field is used to identify and select specific communication capabilities. At the same time the Modes field been recognized and used as an extension mechanism of TWAMP Reflect Octets and Symmetrical Size Features [RFC6038]. The new feature requires a new bit position to identify the ability of a Session-Reflector to monitor Services KPIs. There are changes in both the Control-Client and TWAMP-Test packet formats to support this functionality.

4.1. TWAMP-Control Extensions

The TWAMP-Control is a derivative of the OWAMP-Control, and provides two-way measurement capability. TWAMP; [RFC5357] uses the Modes field to identify and select specific communication capabilities, and this field is a recognized extension mechanism. The following Sections describe one such extension.

4.1.1. Connection Setup with Services KPIs Monitoring

TWAMP-Control connection establishment follows the procedure defined in Section 3.1 of OWAMP; [RFC4656]. The Services KPIs Monitoring using TWAMP mode requires one new bit position (and value) to identify the ability of the Server or the Session-Reflector to monitor the Services KPIs of the sessions. This new extension requires an additional TWAMP mode bit assignment as explained in Section 5.1.

A Control-Client MAY request for Services KPIs monitoring for some of its sessions. To do so, it needs to know which services can be monitored and the corresponding KPIs (of those services)that can be monitored.

Services KPI Monitoring Command (SKMC) consist of a set of messages which SHALL be used for monitoring the KPIs of a service. This new command requires an additional TWAMP Command Number as explained in Section 7.

4.1.2. Services KPI-Monitor-REQ Command

A Control-Client MAY send Services KPI-Monitor-REQ command to the Server to obtain the list of services and their KPIs that can be monitored by the Session-Reflector.

The format of the Services KPI-Monitor-REQ Command is as follows:

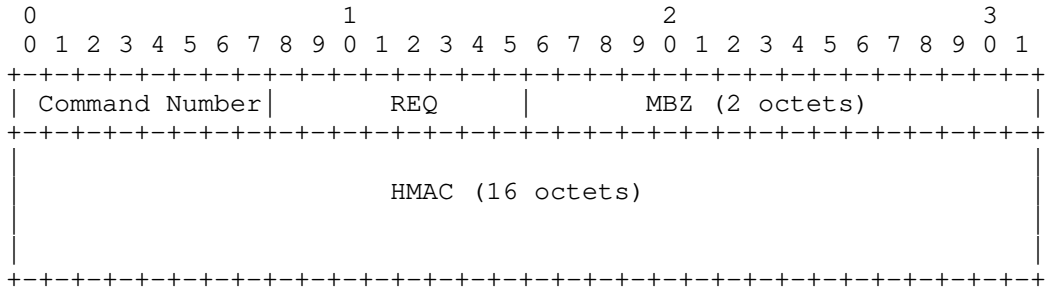


Figure 2: Services KPI-Monitor-REQ Command

Since this is a new command, a Command Number value should be allocated by the IANA in the registry as mentioned in Section 7. The command number indicates that this is one of the Services KPI Monitoring Command. The Control-Client MUST compose this command, and the Server MUST interpret this command, according to the field descriptions below.

The sub-type field MUST be REQ for this message. This message indicates that the Client is requesting Server to send the list of Services and the corrosponding KPIs that can be monitored.

The message is terminated with a single block HMAC, as illustrated above.

The Server MUST respond with Services KPI-Monitor-RSP Command Section 4.1.3.

4.1.3. Services KPI-Monitor-RSP Command

The Server responds to the Services KPI-Monitor-REQ Command by sending a Services KPI-Monitor-RSP Command. The format of the Services KPI-Monitor-RSP Command is as follows:

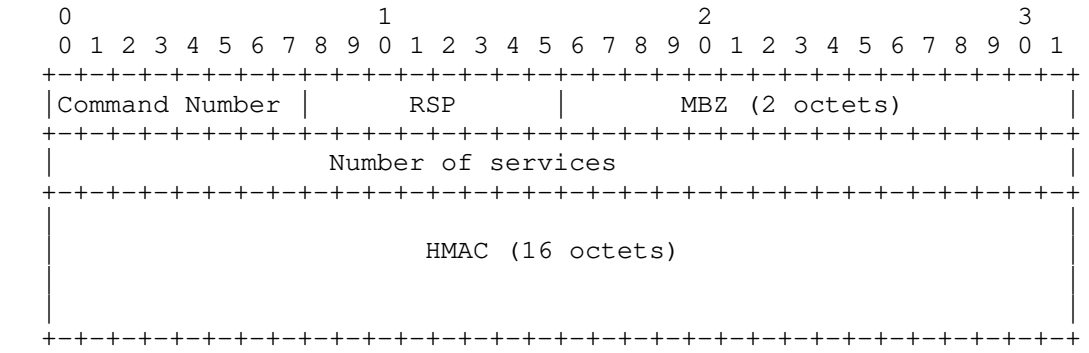


Figure 3: Services KPI-Monitor-RSP Command

The Command Number value here is same as mentioned in Section 7. The Server MUST compose this command, and the Control-Client MUST interpret this command, according to the field descriptions below.

The sub-type field MUST be RSP for this command. The field "Number of Services" indicates the number of Services for which the Session-Reflector can monitor the KPI.

The message is terminated with a single block HMAC, as illustrated above.

4.1.4. Services KPI-Monitor-IND Command

The Server MUST send the Services KPI-Monitor-IND Command after sending Services KPI Monitor-RSP message. This message includes the list of KPIs that can be monitored for a service.

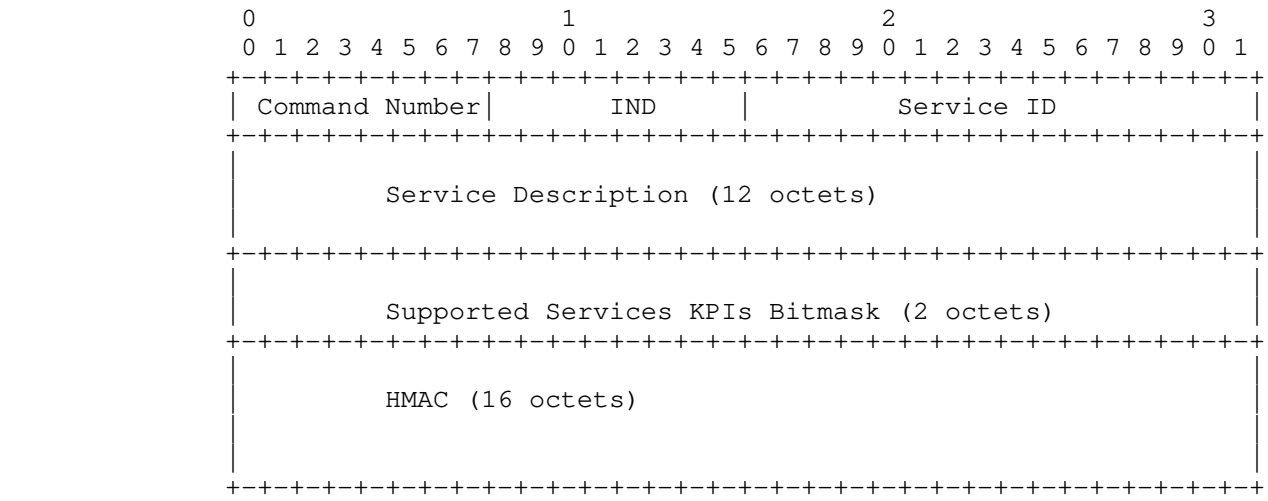


Figure 4: Services KPI-Monitor-IND Command

The Command Number value here is same as mentioned in Section 7. The Server MUST compose this command, and the Control-Client MUST interpret this command, according to the field descriptions below.

The sub-type field MUST be IND for this Command. This field indicates that the Server is responding to the Control-Client with the details of the KPIs of a service that can be monitored by Session-Reflector.

Service ID is an identifier which would be set by the Server to identify a Service. This ID MUST be used in the TWAMP-Control and TWAMP-Test messages to identify a particular Service. The range of Service ID MUST be 1 to 65535; The value 0 is Reserved.

Service Description MAY be set of alphanumeric characters that would provide a brief description of a particular Service. Example: "DPI" "CGNAT" "DNS-Server" "HTTP-Server".

Supported Services KPIs Bitmask is a bitmask that would indicate the kind of KPI Monitoring using TWAMP is supported by the Session-Reflector for a particular Service.

The message is terminated with a single block HMAC, as illustrated above.

For each Services KPIs monitoring supported, the Server MUST send one Services KPI-Monitor-IND Command to the Control-Client.

4.1.5. Services KPI-Monitor-ACK Command

The Control-client MUST respond back with a Services KPI-Monitor-ACK Command for each Services KPI-Monitor-IND Command.

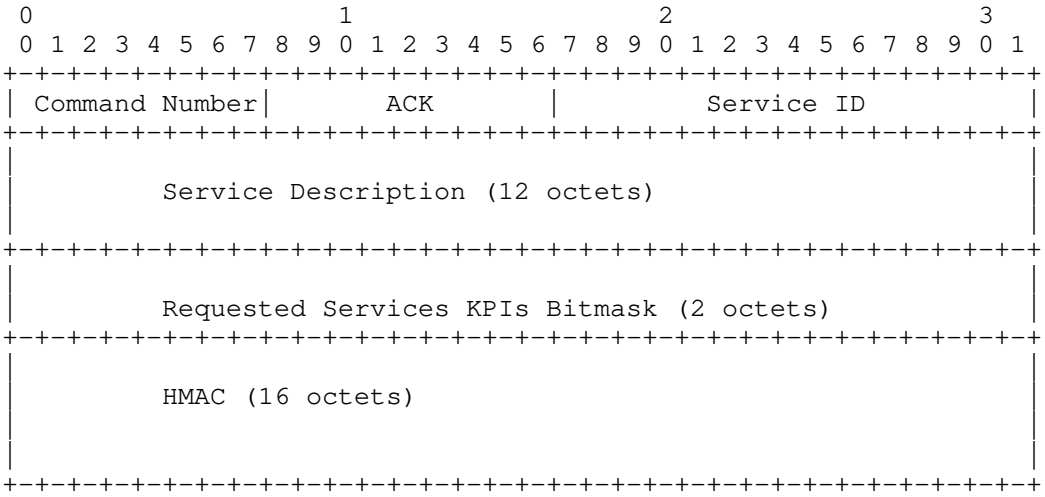


Figure 5: Services KPI-Monitor-ACK Command

The Command Number is same as mentioned in Section 7. The Server MUST frame this command, and the Control-Client MUST interpret this command, according to the field descriptions below.

