Test Protocol for One-way IP Capacity Measurement
draft-ietf-ippm-capacity-protocol-02

Abstract

This memo addresses the problem of protocol support for measuring Network Capacity metrics in RFC 9097, where the method deploys a feedback channel from the receiver to control the sender’s transmission rate in near-real-time. This memo defines a simple protocol to perform the RFC 9097 (and other) measurements.

See Section 10: The authors seek feedback to determine what additional features will be necessary for an IETF Standards Track Protocol, beyond what is present in the running code available now.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 10, 2023.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect
1. Introduction

The IETF’s efforts to define Network and Bulk Transport Capacity have been chartered and finally progressed after over twenty years.

Over that time, the performance community has seen development of Informative definitions in [RFC3148] for Framework for Bulk Transport Capacity (BTC), RFC 5136 for Network Capacity and Maximum IP-layer Capacity, and the Experimental metric definitions and methods in [RFC8337], Model-Based Metrics for BTC.

This memo looks at the problem of measuring Network Capacity metrics defined in [RFC9097] where the method deploys a feedback channel from
the receiver to control the sender’s transmission rate in near-real-time.

Although there are several test protocol already available for support and manage active measurements, this protocol is a major departure from their operation:

1. UDP transport is used for all setup, test activation, and control messages, and for results feedback (not TCP), simplifying operations.

2. TWAMP [RFC5357] and STAMP [RFC8762] use the philosophy that one host is a Session-Reflector, sending test packets every time they receive a test packet. This protocol supports a one-way test with periodic status messages returned to the sender. These messages are also a basis for on-path Round-trip delay measurements, which are a key input to the load adjustment search algorithm.

3. OWAMP [RFC4656] supports one-way testing with results Fetch at the end of the test session. This protocol supports a one-way test and requires periodic status messages returned to the sender to support the load adjustment search algorithm.

4. The security features of OWAMP [RFC4656] and TWAMP [RFC5357] have been described as "unusual", to the point that IESG approved their use while also asking that these methods not be used again. Further, the common OWAMP [RFC4656] and TWAMP [RFC5357] approach to security is over 15 years old at this time.

Note: the -00 update of this draft will be the last that describes version 8 of the protocol in the running code. Updates -01 and -02 of the draft correspond to version 9 of the protocol, which strives to allow interoperability with version 8.

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, Goals, and Applicability

The scope of this memo is to define a protocol to measure the Maximum IP-Layer Capacity metric and according to the standardized method.
The continued goal is to harmonize the specified metric and method across the industry, and this protocol supports the specifications of IETF and other Standards Development Organizations.

All active testing protocols currently defined by the IPPM WG are UDP-based, but this protocol specifies both control and test protocols using UDP transport. Also, the control protocol continues operating during testing to convey results and dynamic configurations.

The primary application of the protocol described here is the same as in Section 2 of [RFC7497] where:

- The access portion of the network is the focus of this problem statement. The user typically subscribes to a service with bidirectional access partly described by rates in bits per second.

3. Protocol Overview

This section gives an informative overview of the communication protocol between two test end-points (without expressing requirements; later sections provide details and requirements).

One end-point takes the role of server, listening for connection requests on a well-known destination port from the other end-point, the client.

The client requires configuration of a test direction parameter (upstream or downstream test, where the client performs the role of sender or receiver, respectively) as well as the hostname or IP address of the server in order to begin the setup and configuration exchanges with the server.

The protocol uses UDP transport and has four phases:

1. Setup Request and Response Exchange: The client requests to begin a test by communicating its protocol version, intended security mode, and jumbo datagram support. The server either confirms matching configuration or rejects the connection. The server also communicates the ephemeral port for further communication when accepting the client’s request.

2. Test Activation Request and Response: the client composes a request conveying parameters such as the testing direction, the duration of the test interval and test sub-intervals, and various thresholds. The server then chooses to accept, ignore or modify any of the test parameters, and communicates the set that will be used unless the client rejects the modifications. Note that the
client assumes that the Test Activation exchange has opened any co-located firewalls and network address/port translators for the test connection (in response to the Request packet on the ephemeral port) and the traffic that follows. If the Test Activation Request is rejected or fails, the client assumes that the firewall will close the address/port combination after the firewall’s configured idle traffic time-out.

3. Test Stream Transmission and Measurement Feedback Messages:
Testing proceeds with one end-point sending load PDUs and the other end-point receiving the load PDUs and sending frequent status messages to communicate status and transmission conditions there. The feedback messages are input to a load-control algorithm at the server, which controls future sending rates at either end-point as needed. The choice to locate the load-control algorithm at the server, regardless of transmission direction, means that the algorithm can be updated more easily at a host within the network, and at a fewer number of hosts than the number of clients.

4. Stopping the Test: When the specified test duration has been reached, the server initiates the phase to stop the test by setting the STOP1 indication in load PDUs or status feedback messages. The client acknowledges by setting the STOP2 in further load PDUs or messages, and a graceful connection termination at each end-point follows. (Since the load PDUs and feedback messages are used, this phase is kind of a sub-phase of 3.) If the Test traffic stops or the communication path fails, the client assumes that the firewall will close the address/port combination after the firewall’s configured idle traffic time-out.

4. General Parameters and Definitions
For Parameters related to the Maximum IP-Layer Capacity Metric and Method, please see Section 4 of [RFC9097].

5. Setup Request and Response Exchange
All messages defined in this section SHALL use UDP transport. The hosts SHALL calculate and include the UDP checksum, or check the UDP checksum as necessary.

5.1. Setup Request
The client SHALL begin the Control protocol connection by sending a Setup Request message to the server’s control port.
The client SHALL simultaneously start a test initiation timer so that if the control protocol fails to complete all exchanges in the allocated time, the client software SHALL exit (close the UDP socket and indicate an error message to the user).

(Note: in version 8, the watchdog time-out is configured, in udpst.h, as #define WARNING_NOTRAFFIC 1 // Receive traffic stopped warning threshold (sec) #define TIMEOUT_NOTRAFFIC (WARNING_NOTRAFFIC + 4) or 5 seconds)

The Setup Request message PDU SHALL be organized as follows:

```c
uint16_t controlId;  // Control ID = 0xACE1
uint16_t protocolVer; // Protocol version = 0x08
uint8_t cmdRequest;  // Command request = 1 (request)
uint8_t cmdResponse; // Command response = 0
* uint16_t maxBandwidth;// Required bandwidth (added in v9)
uint16_t testPort;   // Test port on server (=0 for Request)
* uint8_t modifierBitmap;// Modifier bitmap (replaced jumboStatus in v9)
uint8_t authMode;    // Authentication mode
uint32_t authUnixTime;// Authentication time stamp
unsigned char authDigest[AUTH_DIGEST_LENGTH] // SHA256_DIGEST_LENGTH = 32 oct
```

The UDP PDU format layout SHALL be as follows (big-endian AB):

```
  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          controlId            |          protocolVer          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  cmdRequest   | cmdResponse   |         maxBandwidth          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           testPort            |modifierBitmap |   authMode    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         authUnixTime                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
|                                                               |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          authDigest[AUTH_DIGEST_LENGTH](256 bits)             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
When the client generates the authDigest, the calculation SHALL cover the entire header (9 fields). The current Unix time SHALL be read and inserted immediately prior to the calculation (as immediately as possible, as are all preceding fields).

5.2. Setup Request/Response Processing at the Server

When the server receives the Setup Request it SHALL validate the request by checking the protocol version, the maxBandwidth requested for the test, the modifierBitmap for use of options such as Jumbo datagram status and traditional MTU (1500 bytes), and the authentication data if utilized. The value in the authUnixTime field is a 32-bit time stamp and a 5 minute tolerance window (+/- 2.5 minutes) is used to prevent the replay of a Setup Request. All other fields would remain valid if the authUnixTime field was omitted from the PDU. The authUnixTime is covered by the authDigest hash.

If the client has selected options for:

- Jumbo datagram support status (modifierBitmap),
- Traditional MTU (modifierBitmap),
- Authentication mode (optional)

that do not match the server configuration, the server MUST reject the Setup Request. Note that a server implementation of protocol version 9 allows backward compatibility with version 8 when in use by the client.

(Note: in version 8, the watchdog time is configured, in udpsst.h, as #define WARNING_NOTRAFFIC 1 // Receive traffic stopped warning threshold (sec) #define TIMEOUT_NOTRAFFIC (WARNING_NOTRAFFIC + 4) or 5 seconds)

If the Setup Request must be rejected (due to any of the reasons in the Command response codes listed below), a Setup Response SHALL be sent back to the client with a corresponding command response value indicating the reason for the rejection, unless the server requires Authentication, in which case the Setup Request SHOULD fail silently. The exception is for operations support: server administrators using Authentication are permitted to send a Setup Response to support operations and troubleshooting.
uint16_t controlId;  // Control ID = 0xACE1
uint16_t protocolVer;  // Protocol version = 0x08
uint8_t cmdRequest;   // Command request = 2 (reply)
uint8_t cmdResponse;  // Command response = <see table below>
uint16_t maxBandwidth; // Required bandwidth (added in v9)
uint16_t testPort;    // Test port on server (available port in Response)
uint8_t modifierBitmap; // Modifier bitmap (replaced jumboStatus, table below)
uint8_t authMode;     // Authentication mode
uint32_t authUnixTime; // Authentication time stamp
unsigned char authDigest[AUTH_DIGEST_LENGTH] // 32 octets, MBZ

cmdResponse Code Field: Command Server Response Codes (CSRP)
CHSR_CRSP_NONE 0 = None
CHSR_CRSP_ACKOK 1 = Acknowledgement
CHSR_CRSP_BADVER 2 = Bad Protocol Version
CHSR_CRSP_BADJS 3 = Invalid Jumbo datagram option
CHSR_CRSP_AUTHINC 4 = Unexpected Authentication in Setup Request
CHSR_CRSP_AUTHREQ 5 = Authentication missing in Setup Request
CHSR_CRSP_AUTHINV 6 = Invalid authentication method
CHSR_CRSP_AUTHFAIL 7 = Authentication failure
CHSR_CRSP_AUTHTIME 8 = Authentication time is invalid in Setup Request
CHSR_CRSP_NOMAXBW 9 = No Maximum test Bit rate specified
CHSR_CRSP_CAPEXC 10 = Server Maximum Bit rate exceeded
CHSR_CRSP_TRADITIONAL_MTU 11 = MTU option does not match Server

maxBandwidth Field MSB Code Bit:
CHSR_USDIR_BIT 0x8000 Bandwidth upstream direction bit, Set for Upstream

modifierBitmap Code Field: Setup
CHSR_JUMBO_STATUS 0x01 = set to use Jumbo datagram sizes above 1Gbps
CHSR_TRADITIONAL_MTU 0x02 = set to use datagrams for 1500 byte packets

There is a set of Command Response codes, beginning with: "2 = Bad Protocol Version", one of which SHOULD be communicated to indicate the cause when an error condition detected and testing cannot proceed:

2 = Bad Protocol Version
3 = Invalid Jumbo datagram option
5 = Authentication missing in Setup Request
4 = Unexpected Authentication in Setup Request
6 = Invalid authentication method (SHA-256 not used)
7 = Authentication failure (both shared secret and time)
8 = Authentication time is invalid in Setup Request (replay attack)
9 = No Maximum test Bit rate specified
10 = Server Maximum Bit rate exceeded
11 = MTU option does not match Server
The exceptional circumstances when a server would not communicate the appropriate Command Response Code for an error condition are when

1. the Setup Request PDU size is not correct (for supported versions of the protocol),
2. the control ID is invalid, or
3. a directed attack has been detected,

in which case the server will allow setup attempts to terminate silently. Attack detection is beyond the scope of this specification.

When indicating a Bad Protocol Version error, the server SHALL update the protocolVer field in the Setup Response to indicate the current version supported.

If the server finds that the Setup Request matches its configuration and is otherwise acceptable, the server SHALL initiate a new connection for the client, using a new UDP socket allocated from the UDP ephemeral port range. Then, the server SHALL start a watchdog timer (to terminate the connection in case the client goes silent), and sends the Setup Response back to the client (see below for composition).

When the Setup Request is accepted by the server, a Setup Response SHALL be sent back to the client with a corresponding command response value indicating 1 =Acknowledgement.