The sub-type field MUST be ACK for this message. This field indicates that the Control-client is acknowledging the Server with details of which KPIs of a particular service it is interested in.

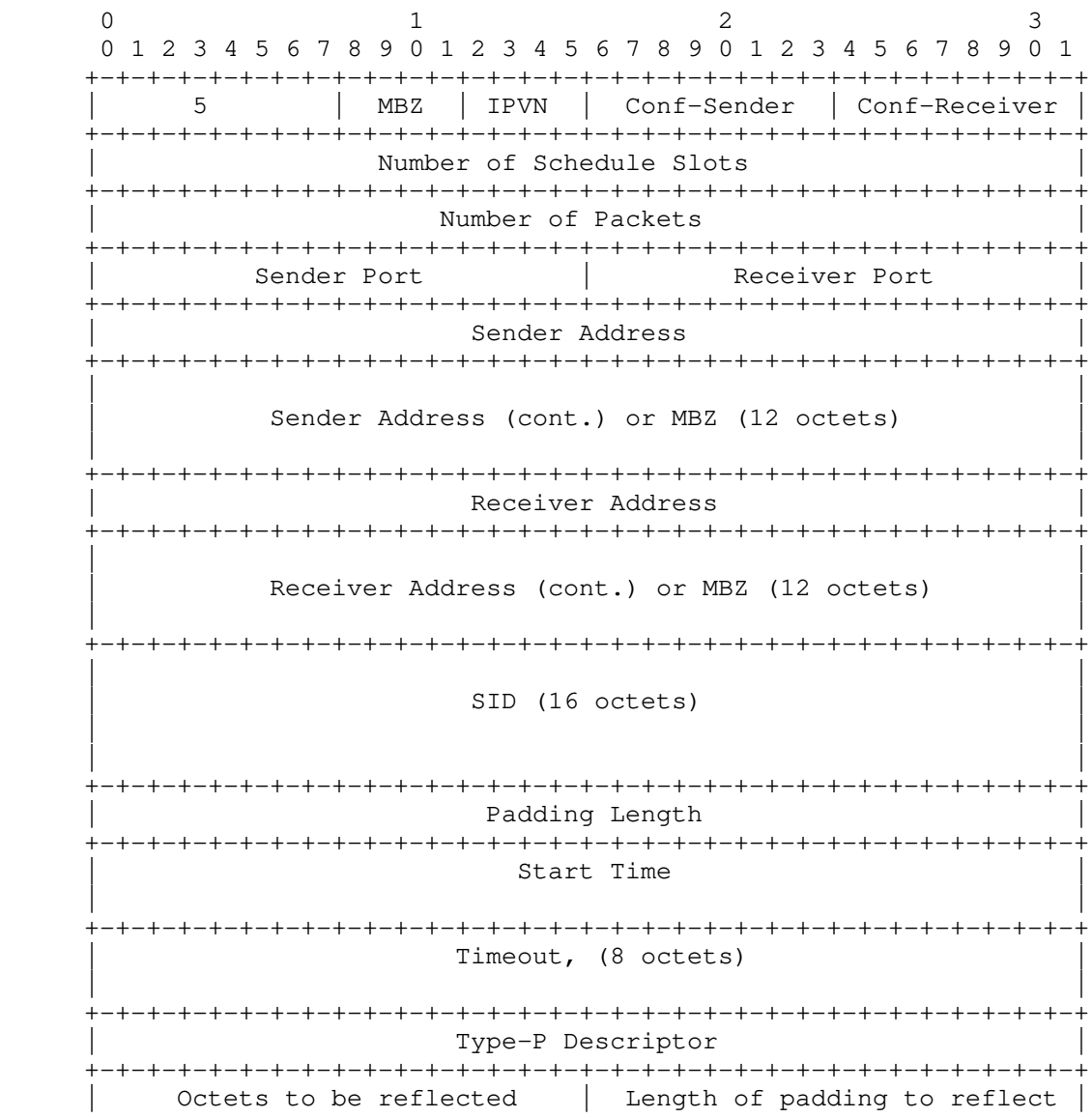
Service ID and Service Description MUST be same as that received in the Services KPI-Monitor-IND Command. These two fields together identify a particular Service.

Requested Services KPIs Bitmask MUST be set by the Control-Client and that indicates the KPIs of the services that the Control-Client is interested in monitoring. The KPIs can be a subset or the full set of KPIs sent in the corresponding Service KPI-Monitor-IND Command.

The Command is terminated with a single block HMAC, as illustrated above.

For each Services KPI-Monitor-IND Command received at the control-client, it acknowledges by sending a Services KPI-Monitor-ACK Command.

4.1.6. Request-TW-Session Command Format



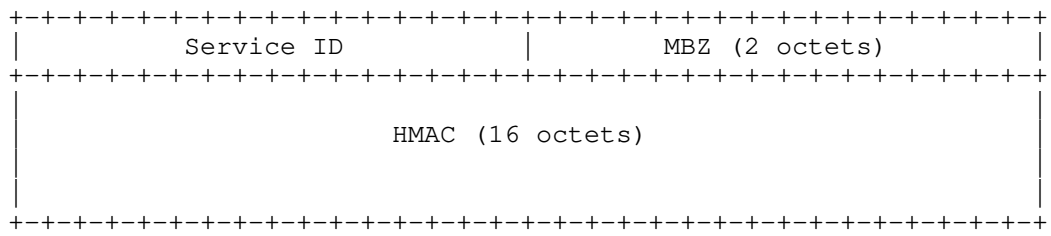


Figure 6: Request-TW-Session Command

This new feature requires 2 octets to indicate the Service ID as a part of Request-TW-Session Command. See Section 7 for details on the octet position. If Services KPIs Monitoring using TWAMP feature is not requested as a part of this TWAMP-Test Session, then the Service ID MUST be 0.

If Service ID has a non-zero value, then the Padding Length field MAY not have any significance. The test packets between Session-Sender and Session-Reflector MAY be of different size based on the implementation.

The actual test packets MAY contain valid data which SHOULD be interpreted by Session-Sender or Session-Reflector to monitor Services KPIs. Please refer TWAMP-Test Extension Section 4.2 for more details.

4.2. TWAMP-Test Extension

As part of this extension, the existing Packet Padding octets in the Test Packet MAY be used for the monitoring of the Services KPIs as explained in Section 5. The Packet Padding octets which were either zero or filled with pseudo-random values MAY now have some valid data like timestamps, statistics, service PDUs and so on.

The Session-Sender Test Session data packet formats are provided below.

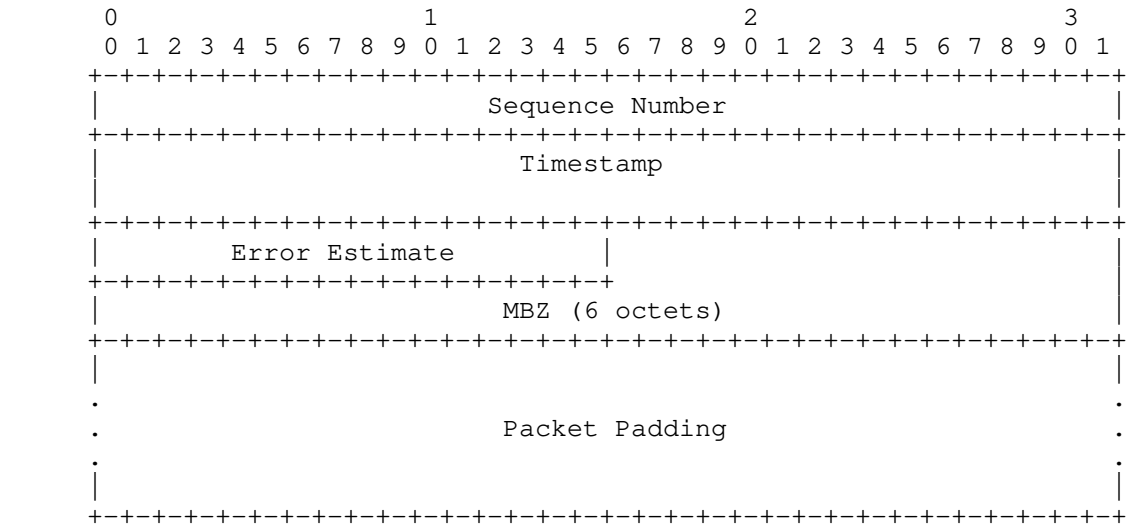


Figure 7: Unauthenticated Mode Session-Sender Data Packet Format

As a part of the extension, 6 octets of MBZ are added after the Error Estimate field.