```
uint16_t controlId; // Control ID = 0xACE1
uint16_t protocolVer; // Protocol version = 0x08
uint8_t cmdRequest; // Command request = 2 (reply)
uint8_t cmdResponse; // Command response = 1 (Acknowledgement)
uint16_t maxBandwidth;// Required bandwidth (added in v9)
uint16_t testPort; // Test port on server (available port in Response )
uint8_t modifierBitmap;// Modifier bitmap (replaced jumboStatus for v9)
uint8_t authMode; // Authentication mode
uint32_t authUnixTime;// Authentication time stamp
unsigned char authDigest[AUTH_DIGEST_LENGTH] // 32 octets, MBZ
```

(Note: in version 8, the watchdog time-out is configured at 5 seconds)

The Setup Response SHALL include the port number at the server for the new socket, and this UDP port-pair SHALL be used for all subsequent communication. The server SHALL confirm or populate the values of:
Internet-Draft Test Protocol for IP Capacity Measurement July 2022

- Protocol version
- Jumbo datagram support status (modifierBitmap),
- Traditional MTU (modifierBitmap),
- Authentication mode (authDigest will be all zeroes if used)
- Required Bandwidth
- Test Port (ephemeral port)
- Modifier Bitmap
- authUnixTime retuned as-received

for the client’s use on the new connection in its Setup Response, and the authentication digest MUST Be Zero (MBZ).

Finally, the new UDP connection associated with the new socket and port number is opened, and the server awaits communication there.

If a Test Activation Request is not subsequently received from the client on this new port number before the watchdog timer expires, the server SHALL close the socket and deallocate the port.

5.2.1. Setup Response Procedure at the Client

When the client receives the Setup Response from the server, the client SHALL check:

1. the message PDU for correct length and formatting (fields have values in-range, beginning with the fields listed below)
2. the controlID, to validate the type of message (Setup)
3. the cmdResponse value

IF the cmdResponse value indicates an error the client SHALL display/report a relevant message to the user or management process and exit. If the client receives a Command Server Response code (CRSP) that is not equal to one of the codes defined above, then the client MUST terminate the connection and terminate operation of the current Setup Request. If the Command Server Response code (CRSP) value indicates success the client SHALL compose a Test Activation Request with all the test parameters it desires, such as the test direction, the test duration, etc.
6. Test Activation Request and Response

This section is divided according to the sending and processing of the client, server, and again at the client.

All messages defined in this section SHALL use UDP transport. The hosts SHALL calculate and include the UDP checksum, or check the UDP checksum as necessary.

6.1. Test Activation Request at the client

Upon a successful setup, the client SHALL then send the Test Activation Request to the UDP port number the server communicated in the Setup Response.

The client SHALL compose Test Activation Request as follows:
Control Header Test Activation Command Request Values:

- CHTA_CREQ_NONE (0) = No Request
- CHTA_CREQ_TESTACTUS (1) = Request test in Upstream direction (client to server, client takes the role of sending test packets)
- CHTA_CREQ_TESTACTDS (2) = Request test in Downstream direction (server to client, client takes the role of receiving test packets)

Control Header Test Activation Command Response Values:

- CHTA_CRSP_NONE (0) = Used by client when making a Request
- CHTA_CRSP_ACKOK (1) = Used by Server in affirmative Response
- CHTA_CRSP_BADPARAM (2) = Used by Server to indicate an error; bad parameter; reject

Note: uint16_t srIndexConf is the table index of the configured fixed or starting send rate (depending on whether CHTA_SRIDX_ISSTART is cleared or set respectively).

The server MAY allow the client to specify any fixed or starting send rate.

Otherwise, the server MAY enforce a maximum of the fixed or starting send rate which the client can successfully request. If the client’s
Test Activation Request exceeds the server’s configured maximum, the
server MUST either reject the request, or coerce the value to the
configured maximum, and communicate that maximum to the client in the
Test Activation Response. The client can of course choose to end the
test, as appropriate.

The UDP PDU format of the Test Activation Request is as follows (big-
endian AB):

0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| controlId            | protocolVer          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| cmdRequest           | cmdResponse           | lowThresh       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| upperThresh          | trialInt              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| testIntTime           | subIntPeriod          | ipTosByte      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| srIndexConf           | useOWDelVar           | highSpeedDelta |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| slowAdjThresh         | seqErrThresh          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| ignoreOooDup          | modifierBitmap        | rateAdjAlgo    | reserved1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Note: This is only 28 octets of the 56 octet PDU sent, the rest are MBZ
for a Test Activation Request.

The client SHALL use the configuration for

- cmdRequest (upstream or downstream)
- cmdResponse is cleared (all zeroes)
- and the remaining fields are populated based on default values or
  command-line options

requested in the Setup Request and confirmed by the server in the
Setup Response.

@@@@@ We could add the authDigest to the Test Activation request/
response. THEN, we would explain that

+ Use of optional Authenticated mode requires checking the validity
  of authDigest in this phase
+ The time stamp in the PDU MUST be within 5 minutes (+/- 2.5 minutes) of the current time at the recipient.

6.2. Test Activation Response

After the server receives the Test Activation Request on the new connection, it MUST choose to accept, ignore or modify any of the test parameters.

When the server sends the Test Activation Response, it SHALL set the cmd Response field to:

```c
uint8_t cmdResponse; // Command response (set to 1, ACK, or 2 error)
```

The server SHALL repeat all test parameters to indicate changes to the client.

If the client has requested an upstream test, the server SHALL

- include the transmission parameters from the first row of the sending rate table in the Sending Rate Structure (defined below), OR
- use the parameters from the configured send rate index (srIndexConf) of the sending rate table, or starting rate index (indicated in the Test Activation modifierBitmap) when these options are present.

The remaining 28 octets of the Test Activation Response (normally read from the first row of the sending rate table) are called the Sending Rate Structure, and SHALL be organized as follows:

```c
uint32_t txInterval1; // Transmit interval (us)
uint32_t udpPayload1; // UDP payload (bytes)
uint32_t burstSize1;  // UDP burst size per interval
uint32_t txInterval2; // Transmit interval (us)
uint32_t udpPayload2; // UDP payload (bytes)
uint32_t burstSize2;  // UDP burst size per interval
uint32_t udpAddon2;   // UDP add-on (bytes)
```

with
6.1.  Test Activation Request action at the client

When the client sends a Test Activation Request, it first checks the command response value:

```c
uint8_t cmdResponse; // Command response (set to 2, error)
```

If activation continues, the new connection is prepared for an upstream OR downstream test.

In the case of a downstream test, the server SHALL prepare to send with either a single timer to send status PDUs at the specified interval OR dual timers to send load PDUs based on

- the transmission parameters from the first row of the sending rate table in the Sending Rate Structure, OR
- the transmission parameters of the configured send rate index (srIndexConf) of the sending rate table, or starting rate index (indicated in the Test Activation modifierBitmap) when these options are present.

The server SHALL then send a Test Activation Response back to the client, update the watchdog timer with a new time-out value, and set a test duration timer to eventually stop the test.

The new connection is now ready for testing.

6.3.  Test Activation Response action at the client

When the client receives the Test Activation Response, it first checks the command response value.
If the client receives a Test Activation Command Response value that indicates an error, the client SHALL display/report a relevant message to the user or management process and exit.

If the client receives a Test Activation Command Response value that is not equal to one of the codes defined above, then the client MUST terminate the connection and terminate operation of the current Setup Request.

If the client receives a Test Activation Command Response value that indicates success (CHTA_CRSP_ACKOK) the client SHALL update its configuration to use any test parameters modified by the server.

Next, the client SHALL prepare its connection for either an upstream test with dual timers set to send load PDUs (based on the starting transmission parameters sent by the server), OR a downstream test with a single timer to send status PDUs at the specified interval.

Then, the client SHALL stop the test initiation timer, set a new time-out value for the watchdog timer, and start the timer (in case the server goes quiet).

The connection is now ready for testing.

7. Test Stream Transmission and Measurement Feedback Messages

This section describes the testing phase of the protocol. The roles of sender and receiver vary depending whether the direction of testing is from server to client, or the reverse.

All messages defined in this section SHALL use UDP transport. The hosts SHALL calculate and include the UDP checksum, or check the received UDP checksum before further processing, as necessary.

7.1. Test Packet PDU and Roles

Testing proceeds with one end point sending load PDUs, based on transmission parameters from the sending rate table, and the other end point receiving the load PDUs and sending status messages to communicate the traffic conditions at the receiver.

The watchdog timer at the receiver SHALL be reset each time a test PDU is received. See non-graceful test stop in Section 8 for handling the watchdog/NOTRAFFIC time-out expiration at each end-point.

When the server is sending Load PDUs in the role of sender, it SHALL use the transmission parameters directly from the sending rate table.
via the index that is currently selected (which was based on the feedback in its received status messages).

However, when the client is sending load PDUs in the role of sender, it SHALL use the discreet transmission parameters that were communicated by the server in its periodic status messages (and not referencing a sending rate table). This approach allows the server to control the individual sending rates as well as the algorithm used to decide when and how to adjust the rate.

The server uses a load adjustment algorithm which evaluates measurements, either it’s own or the contents of received feedback messages. This algorithm is unique to udpst; it provides the ability to search for the Maximum IP Capacity that is absent from other testing tools. Although the algorithm depends on the protocol, it is not part of the protocol per se.

The current algorithm (B) has three paths to its decision on the next sending rate:

1. When there are no impairments present (no sequence errors, low delay variation), resulting in sending rate increase.

2. When there are low impairments present (no sequence errors but higher levels of delay variation), so the same sending rate is retained.

3. When the impairment levels are above the thresholds set for this purpose and "congestion" is inferred, resulting in sending rate decrease.

The algorithm also has two modes for increasing/decreasing the sending rate:

o A high-speed mode to achieve high sending rates quickly, but also back-off quickly when "congestion" is inferred from the measurements. Any two consecutive feedback intervals that have a sequence number anomaly and/or contain an upper delay variation threshold exception in both of the two consecutive intervals, count as the two consecutive feedback measurements required to declare "congestion" within a test.

o A single-step mode where all rate adjustments use the minimum increase or decrease of one step in the sending rate table. The single step mode continues after the first inference of "congestion" from measured impairments.
On the other hand, the test configuration MAY use a fixed sending rate requested by the client, using the field below:

uint16_t srIndexConf; // Configured sending rate index

The client MAY communicate the desired fixed rate in its activation request. The reasons to conduct a fixed-rate test include stable measurement at the maximum determined by the load adjustment (search) algorithm, or the desire to test at a known subscribed rate without searching.

The Load PDU SHALL have the following format and field definitions:

uint16_t loadId; // Load ID (=0xBEFF for the LOAD PDU)
uint8_t testAction; // Test action (= 0x00 normally, until test stop)
uint8_t rxStopped; // Receive traffic stopped indicator (BOOL)
uint32_t lpduSeqNo; // Load PDU sequence number (starts at 1)
uint16_t udpPayload; // UDP payload LENGTH(bytes)
uint16_t spduSeqErr; // Status PDU sequence error count

uint32_t spduTime_sec; // Send time in last received status PDU
uint32_t spduTime_nsec; // Send time in last received status PDU
uint32_t lpduTime_sec; // Send time of this load PDU
uint32_t lpduTime_nsec; // Send time of this load PDU

Test Action Codes
TEST_ACT_TEST 0 // normal
TEST_ACT_STOP1 1 // normal stop at end of test: server sends in STATUS or Test PDU
TEST_ACT_STOP2 2 // ACK of STOP1: sent by client in STATUS or Test PDU

The Test Load UDP PDU format is as follows (big-endian AB):
7.2. Status PDU

The receiver SHALL send a Status PDU to the sender during a test at the configured feedback interval.

The watchdog timer at the test PDU sender SHALL be reset each time a Status PDU is received. See non-graceful test stop in Section 8 for handling the watchdog/NOTRAFFIC time-out expiration at each endpoint.

@@@@ To Do: What protections from bit errors (checksum) or on-path attacks (something stronger) are warranted for the Status PDUs? These PDUs are a key part of the server-client control loop. Added a requirement to calculate and include/check the UDP checksum.

The Status Header PDU SHALL have the following format and field definitions:
// Status feedback header for UDP payload of status PDUs

uint16_t statusId; // Status ID = 0xFEED
uint8_t testAction; // Test action
uint8_t rxStopped; // Receive traffic stopped indicator (BOOL)
uint32_t spduSeqNo; // Status PDU sequence number (starts at 1)

struct sendingRate srStruct; // Sending Rate Structure (28 octets)

uint32_t subIntSeqNo; // Sub-interval sequence number
struct subIntStats sisSav; // Sub-interval Saved Stats Structure (52 octets)

uint32_t seqErrLoss; // Loss sum
uint32_t seqErrOoo; // Out-of-Order sum
uint32_t seqErrDup; // Duplicate sum

uint32_t clockDeltaMin; // Clock delta minimum (either RTT or 1-way delay)

uint32_t delayVarMin; // Delay variation minimum
uint32_t delayVarMax; // Delay variation maximum
uint32_t delayVarSum; // Delay variation sum
uint32_t delayVarCnt; // Delay variation count
uint32_t rttMinimum; // Minimum round-trip time sampled
uint32_t rttSample; // Last round-trip time sample
uint8_t delayMinUpd; // Delay minimum(s) updated observed, communicated in both directions.

uint8_t reserved2; // (alignment)
uint16_t reserved3; // (alignment)

uint32_t tiDeltaTime; // Trial interval delta time
uint32_t tiRxDatagrams; // Trial interval receive datagrams
uint32_t tiRxBytes; // Trial interval receive bytes

uint32_t spduTime_sec; // Send time of this status PDU
uint32_t spduTime_nsec; // Send time of this status PDU

The Status feedback UDP payload PDUs format is as follows (big-endian AB):
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>statusId</td>
<td>testAction</td>
<td>rxStopped</td>
</tr>
<tr>
<td></td>
<td>+------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>spduSeqNo</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>Sending Rate Structure (28 octets)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>subIntSeqNo</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>Sub-interval Saved Stats Structure (52 octets)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>seqErrLoss</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>seqErrOoo</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>seqErrDup</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>clockDeltaMin</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>delayVarMin</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>delayVarMax</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>delayVarSum</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>delayVarCnt</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>rttMinimum</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>rttSample</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>delayMinUpd</td>
<td>reserved2</td>
<td>reserved3</td>
</tr>
<tr>
<td></td>
<td>+------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>tiDeltaTime</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>tiRxDatagrams</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>tiRxBytes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>spduTime_sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>spduTime_nsec</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note that the Sending Rate Structure (28 octets) is defined in the Test Activation section.
Also note that the Sub-interval Saved Stats Structure (52 octets) SHALL be included (and populated as required when the server is in the receiver role) as defined below.

The Sub-interval saved statistics structure for received traffic measurements SHALL be organized and formatted as follows:
### Statistical Summary Structure