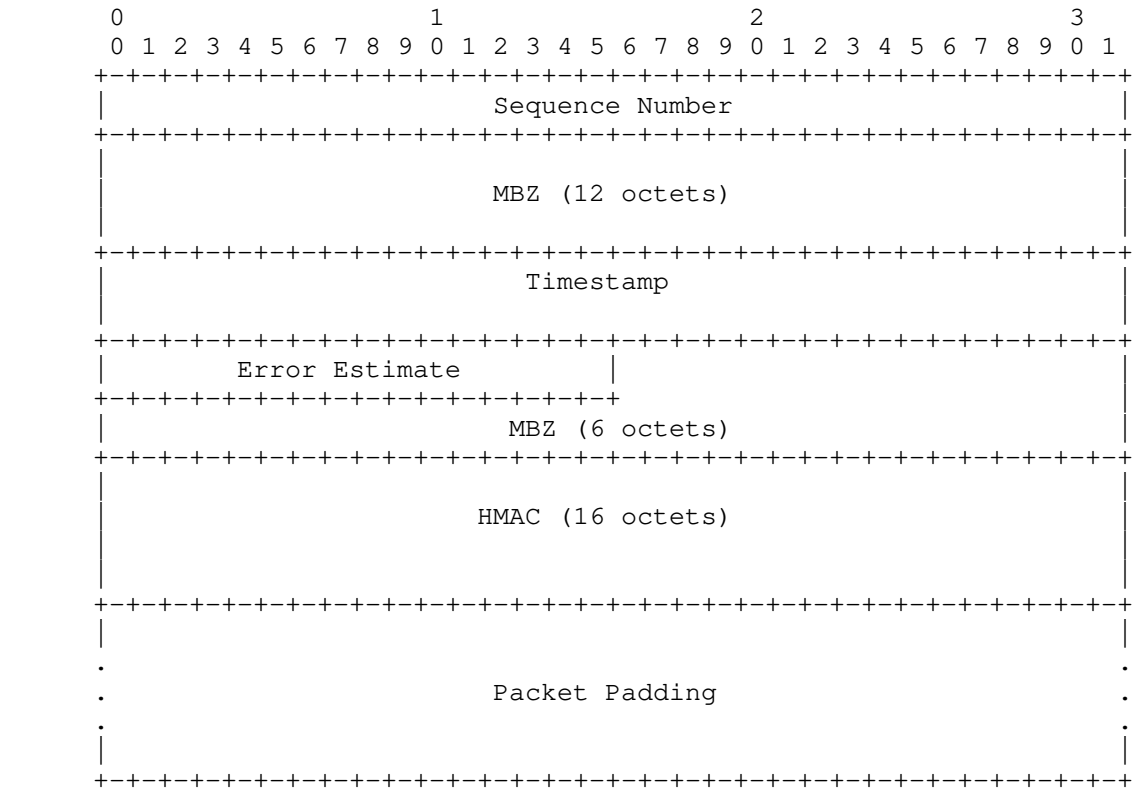


Figure 8: Authenticated and Encrypted Mode Session-Reflector Data Packet Format

The Session-Reflector Test Session data packet formats are provided below.

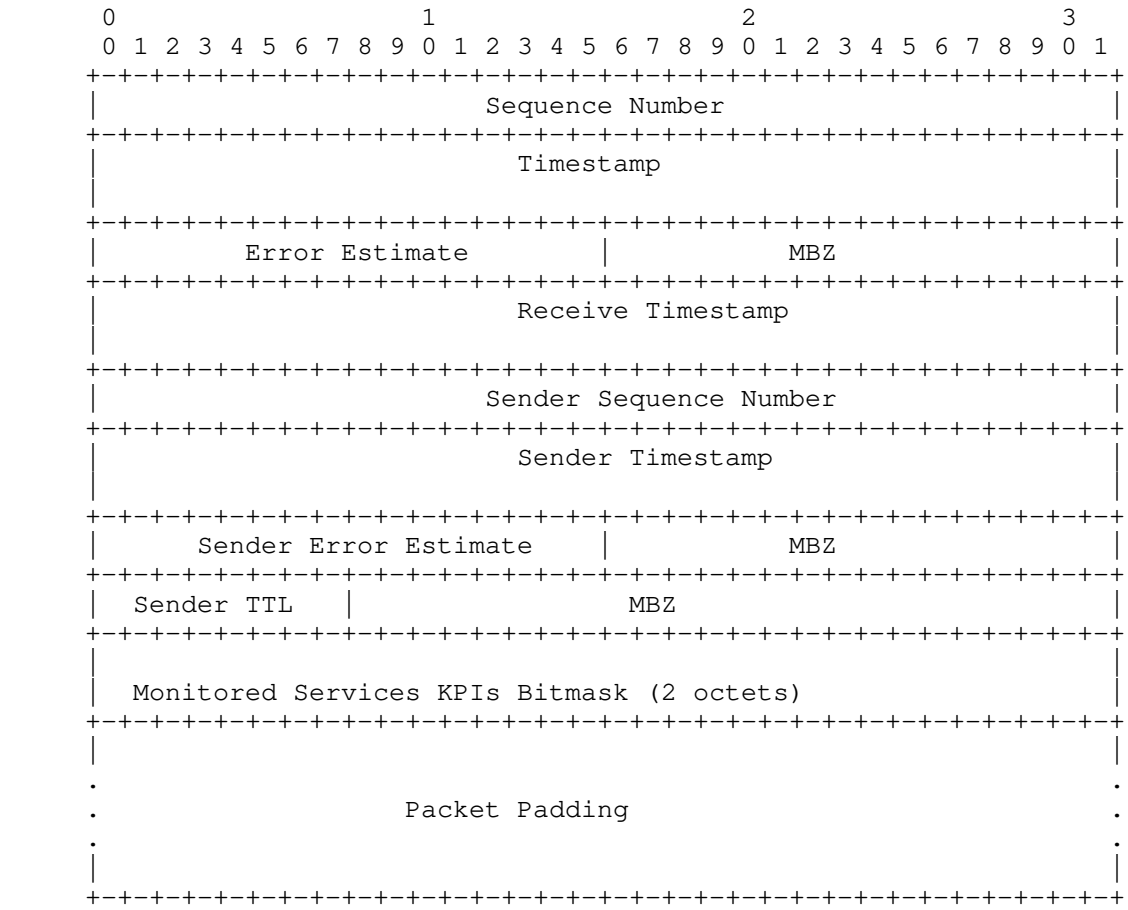
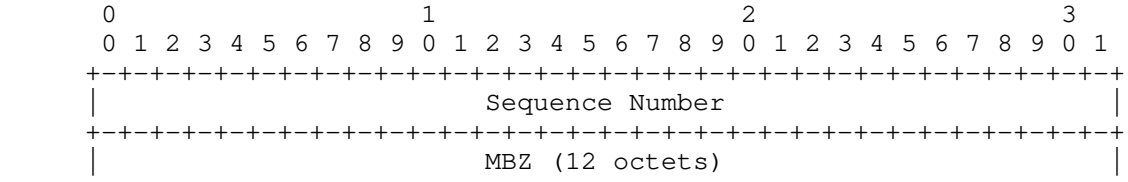


Figure 9: Unauthenticated Mode Session-Reflector Data Packet Format

As a part of this extension, The 3 octets of MBZ are added after the Error Estimate field to align the next set of fields.

Monitored Services KPIs Bitmask indicates the services KPIs that are present in this message. The KPIs would be present in the Packet Padding area in the same order as indicated by Bitmask starting from bit 0 Position.



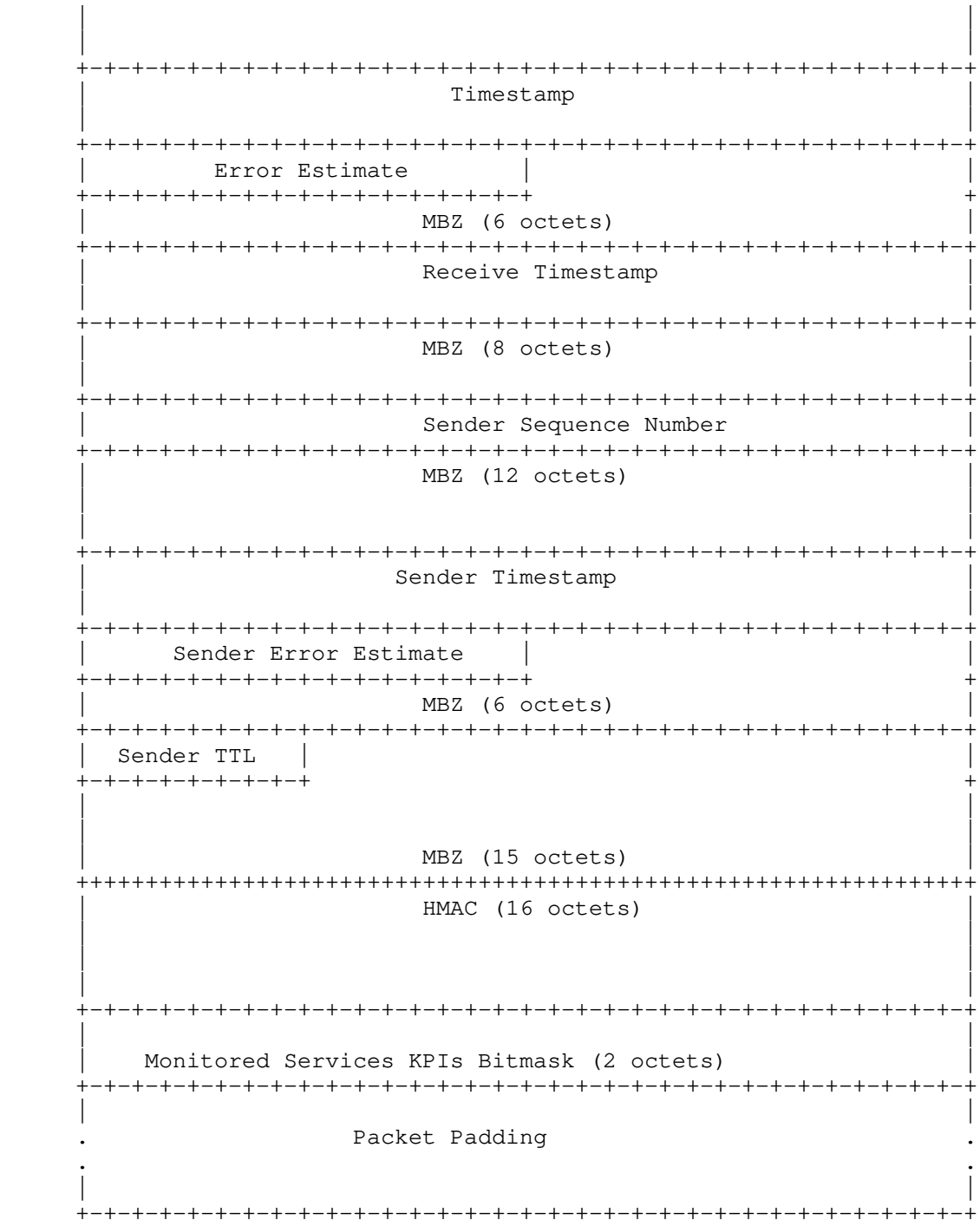

Timestamp
Error Estimate
MBZ (6 octets)
Receive Timestamp
MBZ (8 octets)
Sender Sequence Number
MBZ (12 octets)
Sender Timestamp
Sender Error Estimate
MBZ (6 octets)
Sender TTL
MBZ (15 octets)
HMAC (16 octets)
Monitored Services KPIs Bitmask (2 octets)
Packet Padding

Figure 10: Authenticated and Encrypted Mode Session-Reflector Data Packet Format

As a part of this extension, Monitored Services KPIs Bitmask indicates the services KPIs that are present in this message. The KPIs would be present in the Packet Padding area in the same order as indicated by Bitmask starting from bit 0 Position. The set of KPIs defined for a service are listed in Section 5.

5. Services KPIs

5.1. Services Keepalive Monitoring

The Session-Sender MAY send the Service PDU as part of the TWAMP-Test Packet Padding. When Session-Reflector receives the TWAMP-Test packet, it SHALL extract the Service PDU. Then Session-Reflector SHALL inject the Service PDU to the Service Block for service processing.

Based on whether the Session-Reflector received the response, the Session-Reflector SHALL decide whether the Service is alive or not.

The Session-Reflector MUST start the Packet Padding with the below 4 octets as indicated in Fig.11 [TBD]. This is followed by the Service PDU (which MAY be same as whatever was sent by Session-Sender or can be the reply/response packet of the Service Block).

Setting Bit 0(X) indicates that the Session-Reflector successfully sent the Service Request to the Service Block and received the response from the Service Block. If this bit is NOT set then it indicates that the Service Block is not functional.

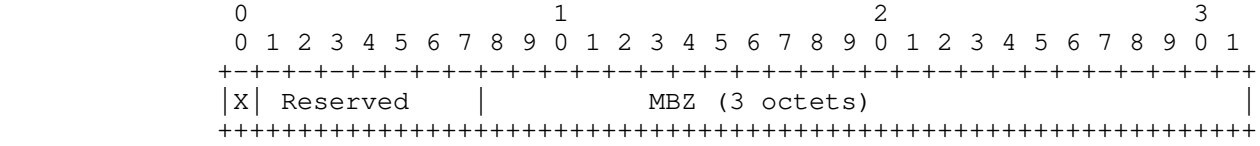


Figure 11: Services Keepalive Monitoring

5.2. Service Latency

The Session-Sender MAY send the Service PDU as part of the TWAMP-Test Packet Padding. When Session-Reflector receives the TWAMP-Test message, it SHALL extract the Service PDU and inject that service PDU in the Service Block.

The Session-Reflector MUST record the time, when this service PDU is injected. This SHALL be the Service latency measurement Sender Timestamp.

Once the Session-Reflector received the Response from the Service Block, it MUST record the time. This time SHALL be the Service latency measurement Receiver Timestamp.

If the Session-Reflector does NOT receive the Service PDU (within pre configured time which is implementation specific), then it shall indicate the Service latency measurement Receiver Timestamp as 0. The Session-Reflector MUST start the Packet Padding with the 16 octets indicated below. This SHALL be followed by the Service PDU (which MAY be same as whatever was sent by Session-Sender or can be the reply/response packet of the Service Block).

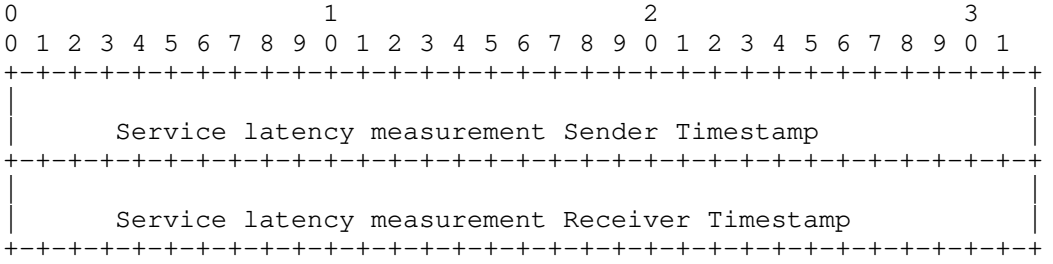


Figure 12: Services Keepalive Monitoring

5.3. Serviced Packets Count

When Session-Reflector receives the TWAMP-Test message, it SHALL get the information about Number of Ingress Service Data packets and Number of Egress Service Data packets from the Service Block. How Session-Reflector gets this information is implemenatation dependant. Once the Session-Reflector gets this information, it MUST start the Packet Padding with the below 16 octets. This SHALL be followed by the actual Packet Padding.

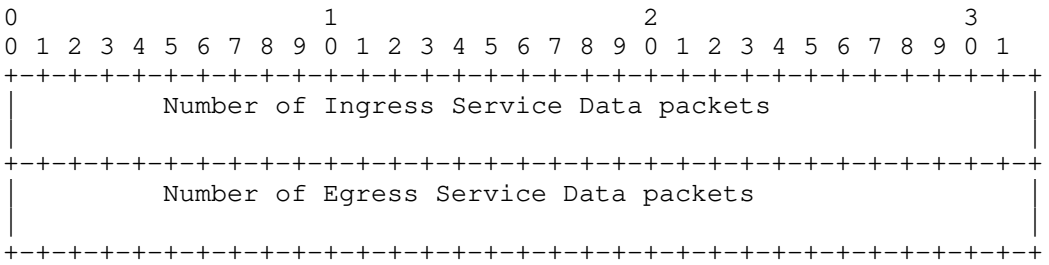


Figure 13: Serviced Packets Count

5.4. Serviced Bytes Count

When Session-Reflector receives the TWAMP-Test message, it SHALL get the information about Number of Ingress Service Data bytes and Number of Egress Service Data bytes from the Service Block. How Session-Reflector gets this information is implemenatation dependant. Once the Session-Reflector gets this information, it MUST start the Packet Padding with the below 16 octets. This SHALL be followed by the actual Packet Padding.

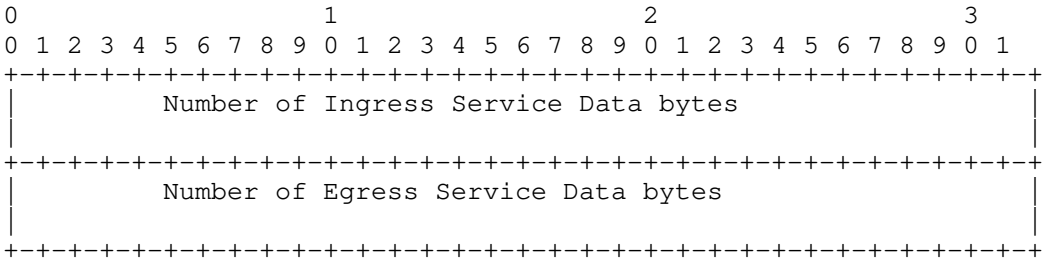


Figure 14: Serviced Bytes Count

5.5. Serviced Subscriber Count

When Session-Reflector receives the TWAMP-Test message, it SHALL get the below information :

Total Number of Subscribers : Total number of subscribers that are currently present for the Service + 1. This count includes Active, Non-Active and any other type of subscribers.

Number of Active Subscribers : Number of subscribers who are currently Actively using the Service + 1. The meaning of Active MAY vary from Service to Service and this is implementation specific.

Number of Non-Active Subscribers : Number of subscribers who are currently NOT Actively using the Service + 1. The meaning of NOT Active MAY vary from Service to Service and this is implementation specific.

Number of Subscribers Added : Number of subscribers who were NEWLY added compared to the last time the statistics was taken + 1.

Number of Subscribers Deleted :Number of subscribers who were deleted, preempted compared to the last time the statistics was taken + 1.

Any of the above field can be 0 if that statistics is not supported or not valid for a particular service. Session-Reflector should always fill the value by increasing the actual service statistics by 1. Say for example, if Numer of Active Subscribers is 0, then Session-Reflector would fill this field as 1. When Session-Sender receives this value, it should subtract 1 from the received value and use it.

How Session-Reflector gets this information is implemenatation dependant. Once the Session-Reflector gets this information, it MUST start the Packet Padding with the below 40 octets. This SHALL be followed by the actual Packet Padding.