```c
uint32_t rxDatagrams; // Received datagrams
uint32_t rxBytes;     // Received bytes
uint32_t deltaTime;   // Time delta
uint32_t seqErrLoss;  // Loss sum
uint32_t seqErrOoo;   // Out-of-Order sum
uint32_t seqErrDup;   // Duplicate sum
uint32_t delayVarMin; // Delay variation minimum
uint32_t delayVarMax; // Delay variation maximum
uint32_t delayVarSum; // Delay variation sum
uint32_t delayVarCnt; // Delay variation count
uint32_t rttMinimum;  // Minimum round-trip time
uint32_t rttMaximum;  // Maximum round-trip time
uint32_t accumTime;   // Accumulated time
```

---

Note that the 52 octet saved statistics structure above has slight differences from the 40 octets that follow in the status feedback PDU, particularly the time-related fields.
Upon receiving the Status Feedback PDU or expiration of the feedback interval, the server SHALL perform calculations required by the Load adjustment algorithm and adjust its sending rate, or signal that the client do so in its role as as sender.

@@@ To Do: Additional measurements, like interface byte counters from a client at a residential gateway, would change the Status Feedback PDU (and the protocol version number as a result). Interface byte counters seem useful for specific circumstances, such as when the client application has access to an interface that sees all traffic to/from a service subscriber’s location.

8. Stopping the Test

When the test duration timer on the server expires, it SHALL set the connection test action to STOP and mark all outgoing load or status PDUs with a test action of STOP1.

```c
uint8_t testAction; // Test action (server sets STOP1)
```

This is simply a non-reversible state for all future messages sent from the server.

When the client receives a load or status PDU with the STOP1 indication, it SHALL finalize testing, display the test results, and also mark its connection with a test action of STOP (so that any PDUs received subsequent to the STOP1 are ignored).

With the test action of the client’s connection set to STOP, the very next expiry of a send timer for either a load or status PDU SHALL cause the client to schedule an immediate end time to exit.

The client SHALL then send all subsequent load or status PDUs with a test action of STOP2

```c
uint8_t testAction; // Test action (client sets STOP2)
```

as confirmation to the server, and a graceful termination of the test can begin.

When the server receives the STOP2 confirmation in the load or status PDU, the server SHALL schedule an immediate end time for the connection which closes the socket and deallocates it.

In a non-graceful test stop, the watchdog/NOTRAFFIC time-outs at each end-point will expire (sometimes at one end-point first), notifications in logs, STDOUT, and/or formatted output SHALL be made,
and the test action of each end-point’s connection SHALL be set to STOP.

9. Method of Measurement

The architecture of the method REQUIRES two cooperating hosts operating in the roles of Src (test packet sender) and Dst (receiver), with a measured path and return path between them.

The duration of a test duration, parameter I, MUST be constrained in a production network, since this is an active test method and it will likely cause congestion on the Src to Dst host path during a test.

9.1. Running Code

This section is for the benefit of the Document Shepherd’s form, and will be deleted prior to final review.

Much of the development of the method and comparisons with existing methods conducted at IETF Hackathons and elsewhere have been based on the example udpst Linux measurement tool (which is a working reference for further development) [udpst]. The current project:

- is a utility that can function as a client or server daemon
- requires a successful client-initiated setup handshake between cooperating hosts and allows firewalls to control inbound unsolicited UDP which either go to a control port [expected and w/ authentication] or to ephemeral ports that are only created as needed. Firewalls protecting each host can both continue to do their job normally. This aspect is similar to many other test utilities available. The firewall at the server will need to open a limited range of ephemeral ports to complete the second exchange: Test Activation (where the client communicates to the server on an ephemeral destination port *assigned by the server*).  
- is written in C, and built with gcc (release 9.3) and its standard run-time libraries
- allows configuration of most of the parameters described in Sections 4 and 7.
- supports IPv4 and IPv6 address families.
- supports IP-layer packet marking.
10. Security Considerations

Active metrics and measurements have a long history of security considerations. The security considerations that apply to any active measurement of live paths are relevant here. See [RFC4656] and [RFC5357].

When considering privacy of those involved in measurement or those whose traffic is measured, the sensitive information available to potential observers is greatly reduced when using active techniques which are within this scope of work. Passive observations of user traffic for measurement purposes raise many privacy issues. We refer the reader to the privacy considerations described in the Large Scale Measurement of Broadband Performance (LMAP) Framework [RFC7594], which covers active and passive techniques.

There are some new considerations for Capacity measurement as described in this memo.

1. Cooperating client and server hosts and agreements to test the path between the hosts are REQUIRED. Hosts perform in either the server or client roles. One way to assure a cooperative agreement employs the optional Authorization mode through the use of the authDigest field and the known identity associated with the key used to create the authDigest field. Other means are also possible, such as access control lists at the server.

2. It is REQUIRED to have a user client-initiated setup handshake between cooperating hosts that allows firewalls to control inbound unsolicited UDP traffic which either goes to a control port [expected and w/authentication] or to ephemeral ports that are only created as needed. Firewalls protecting each host can both continue to do their job normally.

3. Client-server authentication and integrity protection for feedback messages conveying measurements is RECOMMENDED. To accommodate different host limitations and testing circumstances, different modes of operation are recommended:

WG ver 02 proposal/discussion below:

A. Unauthenticated mode (for all phases)

AND

B. OPTIONAL Authenticated set-up only

SHA-256 HMAC time-window verification (5 min time stamp verification)

(could add silent failure option)

New: we could add authDigest everywhere that is possible, as you suggested below.
*** We would like a SEC-DIR recommendation to accomplish C and/or D below:

C. Encrypted Setup Exchange in a tunnel to well-known port:
   (remaining transmissions are on a new UDP port-pair, in the clear)
   New: could combine Test Activation exchange with Setup, on the well-known port, encrypted.
   Need a packet to open the firewall from client to server.

D. Encrypt "all the things"
   (Reduce the options, provide the required protocol protection)

while keeping the following design criteria in mind:
+ the accuracy <-> integrity trade-off (lightweight encryption may see more deployment)
+ synergy: we are already using the OpenSSL library in the running code (for Authentication)

New: we think this mode D might not be used very often, the demands on hosts to generate and measure at Gbps rates usually require all the cycles they can allocate to the measurement process.

Pre-WG 00 proposal below:

A. Unauthenticated mode (for all phases)
   AND
B. OPTIONAL Authenticated set-up only
   SHA-256 HMAC time-window verification (5 min time stamp verification)
   (could add silent failure option)

   =-=-=-=-=-=-=-=-=-=- Above options exist in Running Code -=-=-=-=-=-=

C. Encrypted setup and test-activation
   (currently using OpenSSL Library, so KISS, but may be too slow for test packets)

   -=-=-=-=-=-=-=-=-=-= Old/lowpower host performance impacts -=-=-=-=-=-=

D. Encrypted feedback messages (maybe split into Integrity and encrypt?)

E. Integrity protection for test packets SHA-256 HMAC

F. Encrypted test packets (maybe also valuable to defeat compression on links)

4. Hosts MUST limit the number of simultaneous tests to avoid resource exhaustion and inaccurate results.
5. Senders MUST be rate-limited. This can be accomplished using a pre-built table defining all the offered load rates that will be supported (Section 8.1). The recommended load-control search algorithm results in "ramp up" from the lowest rate in the table.

6. Service subscribers with limited data volumes who conduct extensive capacity testing might experience the effects of Service Provider controls on their service. Testing with the Service Provider’s measurement hosts SHOULD be limited in frequency and/or overall volume of test traffic (for example, the range of I duration values SHOULD be limited).

The exact specification of these features was hopefully accomplished during this protocol development.

11. IANA Considerations

This memo requests IANA to assign a "well-known" UDP port for the Test Setup phase of protocol operation.

12. Acknowledgments

Thanks to Ruediger Geib, Lincoln Lavoie, Can Desem, and Greg Mirsky for reviewing this draft and providing helpful suggestions and areas for further development. Ken Kerpez and Chen Li have provided helpful reviews.

Brian Weis provided an early SEC-DIR review; version 02 captures clarifications and further versions will take on the protocol changes suggested.

13. References

13.1. Normative References

[I-D.ietf-ippm-capacity-metric-method]

13.2. Informative References


Ciavatteone & Morton Expires January 10, 2023 [Page 29]
[LS-SG12-A]

[LS-SG12-B]


Authors’ Addresses

Len Ciavattone
AT&T Labs
200 Laurel Avenue South
Middletown,, NJ  07748
USA

Email: lencia@att.com

Al Morton
AT&T Labs
Chicago,, IL  60660
USA

Phone: +1 732 420 1571
Email: acmorton@att.com
Abstract

RFC8250 describes an optional Destination Option (DO) header embedded in each packet to provide sequence numbers and timing information as a basis for measurements. As this data is sent in clear-text, this may create an opportunity for malicious actors to get information for subsequent attacks. This document defines PDMv2 which has a lightweight handshake (registration procedure) and encryption to secure this data. Additional performance metrics which may be of use are also defined.

Status of This Memo

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

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

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

This Internet-Draft will expire on 22 December 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.
Table of Contents

1. Introduction ............................................. 3
  1.1. Current Performance and Diagnostic Metrics (PDM) .......... 3
  1.2. PDMv2 Introduction .................................... 3
2. Conventions used in this document .............................. 3
3. Terminology .............................................. 4
4. Protocol Flow ............................................. 4
  4.1. Registration Phase ................................... 5
    4.1.1. Rationale of Primary (Writer) and Secondary (Reader)
    Roles ..................................................... 5
    4.1.2. Diagram of Registration Flow ......................... 5
  4.2. Primary (Writer) Client - Primary (Writer) Server
    Negotiation Phase .................................... 5
  4.3. Primary (Writer) Server / Client - Secondary (Reader)
    Server / Client Registration Phase ....................... 6
  4.4. Secondary (Reader) Client - Secondary (Reader) Server
    communication ......................................... 6
5. Security Goals ............................................. 7
  5.1. Security Goals for Confidentiality ....................... 7
  5.2. Security Goals for Integrity ........................... 7
  5.3. Security Goals for Authentication ....................... 7
  5.4. Cryptographic Algorithm ................................ 8
6. PDMv2 Destination Options ................................... 8
  6.1. Destinations Option Header ................................ 8
  6.2. Metrics information in PDMv2 ........................... 8
  6.3. PDMv2 Layout ......................................... 9
7. Security Considerations ..................................... 12
8. Privacy Considerations .................................... 15
9. IANA Considerations ....................................... 16
10. Contributors ............................................. 16
11. References ............................................... 16
    11.1. References ......................................... 16
    11.2. Normative References ................................ 16
    11.3. Informative References ............................ 16
Appendix A. Rationale for Primary (Writer) Server / Primary
    (Writer) Client .......................................... 16
    A.1. One Client / One Server ................................ 16
    A.2. Multiple Clients / One Server ......................... 17
1. Introduction

1.1. Current Performance and Diagnostic Metrics (PDM)

The current PDM is an IPv6 Destination Options header which provides information based on the metrics like Round-trip delay and Server delay. This information helps to measure the Quality of Service (QoS) and to assist in diagnostics. However, there are potential risks involved transmitting PDM data during a diagnostics session.

PDM metrics can help an attacker understand about the type of machine and its processing capabilities. Inferring from the PDM data, the attack can launch a timing attack. For example, if a cryptographic protocol is used, a timing attack may be launched against the keying material to obtain the secret.

Along with this, PDM does not provide integrity. It is possible for a Man-In-The-Middle (MITM) node to modify PDM headers leading to incorrect conclusions. For example, during the debugging process using PDM header, it can mislead the person showing there are no unusual server delays.

1.2. PDMv2 Introduction

PDMv2 introduces confidential, integrity and authentication.

TBD

2. Conventions used in this document

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 BCP 14, RFC 2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying significance described in RFC 2119.
3. Terminology

* Primary (Writer) Client (WC): An authoritative node that creates cryptographic keys for multiple reader clients.

* Primary (Writer) Server (WS): An authoritative node that creates cryptographic keys for multiple reader servers.

* Secondary (Reader) Client (RC): An endpoint node which initiates a session with a listening port and sends PDM data. Connects to the Primary (Writer) Client to get cryptographic key material.

* Secondary (Reader) Server (RS): An endpoint node which has a listening port and sends PDM data. Connects to the Primary (Writer) Server to get cryptographic key material.

Note: a client may act as a server (have listening ports).

* Symmetric Key (K): A uniformly random bitstring as an input to the encryption algorithm, known only to Secondary (Reader) Clients and Secondary (Reader) Servers, to establish a secure communication.

* Public and Private Keys: A pair of keys that is used in asymmetric cryptography. If one is used for encryption, the other is used for decryption. Private Keys are kept hidden by the source of the key pair generator, but Public Key is known to everyone. pkX (Public Key) and skX (Private Key). Where X can be, any client or any server.

* Pre-shared Key (PSK): A symmetric key. Uniformly random bitstring, shared between any client or any server or a key shared between an entity that forms client-server relationship. This could happen through an out-of-band mechanism: e.g., a physical meeting or use of another protocol.

* Session Key: A temporary key which acts as a symmetric key for the whole session.

4. Protocol Flow

The protocol will proceed in 3 steps.

Step 1: Negotiation between Primary (Writer) Server and Primary (Writer) Client.

Step 2: Registration between Primary (Writer) Server / Client and Secondary (Reader) Server / Client
4.1. Registration Phase

4.1.1. Rationale of Primary (Writer) and Secondary (Reader) Roles

Enterprises have many servers and many clients. These clients and servers may be in multiple locations. It may be less overhead to have a secure location (ex. Shared database) for servers and clients to share keys. Otherwise, each client needs to keep track of the keys for each server.

Please view Appendix 1 for some sample topologies and further explanation.

4.1.2. Diagram of Registration Flow

```
+-------------+                       +-------------+
|   Writer    |<--------------------->|   Writer    |
|   Client    |                       |   Server    |
+------+-----+                       +------+-----+
                                    |
+----------+----------+              +----------+----------+
|          |          |              |          |          |
+---+---+  +---+---+  +---+---+      +---+---+  +---+---+  +---+---+
| Reader|  | Reader|  | Reader|      | Reader|  | Reader|  | Reader|
|   1   |  |   2   |  |   3   |      |   1   |  |   2   |  |   3   |
+---+---+  +---+---+  +---+---+      +---+---+  +---+---+  +---+---+
|          |          +--------------+          |          |
|          +------------------------------------+          |
+----------------------------------------------------------+
```

4.2. Primary (Writer) Client - Primary (Writer) Server Negotiation Phase

The two entities exchange a set of data to ensure the respective identities.

They use HPKE KEM to negotiate a "SharedSecret".
4.3. Primary (Writer) Server / Client - Secondary (Reader) Server / Client Registration Phase

The "SharedSecret" is shared securely:

* By the Primary (Writer) Client to all the Secondary (Reader) Clients under its control. How this is achieved is beyond the scope of the present specification.

* By the Primary (Writer) Server to all the Secondary (Reader) Servers under its control. How this is achieved is beyond the scope of the present specification.

4.4. Secondary (Reader) Client - Secondary (Reader) Server communication

Each Client and Server derive a "SessionTemporaryKey" by using HPKE KDF, using the following inputs:

* The "SharedSecret".

* The 5-tuple (SrcIP, SrcPort, DstIP, DstPort, Protocol) of the communication.

* A Key Rotation Index (Kri).

The Kri SHOULD be initialized to zero.

The server and client initialize (separately) a pseudo-random non-repeating sequence between 1 and $2^{15}-1$. How to generate this sequence is beyond the scope of this document, and does not affect the rest of the specification. When the sequence is used fully, or earlier if appropriate, the sender signals the other party that a key change is necessary. This is achieved by flipping the "F bit" and resetting the PRSEQ. The receiver increments the Kri of the sender, and derives another SessionTemporaryKey to be used for decryption.

It shall be stressed that the two SessionTemporaryKeys used in the communication are never the same, as the 5-tuple is reversed for the Server and Client. Moreover, the time evolution of the respective Kri can be different. As a consequence, each entity must maintain a table with (at least) the following informations:

* Flow 5-tuple, Own Kri, Other Kri

An implementation might optimize this further by caching the OwnSessionTemporaryKey (used in Encryption) and OtherSessionTemporaryKey (used in Decryption).
5. Security Goals

As discussed in the introduction, PDM data can represent a serious data leakage in presence of a malicious actor.

In particular, the sequence numbers included in the PDM header allows correlating the traffic flows, and the timing data can highlight the operational limits of a server to a malicious actor. Moreover, forging PDM headers can lead to unnecessary, unwanted, or dangerous operational choices, e.g., to restore an apparently degraded Quality of Service (QoS).

Due to this, it is important that the confidentiality and integrity of the PDM headers is maintained. PDM headers can be encrypted and authenticated using the methods discussed in section [x], thus ensuring confidentiality and integrity. However, if PDM is used in a scenario where the integrity and confidentiality is already ensured by other means, they can be transmitted without encryption or authentication. This includes, but is not limited to, the following cases:

a) PDM is used over an already encrypted medium (For example VPN tunnels).

b) PDM is used in a link-local scenario.

c) PDM is used in a corporate network where there are security measures strong enough to consider the presence of a malicious actor a negligible risk.

5.1. Security Goals for Confidentiality

PDM data must be kept confidential between the intended parties, which includes (but is not limited to) the two entities exchanging PDM data, and any legitimate party with the proper rights to access such data.

5.2. Security Goals for Integrity

PDM data must not be forged or modified by a malicious entity. In other terms, a malicious entity must not be able to generate a valid PDM header impersonating an endpoint, and must not be able to modify a valid PDM header.

5.3. Security Goals for Authentication

TBD
5.4. Cryptographic Algorithm

Symmetric key cryptography has performance benefits over asymmetric cryptography; asymmetric cryptography is better for key management. Encryption schemes that unite both have been specified in [RFC1421], and have been participating practically since the early days of public-key cryptography. The basic mechanism is to encrypt the symmetric key with the public key by joining both yields. Hybrid public-key encryption schemes (HPKE) [RFC9180] used a different approach that generates the symmetric key and its encapsulation with the public key of the receiver.

Our choice is to use the HPKE framework that incorporates key encapsulation mechanism (KEM), key derivation function (KDF) and authenticated encryption with associated data (AEAD). These multiple schemes are more robust and significantly efficient than the traditional schemes and thus lead to our choice of this framework.

6. PDMv2 Destination Options

6.1. Destinations Option Header

The IPv6 Destination Options extension header [RFC8200] 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 RFC 8200 [RFC8200]. The IPv6 PDMv2 destination option is implemented as an IPv6 Option carried in the Destination Options header.

6.2. Metrics information in PDMv2

The IPv6 PDMv2 destination option contains the following base fields:

- SCALEDTLR: Scale for Delta Time Last Received
- SCALEDTLS: Scale for Delta Time Last Sent
- GLOBALPTR: Global Pointer
- PSNTP: Packet Sequence Number This Packet
- PSNLNR: Packet Sequence Number Last Received
- DELTATLR: Delta Time Last Received
- DELTATLS: Delta Time Last Sent

PDMv2 adds a new metric to the existing PDM [RFC8250] called the Global Pointer. The existing PDM fields are identified with respect to the identifying information called a "5-tuple".

The 5-tuple consists of:
SADDR: IP address of the sender
SPORT: Port for the sender
DADDR: IP address of the destination
DPORT: Port for the destination
PROTC: Upper-layer protocol (TCP, UDP, ICMP, etc.)

Unlike PDM fields, Global Pointer (GLOBALPTR) field in PDMv2 is defined for the SADDR type. Following are the SADDR address types considered:

a) Link-Local

b) Global Unicast

The Global Pointer is treated as a common entity over all the 5-tuples with the same SADDR type. It is initialised to the value 1 and increments for every packet sent. Global Pointer provides a measure of the amount of IPv6 traffic sent by the PDMv2 node.

When the SADDR type is Link-Local, the PDMv2 node sends Global Pointer defined for Link-Local addresses, and when the SADDR type is Global Unicast, it sends the one defined for Global Unicast addresses.

6.3. PDMv2 Layout

PDMv2 has two different header formats corresponding to whether the metric contents are encrypted or unencrypted. The difference between the two types of headers is determined from the Options Length value.

Following is the representation of the unencrypted PDMv2 header:

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Option Type  | Option Length | Vrsn  |     Reserved Bits     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Random Number          |f|   ScaleDTLR   |   ScaleDTLS   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Global Pointer                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      PSN This Packet          |    PSN Last Received          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Delta Time Last Received    |     Delta Time Last Sent      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
Following is the representation of the encrypted PDMv2 header:

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Option Type  | Option Length | Vrsn  |     Reserved Bits     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Random Number          |f|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               :
|                      Encrypted PDM Data                       |
:                          (30 bytes)                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Option Type

0x0F

8-bit unsigned integer. The Option Type is adopted from RFC 8250 [RFC8250].

Option Length

0x12: Unencrypted PDM

0x22: Encrypted PDM

8-bit unsigned integer. Length of the option, in octets, excluding the Option Type and Option Length fields. The options length is used for differentiating PDM [RFC8250], unencrypted PDMv2 and encrypted PDMv2.

Version Number

0x2

4-bit unsigned number.

Reserved Bits

12-bits.

Reserved bits for future use. They are initialised to 0 for PDMv2.

Random Number
15-bit unsigned number.

TBD

Flag Bit

1-bit field.

TBD

Scale Delta Time Last Received (SCALEDTLR)

8-bit unsigned number.

This is the scaling value for the Delta Time Last Sent (DELTATLS) field.

Scale Delta Time Last Sent (SCALEDTLS)

8-bit unsigned number.

This is the scaling value for the Delta Time Last Sent (DELTATLS) field.

Global Pointer

32-bit unsigned number.

Global Pointer is initialized to 1 for the different source address types and incremented monotonically for each packet with the corresponding source address type.

This field stores the Global Pointer type corresponding to the SADDR type of the packet.

Packet Sequence Number This Packet (PSNT)

16-bit unsigned number.

This field is initialized at a random number and is incremented monotonically for each packet of the 5-tuple.

Packet Sequence Number Last Recieved (PSNLR)

16-bit unsigned number.

This field is the PSNT of the last received packet on the 5-tuple.
Delta Time Last Received (DELTATLR)

16-bit unsigned integer.

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)

16-bit unsigned integer.

The value is set according to the scale in SCALEDTLS.

Delta Time Last Sent = (receive time packet n - send time packet (n - 1))

7. Security Considerations

PDMv2 DOH can be used by an attacker to gather information about a victim (passive attack) or to force the victim to modify its operational parameters to comply with forged data (active attacks).

In order to mitigate these, it is important that the PDMv2 DOH is subject to:

1) Confidentiality and
2) Integrity

with respect to an attacker.

In the following we will refer to two different "groups", that can or cannot belong to the same operational and management domain:

1) Servers - implementing services.
2) Clients-devices willing to interact with the services offered by Servers.

We will assume, for the sake of generalization, that the Servers are managed by an Organization (OrgA) implementing management procedures over them, and the Clients by a different Organization (OrgB).

An attacker could be in the following positions:

1) External to OrgA or OrgB.
2) Inside OrgA (i.e., a Server), either because it is a legitimate-but-curious device, or as a consequence of an attack to a device.

3) Inside OrgB (i.e., a Client), either because it is a legitimate-but-curious device, or as a consequence of an attack to a device.

Furthermore, since PDMv2 DOH encryption could consume resources (albeit limited), it is possible to foresee a call of DoS by resource exhaustion. Hence, it is relevant to consider a form of access control to verify that the Server and Client belong to OrgA and OrgB respectively. This could be a _delegated trust_.

In other terms, a Client could just want to verify that the Server belongs to OrgA, without actually verifying the identity of the Server.

The Authentication and Authorization of Clients and Servers is thus delegated to the respective Organizations. In other terms, we do not expect, or want, that a Client and a Server should be forced to verify the respective identities (Authentication) or the permissions to use PDMv2 (Authorization).

The simple knowledge of the secrets required by the flow is considered sufficient to enable PDMv2. On the opposite, an unsuccessful decryption MUST result in dropping the PDMv2 DOH without further processing or, if configured to do so, might lead to throttling, filtering, and/or logging the activity of the other entity (Client or Server).

The present document specifies a methodology to enable this delegated trust, along with the Confidentiality and Integrity requirements, in the PDMv2 DOH.

We assume that PS and PC have verified the respective identities and the authorization to enable PDMv2 DOH on a set of devices under their responsibility: Secondary Servers (SS) and Secondary Clients (SC).

**PS-PC**

* Perform a HPKE KEM and obtain a PairMasterSecret (PMS).

* The PMS is stored securely in both PS and PC, and is NOT to be leaked.

* The PMS is valid only for the PC-PS pair.

In other terms, if a PS would want to establish a pair with two PCs, it will have two different PMSs.
* PMS might be re-negotiated after a given amount of time
  [renegotiation TBD]

* PS and PC exchange respectively the list of the SS and SC enabled
to use PDMv2. The list can be:

  - A range of IP addresses, e.g.: 2001:db8:food:beef:cafe::0/80
  - A list of IP addresses, e.g., [2001:db8:food::1/128,
    2001:db8:food::1/128]

Note:

1) How to represent the list in a compact way is out of scope of
   the present document,

2) The list could be dynamically updated.

3) Inside OrgB (i.e., a Client), either because it is a
   legitimate-but-curious device, or as a consequence of an
   attack to a device

* PS sends to the PC the Security Mode of Operation (SecMoP) to be
  used, see below.

PS-SS and PC-SC

* Each Secondary Sever (or Client) MUST authenticate itself with the
  Primary Server (or Client). This is out of scope of the present
  specification.

* Each SS receives a PairServerSecret (PSS), derived using HPKE KDF,
  and valid for the specific SS and the list of SCs defined above.

* Each SC receives a PairClientSecret (PCS), derived using HPKE KDF,
  and valid for the specific SC and the list of SSs defined above.

Since there are multiple use-cases, we define 4 modes of operations:

* *No Protection*: The Secrets are discarded (or not even created),
  and the flows do not use PDMv2. The scheme above is used only to
  disseminate the list of Secondary Clients and Secondary Servers.
  By sharing lists, this mode act as ACL (Access Control List) or
  authorization of the secondaries.

* *TrustedServers*: The Secondary Servers are trusted, and they do
  know a secret derived by the PMS.
* *AsymmetricPoll*: One Secondary (Server or Client) must acquire a secret from the respective Primary.

* *Identity Based Cryptography (IBC)*: IBC (RFC5091) is used to generate a shared secret between the SS and the SC.

The *TrustedServers* MoP has the benefit of requiring no additional steps to send and receive PDMv2 DOH, because each flow is protected by a SessionKey that can be derived autonomously by both the SC and the SS, without any interaction with the PS and PC, or any negotiation between the SS and the SC.

The possible vulnerabilities of the *TrustedServers* MoP are the following:

* Any SS can inspect the flows directed to a different SS in the same group.

* An attack to a SS might result in compromising the security of all the flows between all the clients and the Secondary Servers belonging to the same group.

A possible mitigation is to split the Secondary Servers in different sub-groups. This is a scenario similar to the one of a PC negotiating PDMv2 access with different PSs.

The *AsymmetricPoll* MoP has the benefit of isolating each SS and each SC. Only the SS and SC involved in a communication can decrypt their flows.

The *IBC* MoP has the same security properties of the *AsymmetricPoll* MoP, and the advantage of not requiring any interaction between the Primary and the Secondary. The disadvantage is the requirement of performing a "pairing" session negotiation between the Secondaries.

It must be considered that, while secure, this MoP could be used to perform a resource exhaustion attack on the PairDeviceKey establishment. Hence, a device MUST NOT reply to an IP address that is not in the Secondary[client, server] list, and MUST NOT reply with negative acknowledgments (e.g., in case of an incorrect decoding).

8. Privacy Considerations

TBD
9. IANA Considerations

TBD

10. Contributors

TBD

11. References

11.1. References

11.2. Normative References


11.3. Informative References


Appendix A. Rationale for Primary (Writer) Server / Primary (Writer) Client

A.1. One Client / One Server

Let’s start with one client and one server.
The Client and Server create public / private keys and derive a shared secret. Let’s not consider Authentication or Certificates at this point.

What is stored at the Client and Server to be able to encrypt and decrypt packets? The shared secret or private key.

Since we only have one Server and one Client, then we don’t need to have any kind of identifier for which private key to use for which Server or Client because there is only one of each.

Of course, this is a ludicrous scenario since no real organization of interest has only one server and one client.

A.2. Multiple Clients / One Server

So, let’s try with multiple clients and one Primary (Writer) server

The Clients and Server create public / private keys and derive a shared secret. Each Client has a unique private key.

What is stored at the Client and Server to be able to encrypt and decrypt packets?

Clients each store a private key. Server stores: Client Identifier and Private Key.
Since we only have one Server and multiple Clients, then the Clients don’t need to have any kind of identifier for which private key to use for which Server but the Server needs to know which private key to use for which Client. So, the Server has to store an identifier as well as the Key.

But, this also is a ludicrous scenario since no real organization of interest has only one server.

A.3. Multiple Clients / Multiple Servers

When we have multiple clients and multiple servers, then each not only does the Server need to know which key to use for which Client, but the Client needs to know which private key to use for which Server.