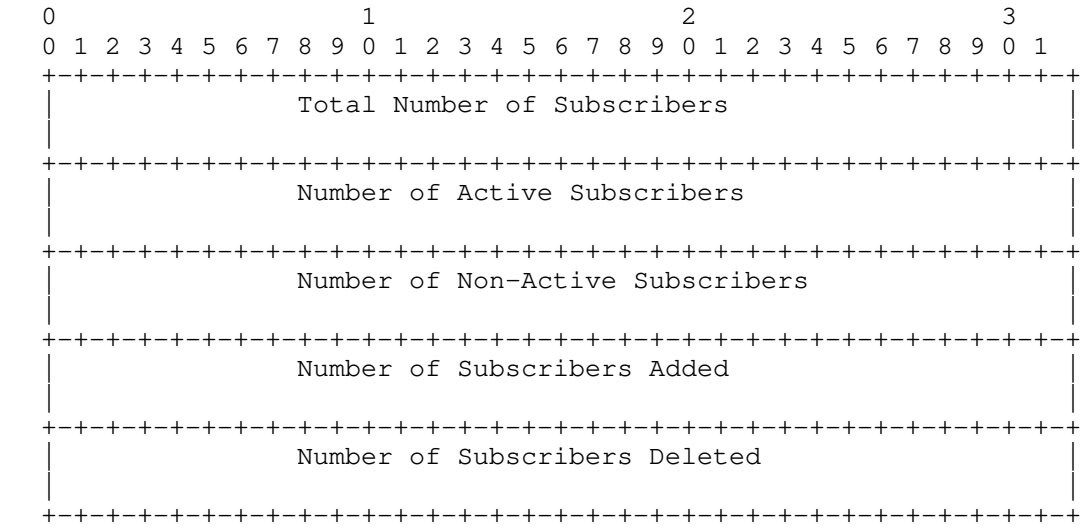


Figure 15: Serviced Subscriber Count

6. Acknowledgements

We would like to thank Perceival A Monteiro for their comments, suggestions, reviews, helpful discussion, and proof-reading

7. IANA Considerations

The TWAMP-Modes registry defined in [RFC6038]. IANA is requested to reserve a new bit in Modes registry for Services KPIs Monitoring Capability as follows:

Value	Description	Semantics	Reference
X (proposed 256)	Services KPIs Monitoring Capability	bit position Y(proposed 8)	This document

Table 1: Services KPIs Monitoring Capability

TWAMP-Control Command Number Registry defined in [RFC5938].IANA is requested to reserve a Command Number for Services KPIs Monitoring Capability as follows:

Value	Description	Semantics	Reference
SKMC (proposed 11)	Services KPIs Monitoring Command		This document

Table 2: New Service Latency Monitoring Command

TWAMP Services KPIs sub-type Registry

IANA is requested to reserve and maintain the sub-types as a part of Services KPIs Monitoring Command as follows:

Value	Description	Explanation
0	Reserved	
1	REQ	Section 4.1.2
2	RESP	Section 4.1.3
3	IND	Section 4.1.4
4	ACK	Section 4.1.5

Table 3: TWAMP Services KPIs sub-type Registry

TWAMP Services KPIs Registry

IANA is requested to reserve and maintain the below Services KPIs:

Value	Description	Explanation
0	None	
1	Keepalive	Whether the respective service is running or not
2	Service Latency	Service Latency which SHALL include the transit time and actual service time
4	Serviced Packets Count	Number of ingress and egress packets for the respective service
8	Serviced Bytes Count	Number of ingress and egress bytes for the respective service.
16	Serviced Subscriber Count	Number of subscribers currently active for the respective service.

Table 4: TWAMP Services KPIs Registry

Request-TW-Session message defined in [RFC6038].IANA is requested to reserve 2 octets for Service ID as follows:

Value	Description	Semantics	Reference
X	Service ID	2 Octets starting from offset 92th Octet	This document

Table 5: New Services KPIs Monitoring Capability

8. Security Considerations

NA

9. Normative References

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

[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.

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

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

Authors' Addresses

Srivathsa Sarangapani
Juniper Networks
Bangalore 560079
INDIA

Phone: +91 9845052354
Email: srivathsas@juniper.net

Peyush Gupta
Juniper Networks
T-313, Keerti Royal Apartment, Outer Ring Road
Bangalore, Karnataka 560043
INDIA

Phone: +91 9449251927
Email: peyushg@juniper.net

Network Working Group
Internet-Draft
Intended status: Experimental
Expires: September 6, 2015

A. Capello
M. Cociglio
G. Fioccola
L. Castaldelli
Telecom Italia
A. Tempia Bonda

March 5, 2015

A packet based method for passive performance monitoring
draft-tempia-ippm-p3m-00.txt

Abstract

This document describes a passive method to perform packet loss, delay and jitter measurements on live traffic. Implementation and deployment details are also explained in order to clarify how the tools and features currently available on existing routing platforms can be used to implement the method. This method, also called PNPM (Packet Network Performance Monitoring), has been invented and engineered in Telecom Italia and it's currently being used in Telecom Italia's network.

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 6, 2015.

Copyright Notice

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction 2
2. Overview of the method 4
3. Detailed description of the method 5
3.1. Packet loss measurement 5
3.2. One-way delay measurement 9
3.2.1. Single marking methodology 9
3.2.2. Average delay 10
3.2.3. Double marking methodology 11
3.3. Delay variation measurement 12
4. Implementation and deployment 12
4.1. Coloring the packets 14
4.2. Counting the packets 15
4.3. Collecting data and calculating packet loss 16
5. Hybrid measurement 16
6. Compliance with RFC6390 guidelines 16
7. Security Considerations 18
8. Conclusions 19
9. IANA Considerations 19
10. Acknowledgements 19
11. References 20
11.1. Normative References 20
11.2. Informative References 20
Authors' Addresses 20

1. Introduction

Nowadays, most of the traffic in Service Providers' networks carries multimedia content. Video contents are highly sensitive to packet loss [RFC2680], while interactive contents are sensitive to delay [RFC2679], and jitter [RFC3393].

In front of this scenario, Service Providers need methodologies and tools to monitor and measure network performances with an adequate accuracy, in order to constantly control the quality of experience perceived by their customers. On the other hand, performance monitoring provides useful information for improving network management (e.g. isolation of network problems, troubleshooting, etc.).

A lot of work related to OAM, that includes also performance monitoring techniques, has been done by Standards Developing Organizations: [RFC7276] provides a good overview of existing OAM

mechanisms defined in IETF, ITU-T and IEEE. Considering IETF, a lot of work has been done on fault detection and connectivity verification, while a minor effort has been dedicated so far to performance monitoring. The IPPM WG has defined standard metrics to measure network performance; however, the methods developed in the WG mainly refer to active measurement techniques. More recently, the MPLS WG has defined mechanisms for measuring packet loss, one-way and two-way delay, and delay variation in MPLS networks[RFC6374], but their applicability to passive measurements has some limitations, especially for pure connection-less networks.

The lack of adequate tools to measure packet loss with the desired accuracy drove an effort in Telecom Italia to design a new method for the performance monitoring of live traffic, possibly easy to implement and deploy. The effort led to the method described in this document: basically, it is a passive performance monitoring technique, potentially applicable to any kind of packet based traffic, including Ethernet, IP, and MPLS, both unicast and multicast. The method addresses primarily packet loss measurement, but it can be easily extended to one-way delay and delay variation measurements as well. It doesn't require any protocol extension or interaction with existing protocols, thus avoiding any interoperability issue. Even if the method doesn't raise any specific need for standardization, it could be further improved by means of some extension to existing protocols, but this aspect is left for further study and it is out of the scope of this document.

The method has been explicitly designed for passive measurements but it can also be used with active probes. Passive measurements are usually more easily understood by customers and provide a much better accuracy, especially for packet loss measurements.

The method described in this document has been invented and engineered in Telecom Italia and it's currently being used in Telecom Italia's network.

This document is organized as follows:

- Section 2 gives an overview of the method, including a comparison with alternate measurement strategies;
- Section 3 describes the method in detail
- Section 4 discusses implementation and deployment considerations, with special regard to the choices adopted in Telecom Italia's own implementation;
- Section 5 includes some considerations about security aspects;

- Section 6 finally summarizes some concluding remarks.

2. Overview of the method

In order to perform packet loss measurements on a live traffic flow, different approaches exist. The most intuitive one consists in numbering the packets, so that each router that receives the flow can immediately detect a packet missing. This approach, though very simple in theory, is not simple to achieve: it requires the insertion of a sequence number into each packet and the devices must be able to extract the number and check it in real time. Such a task can be difficult to implement on live traffic: if UDP is used as the transport protocol, the sequence number is not available; on the other hand, if a higher layer sequence number (e.g. in the RTP header) is used, extracting that information from each packet and process it in real time could overload the device.

An alternate approach is to count the number of packets sent on one end, the number of packets received on the other end, and to compare the two values. This operation is much simpler to implement, but requires that the devices performing the measurement are in sync: in order to compare two counters it is required that they refer exactly to the same set of packets. Since a flow is continuous and cannot be stopped when a counter has to be read, it could be difficult to determine exactly when to read the counter. A possible solution to overcome this problem is to virtually split the flow in consecutive blocks by inserting periodically a delimiter so that each counter refers exactly to the same block of packets. The delimiter could be for example a special packet inserted artificially into the flow. However, delimiting the flow using specific packets has some limitations. First, it requires generating additional packets within the flow and requires the equipment to be able to process those packets. In addition, the method is vulnerable to out of order reception of delimiting packets and, to a lesser extent, to their loss.

The method proposed in this document follows the second approach, but it doesn’t use additional packets to virtually split the flow in blocks. Instead, it "colors" the packets so that the packets belonging to the same block will have the same color, whilst consecutive blocks will have different colors. Each change of color represents a sort of auto-synchronization signal that guarantees the consistency of measurements taken by different devices along the path.