A.4. Primary (Writer) Client / Primary (Writer) Server

Based on this rationale, we have chosen a Primary (Writer) Server / Primary (Writer) Client topology.

Appendix B. Sample Implementation of Registration

B.1. Overall summary

In the Registration phase, the objective is to generate a shared secret that will be used in encryption and decryption during the Data Transfer phase. We have adopted a Primary-Secondary architecture to represent the clients and servers (see Section 4.1.1). The primary server and primary client perform Key Encapsulation Mechanism (KEM) [RFC9180] to generate a primary shared secret. The primary server shares this secret with secondary servers, whereas the primary client performs Key Derivation Function (KDF) [RFC9180] to share client-specific secrets to corresponding secondary clients. During the Data Transfer phase, the secondary servers generate the client-specific secrets on the arrival of the first packet from the secondary client.

B.2. High level flow

The following steps describe the protocol flow:

1. Primary client initiates a request to the primary server. The request contains a list of available ciphersuites for KEM, KDF, and AEAD.

2. Primary server responds to the primary client with one of the available ciphersuites and shares its public key.
3. Primary client generates a secret and its encapsulation. The primary client sends the encapsulation and a salt to the primary server. The salt is required during KDF in the Data Transfer phase.

4. Primary Server generates the secret with the help of the encapsulation and responds with a status message.

5. Primary server shares this key with secondary servers over TLS.

6. Primary client generates the client-specific secrets with the help of KDF by using the info parameter as the Client IP address. The primary client shares these keys with the corresponding secondary clients over TLS.

B.3. Commands used

Two commands are used between the primary client and the primary server to denote the setup and KEM phases. Along with this, we have a "req / resp" to indicate whether it’s a request or response.

Between primary and secondary entities, we have one command to denote the sharing of the secret keys.

Appendix C. Change Log

Note to RFC Editor: if this document does not obsolete an existing RFC, please remove this appendix before publication as an RFC.

Appendix D. Open Issues

Note to RFC Editor: please remove this appendix before publication as an RFC.

Authors’ Addresses

Nalini Elkins
Inside Products, Inc.
36A Upper Circle
Carmel Valley, CA, 93924
United States of America
Phone: +1 831 234 4232
Email: nalini.elkins@insidethestack.com
Michael Ackermann
BCBS Michigan
P.O. Box 2888
Detroit, Michigan, 48231
United States of America
Phone: +1 248 703 3600
Email: mackermann@bcbsm.com
URI: http://www.bcbsm.com

Ameya Deshpande
NITK Surathkal
Pashan-Baner Link Road, Pashan
Pune, Maharashtra, 411021
India
Phone: +91 96893 26060
Email: ameyanrd@gmail.com
URI: https://www.nitk.ac.in/

Tommaso Pecorella
University of Florence
Dept. of Information Engineering, Via di Santa Marta, 3, 50139
Firenze
Italy
Phone: +39 055 2758540
Email: tommaso.pecorella@unifi.it
URI: https://www.unifi.it/

Adnan Rashid
University of Florence
Dept. of Information Engineering, Via di Santa Marta, 3, 50139
Firenze
Italy
Phone: +39 347 9821 467
Email: adnan.rashid@unifi.it
URI: https://www.unifi.it/
Explicit Flow Measurements Techniques
draft-ietf-ippm-explicit-flow-measurements-01

Abstract

This document describes protocol independent methods called Explicit Flow Measurement Techniques that employ few marking bits, inside the header of each packet, for loss and delay measurement. The endpoints, marking the traffic, signal these metrics to intermediate observers allowing them to measure connection performance, and to locate the network segment where impairments happen. Different alternatives are considered within this document. These signaling methods apply to all protocols but they are especially valuable when applied to protocols that encrypt transport header and do not allow traditional methods for delay and loss detection.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.
Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 6, 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction .................................................. 3
2. Latency Bits .................................................. 5
  2.1. Spin Bit .................................................. 5
  2.2. Delay Bit .................................................. 6
    2.2.1. Generation Phase ..................................... 8
    2.2.2. Reflection Phase ..................................... 8
    2.2.3. T_Max Selection ..................................... 9
    2.2.4. Delay Measurement Using Delay Bit ................... 10
      2.2.4.1. RTT Measurement ................................ 10
      2.2.4.2. Half-RTT Measurement ............................ 10
      2.2.4.3. Intra-Domain RTT Measurement .................... 11
    2.2.5. Observer’s Algorithm ................................ 12
    2.2.6. Two Bits Delay Measurement: Spin Bit + Delay Bit ... 13
    2.2.7. Hidden Delay Bit – Delay Bit with Privacy Protection 13
3. Loss Bits ....................................................... 13
  3.1. T Bit – Round Trip Loss Bit ............................... 14
    3.1.1. Round Trip Loss ...................................... 15
    3.1.2. Setting the Round Trip Loss Bit on Outgoing Packets . 16
    3.1.3. Observer’s Logic for Round Trip Loss Signal ........ 18
    3.1.4. Loss Coverage and Signal Timing ..................... 18
  3.2. Q Bit – sQuare Bit ....................................... 19
    3.2.1. Q Block Length Selection ............................ 19
    3.2.2. Upstream Loss ........................................ 20
    3.2.3. Identifying Q Block Boundaries ....................... 20
Packet loss and delay are hard and pervasive problems of day-to-day network operation. Proactively detecting, measuring, and locating them is crucial to maintaining high QoS and timely resolution of crippling end-to-end throughput issues. To this effect, in a TCP-dominated world, network operators have been heavily relying on information present in the clear in TCP headers: sequence and acknowledgment numbers and SACKs when enabled (see [RFC8517]). These allow for quantitative estimation of packet loss and delay by passive
on-path observation. Additionally, the problem can be quickly identified in the network path by moving the passive observer around.

With encrypted protocols, the equivalent transport headers are encrypted and passive packet loss and delay observations are not possible, as described in [RFC9065].

Measuring TCP loss and delay between similar endpoints cannot be relied upon to evaluate encrypted protocol loss and delay. Different protocols could be routed by the network differently, and the fraction of Internet traffic delivered using protocols other than TCP is increasing every year. It is imperative to measure packet loss and delay experienced by encrypted protocol users directly.

This document defines Explicit Flow Measurement Techniques. These hybrid measurement path signals (see [IPM-Methods]) are to be embedded into a transport layer protocol and are explicitly intended for exposing RTT and loss rate information to on-path measurement devices. They are designed to facilitate network operations and management and are "beneficial" for maintaining the quality of service (see [RFC9065]). These measurement mechanisms are applicable to any transport-layer protocol, and, as an example, the document describes QUIC and TCP bindings.

The Explicit Flow Measurement Techniques described in this document can be used alone or in combination with other Explicit Flow Measurement Techniques. Each technique uses a small number of bits and exposes a specific measurement.

Following the recommendation in [RFC8558] of making path signals explicit, this document proposes adding a small number of dedicated measurement bits to the clear portion of the protocol headers. These bits can be added to an unencrypted portion of a header belonging to any protocol layer, e.g. IP (see [IP]) and IPv6 (see [IPv6]) headers or extensions, such as [IPv6AltMark], UDP surplus space (see [UDP-OPTIONS] and [UDP-SURPLUS]), reserved bits in a QUIC v1 header, as already done with the latency Spin bit (see [QUIC-TRANSPORT]).

The measurements are not designed for use in automated control of the network in environments where signal bits are set by untrusted hosts. Instead, the signal is to be used for troubleshooting individual flows as well as for monitoring the network by aggregating information from multiple flows and raising operator alarms if aggregate statistics indicate a potential problem.

The Spin bit, Delay bit and loss bits explained in this document are inspired by [AltMark], [SPIN-BIT], [I-D.trammell-tpwg-spin] and [I-D.trammell-ippm-spin].
Additional details about the Performance Measurements for QUIC are described in the paper [ANRW19-PM-QUIC].

2. Latency Bits

This section introduces bits that can be used for round trip latency measurements. Whenever this section of the specification refers to packets, it is referring only to packets with protocol headers that include the latency bits.

[QUIC-TRANSPORT] introduces an explicit per-flow transport-layer signal for hybrid measurement of RTT. This signal consists of a Spin bit that toggles once per RTT. [SPIN-BIT] discusses an additional two-bit Valid Edge Counter (VEC) to compensate for loss and reordering of the Spin bit and increase fidelity of the signal in less than ideal network conditions.

This document introduces a stand-alone single-bit delay signal that can be used by passive observers to measure the RTT of a network flow, avoiding the Spin bit ambiguities that arise as soon as network conditions deteriorate.

2.1. Spin Bit

This section is a small recap of the Spin bit working mechanism. For a comprehensive explanation of the algorithm, please see [SPIN-BIT].

The Spin bit is an alternate marking [AltMark] generated signal, where the size of the alternation changes with the flight size each RTT.

The latency Spin bit is a single bit signal that toggles once per RTT, enabling latency monitoring of a connection-oriented communication from intermediate observation points.

A "spin period" is a set of packets with the same Spin bit value sent during one RTT time interval. A "spin period value" is the value of the Spin bit shared by all packets in a spin period.

The client and server maintain an internal per-connection spin value (i.e. 0 or 1) used to set the Spin bit on outgoing packets. Both endpoints initialize the spin value to 0 when a new connection starts. Then:

- when the client receives a packet with the packet number larger than any number seen so far, it sets the connection spin value to the opposite value contained in the received packet;
- when the server receives a packet with the packet number larger than any number seen so far, it sets the connection spin value to the same value contained in the received packet.

The computed spin value is used by the endpoints for setting the spin bit on outgoing packets. This mechanism allows the endpoints to generate a square wave such that, by measuring the distance in time between pairs of consecutive edges observed in the same direction, a passive on-path observer can compute the round trip delay of that network flow.

Spin bit enables round trip latency measurement by observing a single direction of the traffic flow.

Note that packet reordering can cause spurious edges that require heuristics to correct. The Spin bit performance deteriorates as soon as network impairments arise as explained in Section 2.2.

### 2.2. Delay Bit

The Delay bit has been designed to overcome accuracy limitations experienced by the Spin bit under difficult network conditions:

- packet reordering leads to generation of spurious edges and errors in delay estimation;

- loss of edges causes wrong estimation of spin periods and therefore wrong RTT measurements;

- application-limited senders cause the Spin bit to measure the application delays instead of network delays.

Unlike the Spin bit, which is set in every packet transmitted on the network, the Delay bit is set only once per round trip.

When the Delay bit is used, a single packet with a marked bit (the Delay bit) bounces between a client and a server during the entire connection lifetime. This single packet is called "delay sample".

An observer placed at an intermediate point, observing a single direction of traffic, tracking the delay sample and the relative timestamp, can measure the round trip delay of the connection.

The delay sample lifetime is comprised of two phases: initialization and reflection. The initialization is the generation of the delay sample, while the reflection realizes the bounce behavior of this single packet between the two endpoints.
The next figure describes the elementary Delay bit mechanism.

(a) No traffic at beginning.

(b) The Client starts sending data and sets the first packet as Delay Sample.

(c) The Server starts sending data and reflects the Delay Sample.

(d) The Client reflects the Delay Sample.

(e) The Server reflects the Delay Sample and so on.

Delay bit mechanism
2.2.1. Generation Phase

Only client is actively involved in the generation phase. It maintains an internal per-flow timestamp variable ("ds_time") updated every time a delay sample is transmitted.

When connection starts, the client generates a new delay sample initializing the Delay bit of the first outgoing packet to 1. Then it updates the "ds_time" variable with the timestamp of its transmission.

The server initializes the Delay bit to 0 at the beginning of the connection, and its only task during the connection is described in Section 2.2.2.

In absence of network impairments, the delay sample should bounce between client and server continuously, for the entire duration of the connection. That is highly unlikely for two reasons:

1. the packet carrying the Delay bit might be lost;
2. an endpoint could stop or delay sending packets because the application is limiting the amount of traffic transmitted.

To deal with these problems, the client generates a new delay sample if more than a predetermined time ("T_Max") has elapsed since the last delay sample transmission (including reflections). Note that "T_Max" should be greater than the max measurable RTT on the network. See Section 2.2.3 for details.

2.2.2. Reflection Phase

Reflection is the process that enables the bouncing of the delay sample between a client and a server. The behavior of the two endpoints is almost the same.

- Server side reflection: when a delay sample arrives, the server marks the first packet in the opposite direction as the delay sample.

- Client side reflection: when a delay sample arrives, the client marks the first packet in the opposite direction as the delay sample. It also updates the "ds_time" variable when the outgoing delay sample is actually forwarded.

In both cases, if the outgoing delay sample is being transmitted with a delay greater than a predetermined threshold after the reception of
the incoming delay sample (1ms by default), the delay sample is not reflected, and the outgoing Delay bit is kept at 0.

By doing so, the algorithm can reject measurements that would overestimate the delay due to lack of traffic on the endpoints. Hence, the maximum estimation error would amount to twice the threshold (e.g. 2ms) per measurement.

2.2.3. T_Max Selection

The internal "ds_time" variable allows a client to identify delay sample losses. Considering that a lost delay sample is regenerated at the end of an explicit time ("T_Max") since the last generation, this same value can be used by an observer to reject a measure and start a new one.

In other words, if the difference in time between two delay samples is greater or equal than "T_Max", then these cannot be used to produce a delay measure. Therefore the value of "T_Max" must also be known to the on-path network probes.

There are two alternatives to select the "T_Max" value so that both client and observers know it. The first one requires that "T_Max" is known a priori ("T_Max_p") and therefore set within the protocol specifications that implements the marking mechanism (e.g. 1 second which usually is greater than the max expectable RTT). The second alternative requires a dynamic mechanism able to adapt the duration of the "T_Max" to the delay of the connection ("T_Max_c").

For instance, client and observers could use the connection RTT as a basis for calculating an effective "T_Max". They should use a predetermined initial value so that "T_Max = T_Max_p" (e.g. 1 second) and then, when a valid RTT is measured, change "T_Max" accordingly so that "T_Max = T_Max_c". In any case, the selected "T_Max" should be large enough to absorb any possible variations in the connection delay.

"T_Max_c" could be computed as two times the measured "RTT" plus a fixed amount of time ("100ms") to prevent low "T_Max" values in case of very small RTTs. The resulting formula is: "T(Max_c = 2RTT + 100ms". If "T(Max_c" is greater than "T(Max_p" then "T(Max_c" is forced to "T(Max_p" value.

Note that the observer’s "T_Max" should always be less than or equal to the client’s "T_Max" to avoid considering as a valid measurement what is actually the client’s "T_Max". To obtain this result, the client waits for two consecutive incoming samples and computes the two related RTTs. Then it takes the largest of them as the basis of
the "T_Max_c" formula. At this point, observers have already measured a valid RTT and then computed their "T_Max_c".

2.2.4. Delay Measurement Using Delay Bit

When the Delay bit is used, a passive observer can use delay samples directly and avoid inherent ambiguities in the calculation of the RTT as can be seen in Spin bit analysis.

2.2.4.1. RTT Measurement

The delay sample generation process ensures that only one packet marked with the Delay bit set to 1 runs back and forth between two endpoints per round trip time. To determine the RTT measurement of a flow, an on-path passive observer computes the time difference between two delay samples observed in a single direction.

To ensure a valid measurement, the observer must verify that the distance in time between the two samples taken into account is less than "T_Max".

```
= *********     -----Obs---->     ********* =
= * Client *                           * Server * =
= *********     <-------------     ********* =
<=================================================================
(a) client-server RTT
```

```
= *********     ------------>     ********* =
= * Client *                           * Server * =
= *********     <----Obs-----     ********* =
<=================================================================
(b) server-client RTT
```

2.2.4.2. Half-RTT Measurement

An observer that is able to observe both forward and return traffic directions can use the delay samples to measure "upstream" and "downstream" RTT components, also known as the half-RTT measurements. It does this by measuring the time between a delay sample observed in one direction and the delay sample previously observed in the opposite direction.
As with RTT measurement, the observer must verify that the distance in time between the two samples taken into account is less than "T_Max".

Note that upstream and downstream sections of paths between the endpoints and the observer, i.e. observer-to-client vs client-to-observer and observer-to-server vs server-to-observer, may have different delay characteristics due to the difference in network congestion and other factors.

(a) client-observer half-RTT

(b) observer-server half-RTT

Half Round-trip time (both direction)

2.2.4.3. Intra-Domain RTT Measurement

Intra-domain RTT is the portion of the entire RTT used by a flow to traverse the network of a provider. To measure intra-domain RTT, two observers capable of observing traffic in both directions must be employed simultaneously at ingress and egress of the network to be measured. Intra-domain RTT is difference between the two computed upstream (or downstream) RTT components.
2.2.5. Observer’s Algorithm

An on-path observer maintains an internal per-flow variable to keep track of time at which the last delay sample has been observed.

A unidirectional observer, upon detecting a delay sample:

- if a delay sample was also detected previously in the same direction and the distance in time between them is less than \( T_{\text{Max}} - K \), then the two delay samples can be used to calculate RTT measurement. \( K \) is a protection threshold to absorb differences in \( T_{\text{Max}} \) computation and delay variations between two consecutive delay samples (e.g. \( K = 10\% T_{\text{Max}} \)).

If the observer can observe both forward and return traffic flows, and it is able to determine which direction contains the client and the server (e.g. by observing the connection handshake), upon detecting a delay sample:

- if a delay sample was also detected in the opposite direction and the distance in time between them is less than \( T_{\text{Max}} - K \), then the two delay samples can be used to measure the observer-client half-RTT or the observer-server half-RTT, according to the direction of the last delay sample observed.
2.2.6. Two Bits Delay Measurement: Spin Bit + Delay Bit

Spin and Delay bit algorithms work independently. If both marking methods are used in the same connection, observers can choose the best measurement between the two available:

- when a precise measurement can be produced using the Delay bit, observers choose it;
- when a Delay bit measurement is not available, observers choose the approximate Spin bit one.

2.2.7. Hidden Delay Bit - Delay Bit with Privacy Protection

Theoretically, delay measurements can be used to roughly evaluate the distance of the client from the server (using the RTT) or from any intermediate observer (using the client-observer half-RTT). To protect users’ privacy, the Delay bit algorithm can be slightly modified to mask the RTT of the connection to an intermediate observer. This result can be achieved by, for example, delaying the client-side reflection of the delay sample by a fixed randomly chosen time value. This would lead an intermediate observer to measure a delay greater than the real one.

This Additional Delay should be randomly selected by the client and kept constant for a certain amount of time across multiple connections. This ensures that the client-server jitter remains the same as if no Additional Delay had been inserted. For example, a new Additional Delay value could be generated whenever the client’s IP address changes.

Using this technique, despite the Additional Delay introduced, it is still possible to correctly measure the right component of RTT (observer-server) and all the intra-domain measurements used to distribute the delay in the network. Furthermore, differently from the Delay bit, the hidden Delay bit makes the use of the client reflection threshold (1ms) redundant. Removing this threshold leads to the further advantage of increasing the number of valid measurements produced by the algorithm.

3. Loss Bits

This section introduces bits that can be used for loss measurements. Whenever this section of the specification refers to packets, it is referring only to packets with protocol headers that include the loss bits - the only packets whose loss can be measured.
- T: the "round Trip loss" bit is used in combination with the Spin bit to measure round-trip loss. See Section 3.1.
- Q: the "sQuare signal" bit is used to measure upstream loss. See Section 3.2.
- L: the "Loss event" bit is used to measure end-to-end loss. See Section 3.3.
- R: the "Reflection square signal" bit is used in combination with Q bit to measure end-to-end loss. See Section 3.1.

Loss measurements enabled by T, Q, and L bits can be implemented by those loss bits alone (T bit requires a working Spin bit). Two-bit combinations Q+L and Q+R enable additional measurement opportunities discussed below.

Each endpoint maintains appropriate counters independently and separately for each separately identifiable flow (each sub-flow for multipath connections).

Since loss is reported independently for each flow, all bits (except for L bit) require a certain minimum number of packets to be exchanged per flow before any signal can be measured. Therefore, loss measurements work best for flows that transfer more than a minimal amount of data.

3.1. T Bit - Round Trip Loss Bit

The round Trip loss bit is used to mark a variable number of packets exchanged twice between the endpoints realizing a two round-trip reflection. A passive on-path observer, observing either direction, can count and compare the number of marked packets seen during the two reflections, estimating the loss rate experienced by the connection. The overall exchange comprises:

- the client selects, generates and consequently transmits a first train of packets, by setting the T bit to 1;
- the server, upon receiving each packet included in the first train, reflects to the client a respective second train of packets of the same size as the first train received, by setting the T bit to 1;
- the client, upon receiving each packet included in the second train, reflects to the server a respective third train of packets of the same size as the second train received, by setting the T bit to 1;
the server, upon receiving each packet included in the third
train, finally reflects to the client a respective fourth train of
packets of the same size as the third train received, by setting
the T bit to 1.

Packets belonging to the first round trip (first and second train)
represent the Generation Phase, while those belonging to the second
round trip (third and fourth train) represent the Reflection Phase.

A passive on-path observer can count and compare the number of marked
packets seen during the two round trips (i.e. the first and third or
the second and the fourth trains of packets, depending on which
direction is observed) and estimate the loss rate experienced by the
connection. This process is repeated continuously to obtain more
measurements as long as the endpoints exchange traffic. These
measurements can be called Round Trip losses.

Since packet rates in two directions may be different, the number of marked
packets in the train is determined by the direction with the
lowest packet rate. See Section 3.1.2 for details on packet
generation.

3.1.1. Round Trip Loss

Since the measurements are performed on a portion of the traffic
exchanged between the client and the server, the observer calculates
the end-to-end Round Trip Packet Loss (RTPL) that, statistically,
will correspond to the loss rate experienced by the connection along
the entire network path.