Figure 1 represents a very simple network and shows how the method can be used to measure packet loss on different network segments: by enabling the measurement on several interfaces along the path, it is

possible to perform link monitoring, node monitoring or end-to-end monitoring. The method is flexible enough to measure packet loss on any segment of the network and can be used to isolate the faulty element.

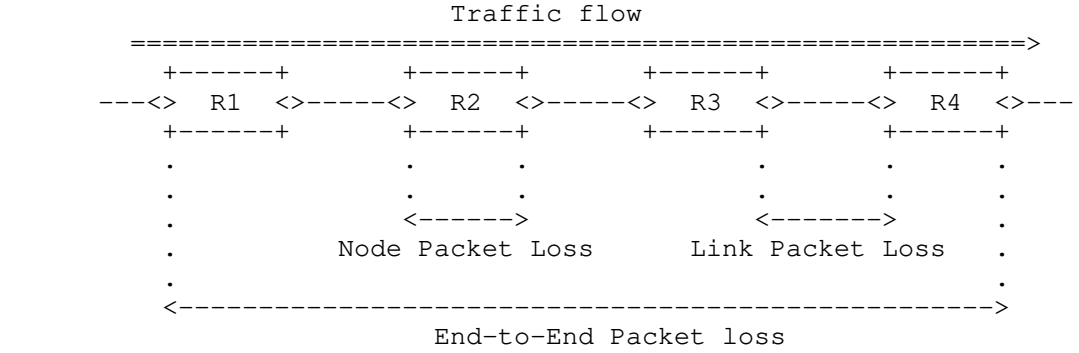


Figure 1: Available measurements

3. Detailed description of the method

This section describes in detail how the method. A special emphasis is given to the measurement of packet loss, that represents the core application of the method, but applicability to delay and jitter measurements is also considered.

3.1. Packet loss measurement

The basic idea is to virtually split traffic flows into consecutive blocks: each block represents a measurable entity unambiguously recognizable by all network devices along the path. By counting the number of packets in each block and comparing the values measured by different network devices along the path, it is possible to measure packet loss occurred in any single block between any two points.

As discussed in the previous section, a simple way to create the blocks is to "color" the traffic (two colors are sufficient) so that packets belonging to different consecutive blocks will have different colors. Whenever the color changes, the previous block terminates and the new one begins. Hence, all the packets belonging to the same block will have the same color and packets of different consecutive blocks will have different colors. The number of packets in each block depends on the criterion used to create the blocks: if the color is switched after a fixed number of packets, then each block will contain the same number of packets (except for any losses); but if the color is switched according to a fixed timer, then the number

of packets may be different in each block depending on the packet rate.

The following figure shows how a flow looks like when it is split in traffic blocks with colored packets.

A: packet with A coloring
B: packet with B coloring

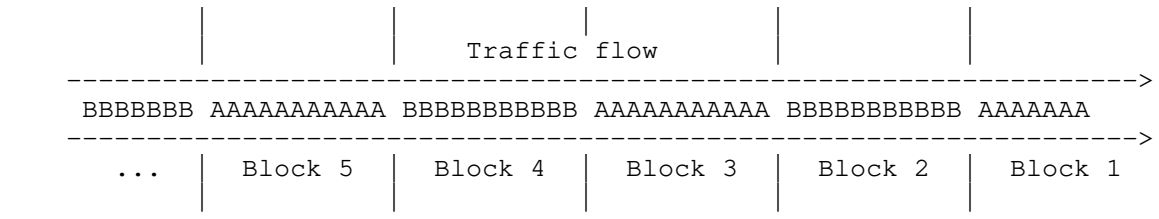


Figure 2: Traffic coloring

Figure 3 shows how the method can be used to measure link packet loss between two adjacent nodes.

Referring to the figure, let's assume we want to monitor the packet loss on the link between two routers: router R1 and router R2. According to the method, the traffic is colored alternatively with two different colors, A and B. Whenever the color changes, the transition generates a sort of square-wave signal, as depicted in the following figure.

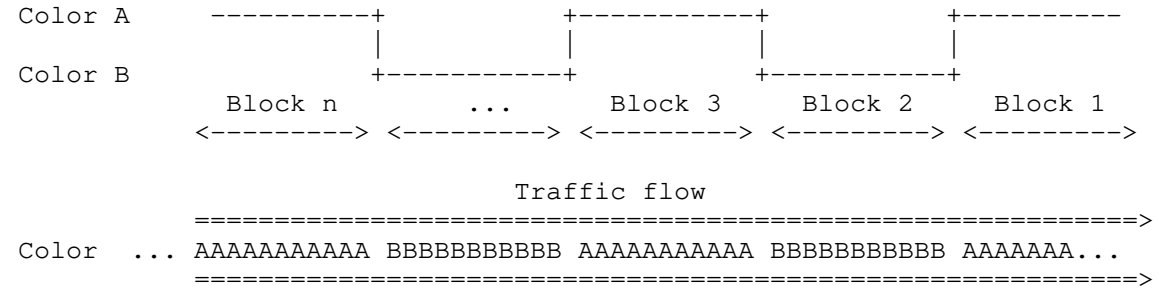


Figure 3: Application of the method to compute link packet loss

Traffic coloring could be done by R1 itself or by an upward router. R1 needs two counters, C(A)R1 and C(B)R1, on its egress interface: C(A)R1 counts the packets with color A and C(B)R1 counts those with color B. As long as traffic is colored A, only counter C(A)R1 will

be incremented, while C(B)R1 is not incremented; vice versa, when the traffic is colored as B, only C(B)R1 is incremented. C(A)R1 and C(B)R1 can be used as reference values to determine the packet loss from R1 to any other measurement point down the path. Router R2, similarly, will need two counters on its ingress interface, C(A)R2 and C(B)R2, to count the packets received on that interface and colored with color A and B respectively. When an A block ends, it is possible to compare C(A)R1 and C(A)R2 and calculate the packet loss within the block; similarly, when the successive B block terminates, it is possible to compare C(B)R1 with C(B)R2, and so on for every successive block.

Likewise, by using two counters on R2 egress interface it is possible to count the packets sent out of R2 interface and use them as reference values to calculate the packet loss from R2 to any measurement point down R2.

Using a fixed timer for color switching offers a better control over the method: the (time) length of the blocks can be chosen large enough to simplify the collection and the comparison of measures taken by different network devices. It’s preferable to read the value of the counters not immediately after the color switch: some packets could arrive out of order and increment the counter associated to the previous block (color), so it is worth waiting for some seconds. The drawback is that the longer the duration of the block, the less frequent the measurement can be taken.

The following table shows how the counters can be used to calculate the packet loss between R1 and R2. The first column lists the sequence of traffic blocks while the other columns contain the counters of A-colored packets and B-colored packets for R1 and R2. In this example, we assume that the values of the counters are reset to zero whenever a block ends and its associated counter has been read: with this assumption, the table shows only relative values, that is the exact number of packets of each color within each block. If the values of the counters were not reset, the table would contain cumulative values, but the relative values could be determined simply by difference from the value of the previous block of the same color.

The color is switched on the basis of a fixed timer (not shown in the table), so the number of packets in each block is different.

Block	C(A)R1	C(B)R1	C(A)R2	C(B)R2	Loss
1	375	0	375	0	0
2	0	388	0	388	0
3	382	0	381	0	1
4	0	377	0	374	3
...	...	...	...	...	...
n	0	387	0	387	0
n+1	379	0	377	0	2

Table 1: Evaluation of counters for packet loss measurements

During an A block (blocks 1, 3 and n+1), all the packets are A-colored, therefore the C(A) counters are incremented to the number seen on the interface, while C(B) counters are zero. Vice versa, during a B block (blocks 2, 4 and n), all the packets are B-colored: C(A) counters are zero, while C(B) counters are incremented.

When a block ends (because of color switching) the relative counters stop incrementing and it is possible to read them, compare the values measured on router R1 and R2 and calculate the packet loss within that block.

For example, looking at the table above, during the first block (A-colored), C(A)R1 and C(A)R2 have the same value (375), which corresponds to the exact number of packets of the first block (no loss). Also during the second block (B-colored) R1 and R2 counters have the same value (388), which corresponds to the number of packets of the second block (no loss). During blocks three and four, R1 and R2 counters are different, meaning that some packets have been lost: in the example, one single packet (382-381) was lost during block three and three packets (377-374) were lost during block four.

The method applied to R1 and R2 can be extended to any other router and applied to more complex networks, as far as the measurement is enabled on the path followed by the traffic flow(s) being observed.

3.2. One-way delay measurement

The same principle used to measure packet loss can be applied also to one-way delay measurement. There are three alternatives, as described hereinafter.

3.2.1. Single marking methodology

The alternation of colors can be used as a time reference to calculate the delay. Whenever the color changes (that means that a new block has started) a network device can store the timestamp of the first packet of the new block; that timestamp can be compared with the timestamp of the same packet on a second router to compute packet delay. Considering Figure 4, R1 stores a timestamp TS(A1)R1 when it sends the first packet of block 1 (A-colored), a timestamp TS(B2)R1 when it sends the first packet of block 2 (B-colored) and so on for every other block. R2 performs the same operation on the receiving side, recording TS(A1)R2, TS(B2)R2 and so on. Since the timestamps refer to specific packets (the first packet of each block) we are sure that timestamps compared to compute delay refer to the same packets. By comparing TS(A1)R1 with TS(A1)R2 (and similarly TS(B2)R1 with TS(B2)R2 and so on) it is possible to measure the delay between R1 and R2. In order to have more measurements, it is possible to take and store more timestamps, referring to other packets within each block.

In order to coherently compare timestamps collected on different routers, the network nodes must be in sync. Furthermore, a measurement is valid only if no packet loss occurs and if packet misordering can be avoided, otherwise the first packet of a block on R1 could be different from the first packet of the same block on R2 (f.i. if that packet is lost between R1 and R2 or it arrives after the next one).

The following table shows how timestamps can be used to calculate the delay between R1 and R2. The first column lists the sequence of blocks while other columns contain the timestamp referring to the first packet of each block on R1 and R2. The delay is computed as a difference between timestamps. For the sake of simplicity, all the values are expressed in milliseconds.

Block	TS(A)R1	TS(B)R1	TS(A)R2	TS(B)R2	Delay R1-R2
1	12.483	-	15.591	-	3.108
2	-	6.263	-	9.288	3.025
3	27.556	-	30.512	-	2.956
	-	18.113	-	21.269	3.156
...	...	...	...	...	...
n	77.463	-	80.501	-	3.038
n+1	-	24.333	-	27.433	3.100

Table 2: Evaluation of timestamps for delay measurements

The first row shows timestamps taken on R1 and R2 respectively and referring to the first packet of block 1 (which is A-colored). Delay can be computed as a difference between the timestamp on R2 and the timestamp on R1. Similarly, the second row shows timestamps (in milliseconds) taken on R1 and R2 and referring to the first packet of block 2 (which is B-colored). Comparing timestamps taken on different nodes in the network and referring to the same packets (identified using the alternation of colors) it is possible to measure delay on different network segments.

For the sake of simplicity, in the above example a single measurement is provided within a block, taking into account only the first packet of each block. The number of measurements can be easily increased by considering multiple packets in the block: for instance, a timestamp could be taken every N packets, thus generating multiple delay measurements. Taking this to the limit, in principle the delay could be measured for each packet, by taking and comparing the corresponding timestamps (possible but impractical from an implementation point of view).

3.2.2. Average delay

As mentioned before, the method previously exposed for measuring the delay is sensitive to out of order reception of packets. In order to overcome this problem, a different approach has been considered: it is based on the concept of average delay. The average delay is calculated by considering the average arrival time of the packets within a single block. The network device locally stores a timestamp

for each packet received within a single block: summing all the timestamps and dividing by the total number of packets received, the average arrival time for that block of packets can be calculated. By subtracting the average arrival times of two adjacent devices it is possible to calculate the average delay between those nodes. This method is robust to out of order packets and also to packet loss (only a small error is introduced). Moreover, it greatly reduces the number of timestamps (only one per block for each network device) that have to be collected by the management system. On the other hand, it only gives one measure for the duration of the block (f.i. 5 minutes), and it doesn't give the minimum and maximum delay values. This limitation could be overcome by reducing the duration of the block (f.i. from 5 minutes to a few seconds) by means of an highly optimized implementation of the method.

By summing the average delays of the two directions of a path, it is also possible to measure the two-way delay (round-trip delay).

3.2.3. Double marking methodology

The Single marking methodology for one-way delay measurement is sensitive to out of order reception of packets. The first approach to overcome this problem is described before and is based on the concept of average delay. But the limitation of average delay is that it doesn't give information about the delay values distribution for the duration of the block. Additionally it may be useful to have not only the average delay but also the minimum and maximum delay values and, in wider terms, to know more about the statistic distribution of delay values. So in order to have more information about the delay and to overcome out of order issues, a different approach can be introduced: it is based on double marking methodology.

Basically, the idea is to use the first marking to create the alternate flow and, within this colored flow, a second marking to select the packets for measuring delay/jitter. The first marking is needed for packet loss and average delay measurement. The second marking creates a new set of marked packets that are fully identified over the network, so that a network device can store the timestamps of these packets; these timestamps can be compared with the timestamps of the same packets on a second router to compute packet delay values for each packet. The number of measurements can be easily increased by changing the frequency of the second marking. But the frequency of the second marking must be not too high in order to avoid out of order issues. Between packets with the second marking there should be a security time gap (e.g. this gap could be, at the minimum, the average network delay calculated with the previous methodology) to avoid out of order issues and also to have a

number of measurement packets that is rate independent. If a second marking packet is lost, the delay measurement for the considered block is corrupted and should be discarded.

3.3. Delay variation measurement

Similarly to one-way delay measurement (both for single marking and double marking), the method can also be used to measure the inter-arrival jitter. The alternation of colors can be used as a time reference to measure delay variations. Considering the example depicted in Figure 4, R1 stores a timestamp TS(A)R1 whenever it sends the first packet of a block and R2 stores a timestamp TS(B)R2 whenever it receives the first packet of a block. The inter-arrival jitter can be easily derived from one-way delay measurement, by evaluating the delay variation of consecutive samples.

The concept of average delay can also be applied to delay variation, by evaluating the variation of average interval beetween consecutive packets of the flow from the transmitting point R1 to the receiving point R2. (by evaluating the average difference in one-way delay between consecutive packets of the flow from the transmitting point R1 to the receiving point R2).

4. Implementation and deployment

The methodology described in the previous sections has been implemented in Telecom Italia by leveraging functions and tools available on IP routers and it's currently being used to monitor packet loss in some portions of Telecom Italia's network. The application of the method to delay measurement is currently being evaluated in Telecom Italia's labs.

The fundamental steps for the implementation of the method can be summarized in the following items:

- coloring the packets;
- counting the packets;
- collecting data and calculating the packet loss.

Before going deeper into the implementation details, it's worth mentioning two different strategies that can be used when implementing the method:

- flow-based: the flow-based strategy is used when only a limited number of traffic flows need to be monitored. This could be the case, for example, of IPTV channels or other specific applications

traffic with high QoS requirements (i.e. Mobile Backhauling traffic). According to this strategy, only a subset of the flows is colored. Counters for packet loss measurements can be instantiated for each single flow, or for the set as a whole, depending on the desired granularity. A relevant problem with this approach is the necessity to know in advance the path followed by flows that are subject to measurement. Path rerouting and traffic load-balancing increase the issue complexity, especially for unicast traffic. The problem is easier to solve for multicast traffic where load balancing is seldom used, especially for IPTV traffic where static joins are frequently used to force traffic forwarding and replication. Another application is on Mobile Backhauling, implemented with a VPN MPLS in Telecom Italia's network; in this case the problem with unicast traffic is overcome by monitoring just the two Provider Edge nodes of the VPN MPLS.

- link-based: measurements are performed on all the traffic on a link by link basis. The link could be a physical link or a logical link (for instance an Ethernet VLAN or a MPLS PW). Counters could be instantiated for the traffic as a whole or for each traffic class (in case it is desired to monitor each class separately), but in the second case a couple of counters is needed for each class.

The current implementation in Telecom Italia uses the first strategy. As mentioned, the flow-based measurement requires the identification of the flow to be monitored and the discovery of the path followed by the selected flow. It is possible to monitor a single flow or multiple flows grouped together, but in this case measurement is consistent only if all the flows in the group follow the same path. Moreover, a Service Provider should be aware that, if a measurement is performed by grouping many flows, it is not possible to determine exactly which flow was affected by packets loss. In order to have measures per single flow it is necessary to configure counters for each specific flow. Once the flow(s) to be monitored have been identified, it is necessary to configure the monitoring on the proper nodes. Configuring the monitoring means configuring the policy to intercept the traffic and configuring the counters to count the packets. To have just an end-to-end monitoring, it is sufficient to enable the monitoring on the first and the last hop routers of the path: the mechanism is completely transparent to intermediate nodes and independent from the path followed by traffic flows. On the contrary, to monitor the flow on a hop-by-hop basis along its whole path it is necessary to enable the monitoring on every node from the source to the destination. In case the exact path followed by the flow is not known a priori (i.e. the flow has multiple paths to reach the destination) it is necessary to enable the monitoring system on

every path: counters on interfaces traversed by the flow will report packet count, counters on other interfaces will be null.

4.1. Coloring the packets

The coloring operation is fundamental in order to create packet blocks. This implies choosing where to activate the coloring and how to color the packets.

In case of flow-based measurements, it is desirable, in general, to have a single coloring node because it is easier to manage and doesn't rise any risk of conflict (consider the case where two nodes color the same flow). Thus it is necessary to color the flow as close as possible to the source. In addition, coloring a flow close to the source allows an end-to-end measure if a measurement point is enabled on the last-hop router as well. The only requirement is that the coloring must change periodically and every node along the path must be able to identify unambiguously the colored packets. For link-based measurements, all traffic needs to be colored when transmitted on the link. If the traffic had already been colored, then it has to be re-colored because the color must be consistent on the link. This means that each hop along the path must (re-)color the traffic; the color is not required to be consistent along different links.

Traffic coloring can be implemented by setting a specific bit in the packet header and changing the value of that bit periodically. With current router implementations, only QoS related fields and features offer the required flexibility to set bits in the packet header. In case a Service Provider only uses the three most significant bits of the DSCP field (corresponding to IP Precedence) for QoS classification and queuing, it is possible to use the two less significant bits of the DSCP field (bit 0 and bit 1) to implement the method without affecting QoS policies. One of the two bits (bit 0) could be used to identify flows subject to traffic monitoring (set to 1 if the flow is under monitoring, otherwise it is set to 0), while the second (bit 1) can be used for coloring the traffic (switching between values 0 and 1, corresponding to color A and B) and creating the blocks.

In practice, coloring the traffic using the DSCP field can be implemented by configuring on the router output interface an access list that intercepts the flow(s) to be monitored and applies to them a policy that sets the DSCP field accordingly. Since traffic coloring has to be switched between the two values over time, the policy needs to be modified periodically: an automatic script ca be used perform this task on the basis of a fixed timer. In Telecom Italia's implementation this timer is set to 5 minutes: this value

showed to be a good compromise between measurement frequency and stability of the measurement (i.e. possibility to collect all the measures referring to the same block).