```
+----------------------|---------------------->
|  *********  -----Obs----->  *********  |
|  * Client *                        * Server *  |
|  *********  <---------------  *********  |
<++++++++++++++++++++++++++++++++++++++>
```

(a) client-server RTPL

```
+----------------------|---------------------->
|  *********  <-----Obs----->  *********  |
|  * Client *                        * Server *  |
|  *********  <---------------  *********  |
<++++++++++++++++++++++++++++++++++++++>
```

(b) server-client RTPL

Round-trip packet loss (both direction)
This methodology also allows the Half-RTPL measurement and the Intra-domain RTPL measurement in a way similar to RTT measurement.

(a) client-observer half-RTPL

(b) observer-server half-RTPL

Half Round-trip packet loss (both direction)

(a) observer-server RTPL components (half-RTPLs)

(b) the intra-domain RTPL resulting from the subtraction of the above RTPL components

Intra-domain Round-trip packet loss (observer-server)

3.1.2. Setting the Round Trip Loss Bit on Outgoing Packets

The round Trip loss signal requires a working Spin-bit signal to separate trains of marked packets (packets with T bit set to 1). A "pause" of at least one empty spin-bit period between each phase of the algorithm serves as such separator for the on-path observer.
The client maintains a "generation token" count that is set to zero at the beginning of the session and is incremented every time a packet is received (marked or unmarked). The client also maintains a "reflection counter" that starts at zero at the beginning of the session.

The client is in charge of launching trains of marked packets and does so according to the algorithm:

1. Generation Phase. The client starts generating marked packets for two consecutive spin-bit periods; when the client transmits a packet and a "generation token" is available, the client marks the packet and retires a "generation token". If no token is available, the outgoing packet is transmitted unmarked. At the end of the first spin-bit period spent in generation, the reflection counter is unlocked to start counting incoming marked packets that will be reflected later;

2. Pause Phase. When the generation is completed, the client pauses till it has observed one entire Spin bit period with no marked packets. That Spin bit period is used by the observer as a separator between generated and reflected packets. During this marking pause, all the outgoing packets are transmitted with T bit set to 0. The reflection counter is still incremented every time a marked packet arrives;

3. Reflection Phase. The client starts transmitting marked packets, decrementing the reflection counter for each transmitted marked packet until the reflection counter reached zero. The "generation token" method from the generation phase is used during this phase as well. At the end of the first spin-period spent in reflection, the reflection counter is locked to avoid incoming reflected packets incrementing it;

4. Pause Phase 2. The pause phase is repeated after the reflection phase and serves as a separator between the reflected packet train and a new packet train.

The generation token counter should be capped to limit the effects of a subsequent sudden reduction in the other endpoint’s packet rate that could prevent that endpoint from reflecting collected packets. The most conservative cap value is "1".

A server maintains a "marking counter" that starts at zero and is incremented every time a marked packet arrives. When the server transmits a packet and the "marking counter" is positive, the server marks the packet and decrements the "marking counter". If the
"marking counter" is zero, the outgoing packet is transmitted unmarked.

Note that a choice of 2-RTT (two spin periods) for the generation phase is a tradeoff between the percentage of marked packets (i.e. the percentage of traffic monitored) and the measurement delay. Using this value the algorithm produces a measurement approximately every 6-RTT ("2" generation, "2" reflection, "2" pauses), marking "1/3" of packets exchanged in the slower direction (see Section 3.1.4). Choosing a generation phase of 1-RTT, we would produce measurements every 4-RTT, monitoring just "1/4" of packets in the slower direction.

3.1.3. Observer’s Logic for Round Trip Loss Signal

The on-path observer counts marked packets and separates different trains by detecting spin-bit periods (at least one) with no marked packets. The Round Trip Packet Loss (RTPL) is the difference between the size of the Generation train and the Reflection train.

In the following example, packets are represented by two bits (first one is the Spin bit, second one is the round Trip loss bit):

```
<table>
<thead>
<tr>
<th>Generation</th>
<th>Pause</th>
<th>Reflection</th>
<th>Pause</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 01 00 01 11 10 11 00 00 10 10 10 01 00 01 01 10 11 10 00 00 10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Round Trip Loss signal example

Note that 5 marked packets have been generated of which 4 have been reflected.

3.1.4. Loss Coverage and Signal Timing

A cycle of the round Trip loss signaling algorithm contains 2 RTTs of Generation phase, 2 RTTs of Reflection phase, and two Pause phases at least 1 RTT in duration each. Hence, the loss signal is delayed by about 6 RTTs since the loss events.

The observer can only detect loss of marked packets that occurs after its initial observation of the Generation phase and before its subsequent observation of the Reflection phase. Hence, if the loss occurs on the path that sends packets at a lower rate (typically ACKs in such asymmetric scenarios), "2/6" ("1/3") of the packets will be sampled for loss detection.
If the loss occurs on the path that sends packets at a higher rate, "lowPacketRate/(3*highPacketRate)" of the packets will be sampled for loss detection. For protocols that use ACKs, the portion of packets sampled for loss in the higher rate direction during unidirectional data transfer is "1/(3*packetsPerAck)", where the value of "packetsPerAck" can vary by protocol, by implementation, and by network conditions.

3.2. Q Bit - sQuare Bit

The sQuare bit (Q bit) takes its name from the square wave generated by its signal. Every outgoing packet contains the Q bit value, which is initialized to the 0 and inverted after sending N packets (a sQuare Block or simply Q Block). Hence, Q Period is 2*N. The Q bit represents "packet color" as defined by [AltMark].

Observation points can estimate upstream losses by watching a single direction of the traffic flow and counting the number of packets in each observed Q Block, as described in Section 3.2.2.

3.2.1. Q Block Length Selection

The length of the block must be known to the on-path network probes. There are two alternatives to selecting the Q Block length. The first one requires that the length is known a priori and therefore set within the protocol specifications that implements the marking mechanism. The second requires the sender to select it.

In this latter scenario, the sender is expected to choose N (Q Block length) based on the expected amount of loss and reordering on the path. The choice of N strikes a compromise—the observation could become too unreliable in case of packet reordering and/or severe loss if N is too small, while short flows may not yield a useful upstream loss measurement if N is too large (see Section 3.2.2).

The value of N should be at least 64 and be a power of 2. This requirement allows an Observer to infer the Q Block length by observing one period of the square signal. It also allows the Observer to identify flows that set the loss bits to arbitrary values (see Section 5).

If the sender does not have sufficient information to make an informed decision about Q Block length, the sender should use N=64, since this value has been extensively tried in large-scale field tests and yielded good results. Alternatively, the sender may also choose a random power-of-2 N for each flow, increasing the chances of using a Q Block length that gives the best signal for some flows.
The sender must keep the value of N constant for a given flow.

3.2.2. Upstream Loss

Blocks of N (Q Block length) consecutive packets are sent with the same value of the Q bit, followed by another block of N packets with an inverted value of the Q bit. Hence, knowing the value of N, an on-path observer can estimate the amount of upstream loss after observing at least N packets. The upstream loss rate ("uloss") is one minus the average number of packets in a block of packets with the same Q value ("p") divided by N ("uloss=1-avg(p)/N").

The observer needs to be able to tolerate packet reordering that can blur the edges of the square signal, as explained in Section 3.2.3.

(a) in client-server channel (uloss_up)

(b) in server-client channel (uloss_down)

3.2.3. Identifying Q Block Boundaries

Packet reordering can produce spurious edges in the square signal. To address this, the observer should look for packets with the current Q bit value up to X packets past the first packet with a reverse Q bit value. The value of X, a "Marking Block Threshold", should be less than "N/2".

The choice of X represents a trade-off between resiliency to reordering and resiliency to loss. A very large Marking Block Threshold will be able to reconstruct Q Blocks despite a significant amount of reordering, but it may erroneously coalesce packets from multiple Q Blocks into fewer Q Blocks, if loss exceeds 50% for some Q Blocks.
3.2.3.1. Improved Resilience to Burst Losses

Burst losses can affect Q measurements accuracy. Generally, burst losses can be absorbed and correctly measured if smaller than the established Q Block length. If entire Q Block length of packets get lost in a burst, however, the observer may be left completely unaware of the loss.

To improve burst loss resilience, an observer may consider a received Q Block larger than the selected Q Block length as an indication of a burst loss event. The observer would then compute the loss as three times Q Block length minus the measured block length. By doing so, the observer can detect burst losses of less than two blocks (e.g., less than 128 packets for Q Block length of 64 packets). A burst loss of two or more consecutive periods would still remain unnoticed by the observer (or underestimated if a period longer than Q Block length were formed).

3.3. L Bit - Loss Event Bit

The Loss Event bit uses an Unreported Loss counter maintained by the protocol that implements the marking mechanism. To use the Loss Event bit, the protocol must allow the sender to identify lost packets. This is true of protocols such as QUIC, partially true for TCP and SCTP (losses of pure ACKs are not detected) and is not true of protocols such as UDP and IP/IPv6.

The Unreported Loss counter is initialized to 0, and L bit of every outgoing packet indicates whether the Unreported Loss counter is positive (L=1 if the counter is positive, and L=0 otherwise).

The value of the Unreported Loss counter is decremented every time a packet with L=1 is sent.

The value of the Unreported Loss counter is incremented for every packet that the protocol declares lost, using whatever loss detection machinery the protocol employs. If the protocol is able to rescind the loss determination later, a positive Unreported Loss counter may be decremented due to the rescission, but it should not become negative due to the rescission.

This loss signaling is similar to loss signaling in [ConEx], except the Loss Event bit is reporting the exact number of lost packets, whereas Echo Loss bit in [ConEx] is reporting an approximate number of lost bytes.

For protocols, such as TCP ([TCP]), that allow network devices to change data segmentation, it is possible that only a part of the
packet is lost. In these cases, the sender must increment Unreported Loss counter by the fraction of the packet data lost (so Unreported Loss counter may become negative when a packet with L=1 is sent after a partial packet has been lost).

Observation points can estimate the end-to-end loss, as determined by the upstream endpoint, by counting packets in this direction with the L bit equal to 1, as described in Section 3.3.1.

3.3.1. End-To-End Loss

The Loss Event bit allows an observer to estimate the end-to-end loss rate by counting packets with L bit value of 0 and 1 for a given flow. The end-to-end loss rate is the fraction of packets with L=1.

The assumption here is that upstream loss affects packets with L=0 and L=1 equally. If some loss is caused by tail-drop in a network device, this may be a simplification. If the sender’s congestion controller reduces the packet send rate after loss, there may be a sufficient delay before sending packets with L=1 that they have a greater chance of arriving at the observer.

3.3.1.1. Loss Profile Characterization

The Loss Event bit allows an observer to characterize loss profile, since the distribution of observed packets with L bit set to 1 roughly corresponds to the distribution of packets lost between 1 RTT and 1 RTO before (see Section 3.3.2.1). Hence, observing random single instances of L bit set to 1 indicates random single packet loss, while observing blocks of packets with L bit set to 1 indicates loss affecting entire blocks of packets.

3.3.2. L+Q Bits - Loss Measurement Using L and Q Bits

Combining L and Q bits allows a passive observer watching a single direction of traffic to accurately measure:

- upstream loss: sender-to-observer loss (see Section 3.2.2)
- downstream loss: observer-to-receiver loss (see Section 3.3.2.2)
- end-to-end loss: sender-to-receiver loss on the observed path (see Section 3.3.1) with loss profile characterization (see Section 3.3.1.1)
3.3.2.1. Correlating End-to-End and Upstream Loss

Upstream loss is calculated by observing packets that did not suffer the upstream loss (Section 3.2.2). End-to-end loss, however, is calculated by observing subsequent packets after the sender’s protocol detected the loss. Hence, end-to-end loss is generally observed with a delay of between 1 RTT (loss declared due to multiple duplicate acknowledgments) and 1 RTO (loss declared due to a timeout) relative to the upstream loss.

The flow RTT can sometimes be estimated by timing protocol handshake messages. This RTT estimate can be greatly improved by observing a dedicated protocol mechanism for conveying RTT information, such as the Spin bit (see Section 2.1) or Delay bit (see Section 2.2).

Whenever the observer needs to perform a computation that uses both upstream and end-to-end loss rate measurements, it should use upstream loss rate leading the end-to-end loss rate by approximately 1 RTT. If the observer is unable to estimate RTT of the flow, it should accumulate loss measurements over time periods of at least 4 times the typical RTT for the observed flows.

If the calculated upstream loss rate exceeds the end-to-end loss rate calculated in Section 3.3.1, then either the Q Period is too short for the amount of packet reordering or there is observer loss, described in Section 3.3.2.3. If this happens, the observer should adjust the calculated upstream loss rate to match end-to-end loss rate, unless the following applies.

In case of a protocol, such as TCP or SCTP, that does not track losses of pure ACK packets, observing a direction of traffic dominated by pure ACK packets could result in measured upstream loss that is higher than measured end-to-end loss, if said pure ACK packets are lost upstream. Hence, if the measurement is applied to such protocols, and the observer can confirm that pure ACK packets dominate the observed traffic direction, the observer should adjust the calculated end-to-end loss rate to match upstream loss rate.