4.2. Counting the packets

Assuming that the coloring of the packets is performed only by the source node, the nodes between source and destination (included) have to count the colored packets that they receive and forward: this operation can be enabled on every router along the path or only on a subset, depending on which network segment is being monitored (a single link, a particular metro area, the backbone, the whole path).

Since the color switches periodically between two values, two counters (one for each value) are needed: one counter for packets with color A and one counter for packets with color B. For each flow (or group of flows) being monitored and for every interface where the monitoring is active, a couple od counters is needed. For example, in order to monitor separately 3 flows on a router with 4 interfaces involved, 24 counters are needed (2 counters for each of the 3 flows on each of the 4 interfaces). If traffic is colored using the DSCP field, as in Telecom Italia's implementation, an access-list that matches specific DSCP values can be used to count the packets of the flow(s) being monitored.

In case of link-based measurements the behaviour is similar except that coloring and counting operations are performed on a link by link basis at each endpoint of the link.

Another important aspect to take into consideration is when to read the counters: in order to count the exact number of packets of a block the routers must perform this operation when that block has ended: in other words, the counter for color A must be read when the current block has color B, in order to be sure that the value of the counter is stable. This task can be accomplished in two ways. The general approach suggests to read the counters periodically, many times during a block duration, and to compare these successive readings: when the counter stops incrementing means that the current block has ended and its value can be elaborated safely. Alternatively, if the coloring operation is performed on the basis of a fixed timer, it is possible to configure the reading of the counters according to that timer: for example, if each block is 5 minutes long, reading the counter for color A every 5 minute in the middle of the subsequent block (with color B) is a safe choice. A sufficient margin should be considered between the end of a block and the reading of the counter, in order to take into account any out-of-order packets. The choice of a 5 minutes timer for colore switching was also inspired by these considerations

4.3. Collecting data and calculating packet loss

The nodes enabled to perform performance monitoring collect the value of the counters, but they are not able to directly use this information to measure packet loss, because they only have their own samples. For this reason, an external Network Management System (NMS) is required to collect and elaborate data and to perform packet loss calculation. The NMS compares the values of counters from different nodes and can calculate if some packets were lost (even a single packet) and also where packets were lost.

The value of the counters needs to be transmitted to the NMS as soon as it has been read. This can be accomplished by using SNMP or FTP and can be done in Push Mode or Polling Mode. In the first case, each router periodically sends the information to the NMS, in the latter case it is the NMS that periodically polls routers to collect information. In any case, the NMS has to collect all the relevant values from all the routers within one cycle of the timer (5 minutes).

If link-based measurement is used, it would be possible to use a protocol to exchange values of counters between the two endpoints in order to let them perform the packet loss calculation for each traffic direction. A similar approach could be complicated if applied to a flow-based measurement.

5. Hybrid measurement

The method has been explicitly designed for passive measurements but it can also be used with active measurements. In order to have both end to end measurements and intermediate measurements (hybrid measurements) two end points can exchanges artificial traffic flows and apply alternate marking over these flows. In the intermediate points artificial traffic is managed in the same way as real traffic and measured as specified before.

6. Compliance with RFC6390 guidelines

RFC6390 [RFC6390] defines a framework and a process for developing Performance Metrics for protocols above and below the IP layer (such as IP-based applications that operate over reliable or datagram transport protocols).

This document doesn't aim to propose a new Performance Metric but a new method of measurement for a few Performance Metrics that have already been standardized. Nevertheless, it's worth applying [RFC6390] guidelines to the present document, in order to provide a more complete and coherent description of the proposed method. We

used a subset of the Performance Metric Definition template defined by [RFC6390].

- Metric name and description: as already stated, this document doesn't propose any new Performance Metric. On the contrary, it describes a novel method for measuring packet loss [RFC2680]. The same concept, with small differences, can also be used to measure delay [RFC2679], and jitter [RFC3393]. The document mainly describes the applicability to packet loss measurement.

- Method of Measurement or Calculation: according to the method described in the previous sections, the number of packets lost is calculated by subtracting the value of the counter on the source node from the value of the counter on the destination node. Both counters must refer to the same color. The calculation is performed when the value of the counters is in a steady state.

- Units of Measurement: the method calculates and reports the exact number of packets sent by the source node and not received by the destination node.

- Measurement Points: the measurement can be performed between adjacent nodes, on a per-link basis, or along a multi-hop path, provided that the traffic under measurement follows that path. In case of a multi-hop path, the measurements can be performed both end-to-end and hop-by-hop.

- Measurement Timing: the method have a constraint on the frequency of measurements. In order to perform a measure, the counter must be in a steady state: this happens when the traffic is being colored with the alternate color; in the current implementation the time interval is set to 5 minutes.

- Implementation: the current implementation of the method uses two encodings of the DSCP field to color the packets; this enables the use of policy configurations on the router to color the packets and accordingly configure the counter for each color. The path followed by traffic being measured should be known in advance in order to configure the counters along the path and be able to compare the correct values.

- Use and Applications: the method can be used to measure packet loss with high precision (i.e. 10exp(-7)) on live traffic; moreover, by combining end-to-end and per-link measurements, the method is useful to pinpoint the single link that is experiencing loss events.

- Reporting Model: the value of the counters has to be sent to a centralized management system that perform the calculations; such samples must contain a reference to the time interval they refer to, so that the management system can perform the correct correlation; the samples have to be sent while the corresponding counter is in a steady state (within a time interval), otherwise the value of the sample should be stored locally.

- Dependencies: the values of the counters have to be correlated to the time interval they refer to; moreover, as far the current implementation is based on DSCP values, there are significant dependencies on the usage of the DSCP field: it must be possible to rely on unused DSCP values without affecting QoS-related configuration and behavior; moreover, the intermediate nodes must not change the value of the DSCP field not to alter the measurement.

- Organization of Results: the method of measurement produces singletons.

- Parameters: currently, the main parameter of the method is the time interval used to alternate the colors and read the counters.

7. Security Considerations

This document specifies a method to perform measurements in the context of a Service Provider's network and has not been developed to conduct Internet measurements, so it does not directly affect Internet security nor applications which run on the Internet. However, implementation of this method must be mindful of security and privacy concerns.

There are two types of security concerns: potential harm caused by the measurements and potential harm to the measurements. For what concerns the first point, the measurements described in this document are passive, so there are no packets injected into the network causing potential harm to the network itself and to data traffic. Nevertheless, the method implies modifications on the fly to the IP header of data packets: this must be performed in a way that doesn't alter the quality of service experienced by packets subject to measurements and that preserve stability and performance of routers doing the measurements. The measurements themselves could be harmed by routers altering the coloring of the packets, or by an attacker injecting artificial traffic. Authentication techniques, such as digital signatures, may be used where appropriate to guard against injected traffic attacks.

The privacy concerns of network measurement are limited because the method only relies on information contained in the IP header without any release of user data.

8. Conclusions

The advantages of the method described in this document are:

- easy implementation: it can be implemented using features already available on major routing platforms;
- low computational effort: the additional load on processing is negligible;
- accurate packet loss measurement: single packet loss granularity is achieved with a passive measurement;
- potential applicability to any kind of packet/frame –based traffic: Ethernet, IP, MPLS, etc., both unicast and multicast;
- robustness: the method can tolerate out of order packets and it’s not based on "special" packets whose loss could have a negative impact;
- no interoperability issues: the features required to implement the method are available on all current routing platforms.

The method doesn’t raise any specific need for standardization, but it could be further improved by means of some extension to existing protocols. Specifically, the use of DiffServ bits for coloring the packets could not be a viable solution in some cases: a standard method to color the packets for this specific application could be beneficial.

9. IANA Considerations

There are no IANA actions required.

10. Acknowledgements

The authors would like to thank Domenico Laforgia, Daniele Accetta and Mario Bianchetti for their contribution to the definition and the implementation of the method.

11. References

11.1. Normative References

[RFC2679] Almes, G., Kalidindi, S., and M. Zekauskas, "A One-way Delay Metric for IPPM", RFC 2679, September 1999.

[RFC2680] Almes, G., Kalidindi, S., and M. Zekauskas, "A One-way Packet Loss Metric for IPPM", RFC 2680, September 1999.

[RFC3393] Demichelis, C. and P. Chimento, "IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)", RFC 3393, November 2002.

11.2. Informative References

[RFC6374] Frost, D. and S. Bryant, "Packet Loss and Delay Measurement for MPLS Networks", RFC 6374, September 2011.

[RFC6390] Clark, A. and B. Claise, "Guidelines for Considering New Performance Metric Development", BCP 170, RFC 6390, October 2011.

[RFC7276] Mizrahi, T., Sprecher, N., Bellagamba, E., and Y. Weingarten, "An Overview of Operations, Administration, and Maintenance (OAM) Tools", RFC 7276, June 2014.

Authors' Addresses

Alessandro Capello
Telecom Italia
Via Reiss Romoli, 274
Torino 10148
Italy

Email: alessandro.capello@telecomitalia.it

Mauro Cociglio
Telecom Italia
Via Reiss Romoli, 274
Torino 10148
Italy

Email: mauro.cociglio@telecomitalia.it

Giuseppe Fioccola
Telecom Italia
Via Reiss Romoli, 274
Torino 10148
Italy

Email: giuseppe.fioccola@telecomitalia.it

Luca Castaldelli
Telecom Italia
Via Reiss Romoli, 274
Torino 10148
Italy

Email: luca.castaldelli@telecomitalia.it

Alberto Tempia Bonda

Email: alberto.tempia@gmail.com