3.3.2.2. Downstream Loss

Because downstream loss affects only those packets that did not suffer upstream loss, the end-to-end loss rate ("eloss") relates to the upstream loss rate ("uloss") and downstream loss rate ("dloss") as 

\[(1-\text{uloss})(1-\text{dloss})=1-\text{eloss}.\]

Hence, 

\[\text{dloss}=(\text{eloss}-\text{uloss})/(1-\text{uloss}).\]
3.3.2.3. Observer Loss

A typical deployment of a passive observation system includes a network tap device that mirrors network packets of interest to a device that performs analysis and measurement on the mirrored packets. The observer loss is the loss that occurs on the mirror path.

Observer loss affects upstream loss rate measurement, since it causes the observer to account for fewer packets in a block of identical Q bit values (see Section 3.2.2). The end-to-end loss rate measurement, however, is unaffected by the observer loss, since it is a measurement of the fraction of packets with the L bit value of 1, and the observer loss would affect all packets equally (see Section 3.3.1).

The need to adjust the upstream loss rate down to match end-to-end loss rate as described in Section 3.3.2.1 is an indication of the observer loss, whose magnitude is between the amount of such adjustment and the entirety of the upstream loss measured in Section 3.2.2. Alternatively, a high apparent upstream loss rate could be an indication of significant packet reordering, possibly due to packets belonging to a single flow being multiplexed over several upstream paths with different latency characteristics.

3.4. R Bit - Reflection Square Bit

R bit requires a deployment alongside Q bit. Unlike the square signal for which packets are transmitted in blocks of fixed size, the number of packets in Reflection square signal blocks (also an alternate marking signal) varies according to these rules:

- when the transmission of a new block starts, its size is set equal to the size of the last Q Block whose reception has been completed;

- if, before transmission of the block is terminated, the reception of at least one further Q Block is completed, the size of the block is updated to be the average size of the further received Q Blocks.

The Reflection square value is initialized to 0 and is applied to the R bit of every outgoing packet. The Reflection square value is toggled for the first time when the completion of a Q Block is detected in the incoming square signal (produced by the other endpoint using the Q bit). The number of packets detected within this first Q Block ("p"), is used to generate a reflection square signal that toggles every "M=p" packets (at first). This new signal
produces blocks of \(M\) packets (marked using the R bit) and each of them is called "Reflection Block" (R Block).

The \(M\) value is then updated every time a completed Q Block in the incoming square signal is received, following this formula: \(M=\text{round}(avg(p))\).

The parameter "avg(p)", the average number of packets in a marking period, is computed based on all the Q Blocks received since the beginning of the current R Block.

The transmission of an R Block is considered complete (and the signal toggled) when the number of packets transmitted in that block is at least the latest computed \(M\) value.

To ensure a proper computation of the \(M\) value, endpoints implementing the R bit must identify the boundaries of incoming Q Blocks. The same approach described in Section 3.2.3 should be used.

Looking at the R bit, unidirectional observation points have an indication of loss experienced by the entire unobserved channel plus the loss on the path from the sender to them.

Since the Q Block is sent in one direction, and the corresponding reflected R Block is sent in the opposite direction, the reflected R signal is transmitted with the packet rate of the slowest direction. Namely, if the observed direction is the slowest, there can be multiple Q Blocks transmitted in the unobserved direction before a complete R Block is transmitted in the observed direction. If the unobserved direction is the slowest, the observed direction can be sending R Blocks of the same size repeatedly before it can update the signal to account for a newly-completed Q Block.

3.4.1. Enhancement of R Block Length Computation

The use of the rounding function used in the \(M\) computation introduces errors that can be minimized by storing the rounding applied each time \(M\) is computed, and using it during the computation of the \(M\) value in the following R Block.

This can be achieved introducing the new "\(r_{avg}\)" parameter in the computation of \(M\). The new formula is \(M_r=\text{avg}(p)+r_{avg}; M=\text{round}(M_r); r_{avg}=M_r-M\) where the initial value of "\(r_{avg}\)" is equal to 0.
3.4.2. Improved Resilience to Packet Reordering

When a protocol implementing the marking mechanism is able to detect when packets are received out of order, it can improve resilience to packet reordering beyond what is possible using methods described in Section 3.2.3.

This can be achieved by updating the size of the current R Block while it is being transmitted. The reflection block size is then updated every time an incoming reordered packet of the previous Q Block is detected. This can be done if and only if the transmission of the current reflection block is in progress and no packets of the following Q Block have been received.

3.4.2.1. Improved Resilience to Burst Losses

Burst losses can affect R measurements accuracy similarly to how they affect Q measurements accuracy. Therefore, recommendations in section Section 3.2.3.1 apply equally to improving burst loss resilience for R measurements.

3.4.3. R+Q Bits - Loss Measurement Using R and Q Bits

Since both sQuare and Reflection square bits are toggled at most every N packets (except for the first transition of the R bit as explained before), an on-path observer can count the number of packets of each marking block and, knowing the value of N, can estimate the amount of loss experienced by the connection. An observer can calculate different measurements depending on whether it is able to observe a single direction of the traffic or both directions.

Single directional observer:

- upstream loss in the observed direction: the loss between the sender and the observation point (see Section 3.2.2)
- "three-quarters" connection loss: the loss between the receiver and the sender in the unobserved direction plus the loss between the sender and the observation point in the observed direction
- end-to-end loss in the unobserved direction: the loss between the receiver and the sender in the opposite direction

Two directions observer (same metrics seen previously applied to both direction, plus):
- client-observer half round-trip loss: the loss between the client and the observation point in both directions
- observer-server half round-trip loss: the loss between the observation point and the server in both directions
- downstream loss: the loss between the observation point and the receiver (applicable to both directions)

3.4.3.1. Three-Quarters Connection Loss

Except for the very first block in which there is nothing to reflect (a complete Q Block has not been yet received), packets are continuously R-bit marked into alternate blocks of size lower or equal than N. Knowing the value of N, an on-path observer can estimate the amount of loss occurred in the whole opposite channel plus the loss from the sender up to it in the observation channel. As for the previous metric, the "three-quarters" connection loss rate ("tqloss") is one minus the average number of packets in a block of packets with the same R value ("t") divided by "N" ("tqloss=1-avg(t)/N").

(a) in client-server channel (tqloss_up)

(b) in server-client channel (tqloss_down)

Three-quarters connection loss

The following metrics derive from this last metric and the upstream loss produced by the Q bit.

3.4.3.2. End-To-End Loss in the Opposite Direction

End-to-end loss in the unobserved direction ("eloss_unobserved") relates to the "three-quarters" connection loss ("tqloss") and upstream loss in the observed direction ("uloss") as
"(1-eloss_unobserved)(1-uloss)=1-tqloss". Hence, "eloss_unobserved=(tqloss-uloss)/(1-uloss)".

**********     -----Obs---->     **********
* Client *                       * Server *
**********     <---------------     **********
<=============================================================================

(a) in client-server channel (eloss_down)

============================================================================>

**********     -------------------->     **********
* Client *                       * Server *
**********     <----Obs-----     **********

(b) in server-client channel (eloss_up)

End-To-End loss in the opposite direction

3.4.3.3. Half Round-Trip Loss

If the observer is able to observe both directions of traffic, it is able to calculate two "half round-trip" loss measurements - loss from the observer to the receiver (in a given direction) and then back to the observer in the opposite direction. For both directions, "half round-trip" loss ("hrtloss") relates to "three-quarters" connection loss ("tqloss_opposite") measured in the opposite direction and the upstream loss ("uloss") measured in the given direction as "(1-uloss)(1-hrtloss)=1-tqloss_opposite". Hence, "hrtloss=(tqloss_opposite-uloss)/(1-uloss)".

============================================================================>
= **********     -----|----->     **********
= * Client *          Obs          * Server *
= **********     <-----|------     **********
<=============================================================================

(a) client-observer half round-trip loss (hrtloss_co)

============================================================================>

**********     -----|----->     ********** =
* Client *          Obs          * Server * =
**********     <-----|------     *******
<=============================================================================

(b) observer-server half round-trip loss (hrtloss_os)

Half Round-trip loss (both direction)
3.4.3.4. Downstream Loss

If the observer is able to observe both directions of traffic, it is able to calculate two downstream loss measurements using either end-to-end loss and upstream loss, similar to the calculation in Section 3.3.2.2 or using "half round-trip" loss and upstream loss in the opposite direction.

For the latter, "dloss=(hrtloss-uloss_opposite)/(1-uloss_opposite)".

**-------------------->
* Client * ------|-----> * Server *
* Obs *<--------|--------> * Server *
****************************
(a) in client-server channel (dloss_up)

**-------------------->
* Client * ------|-----> * Server *
* Obs *<--------|--------> * Server *
****************************
(b) in server-client channel (dloss_down)

Downstream loss

3.5. E Bit - ECN-Echo Event Bit

While the primary focus of this draft is on exposing packet loss and delay, modern networks can report congestion before they are forced to drop packets, as described in [ECN]. When transport protocols keep ECN-Echo feedback under encryption, this signal cannot be observed by the network operators. When tasked with diagnosing network performance problems, knowledge of a congestion downstream of an observation point can be instrumental.

If downstream congestion information is desired, this information can be signaled with an additional bit.

- E: The "ECN-Echo Event" bit is set to 0 or 1 according to the Unreported ECN Echo counter, as explained below in Section 3.5.1.

3.5.1. Setting the ECN-Echo Event Bit on Outgoing Packets

The Unreported ECN-Echo counter operates identically to Unreported Loss counter (Section 3.3), except it counts packets delivered by the network with CE markings, according to the ECN-Echo feedback from the receiver.
This ECN-Echo signaling is similar to ECN signaling in [ConEx]. ECN-Echo mechanism in QUIC provides the number of packets received with CE marks. For protocols like TCP, the method described in [ConEx-TCP] can be employed. As stated in [ConEx-TCP], such feedback can be further improved using a method described in [ACCURATE].

3.5.2. Using E Bit for Passive ECN-Reported Congestion Measurement

A network observer can count packets with CE codepoint and determine the upstream CE-marking rate directly.

Observation points can also estimate ECN-reported end-to-end congestion by counting packets in this direction with a E bit equal to 1.

The upstream CE-marking rate and end-to-end ECN-reported congestion can provide information about downstream CE-marking rate. Presence of E bits along with L bits, however, can somewhat confound precise estimates of upstream and downstream CE-markings in case the flow contains packets that are not ECN-capable.

4. Summary of Delay and Loss Marking Methods

This section summarizes the marking methods described in this draft.

For the Delay measurement, it is possible to use the Spin bit and/or the delay bit. A unidirectional or bidirectional observer can be used.
<table>
<thead>
<tr>
<th>Method</th>
<th># of bits</th>
<th>Available Delay Metrics</th>
<th>Impairments Resiliency</th>
<th># of meas.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>UniDir Observer</td>
<td>BiDir Observer</td>
<td></td>
</tr>
<tr>
<td>S: Spin Bit</td>
<td>1</td>
<td>RTT</td>
<td>x2</td>
<td>low</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Half RTT</td>
<td></td>
</tr>
<tr>
<td>D: Delay Bit</td>
<td>1</td>
<td>RTT</td>
<td>x2</td>
<td>high</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Half RTT</td>
<td></td>
</tr>
<tr>
<td>D^: Hidden Delay Bit</td>
<td>1</td>
<td>RTT^</td>
<td>x2</td>
<td>high</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Left Half^</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Right Half</td>
<td></td>
</tr>
<tr>
<td>SD: Spin Bit &amp; Delay Bit*</td>
<td>2</td>
<td>RTT</td>
<td>x2</td>
<td>high</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Half RTT</td>
<td></td>
</tr>
</tbody>
</table>

x2 Same metric for both directions  
* Both bits work independently; an observer could use less accurate Spin bit measurements when Delay bit ones are unavailable  
^ Masked metric (real value can be calculated only by those who know the Additional Delay)

**Figure 1: Delay Comparison**

For the Loss measurement, each row in the table of Figure 2 represents a loss marking method. For each method the table specifies the number of bits required in the header, the available metrics using an unidirectional or bidirectional observer, applicable protocols, measurement fidelity and delay.
<table>
<thead>
<tr>
<th>Method</th>
<th>Bit</th>
<th>Available Loss Metrics</th>
<th>P</th>
<th>Measurement Aspects</th>
</tr>
</thead>
<tbody>
<tr>
<td>T: Round Trip</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loss Bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>UniDir Observer</td>
<td></td>
<td>Rate by sampling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BiDir Observer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>~6 RTT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1/3 to 1/(3*ppa) of</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>pkts over 2 RTT</td>
</tr>
<tr>
<td>Q: sQuare Bit</td>
<td>1</td>
<td>Upstream</td>
<td>2</td>
<td>Rate over N pkts</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(e.g. 64)</td>
</tr>
<tr>
<td>L: Loss Event</td>
<td>1</td>
<td>E2E</td>
<td>x2</td>
<td>Loss shape (and rate)</td>
</tr>
<tr>
<td>Bit</td>
<td></td>
<td></td>
<td></td>
<td>Min: RTT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Max: RTO</td>
</tr>
<tr>
<td>QL: sQuare +</td>
<td>2</td>
<td>Upstream</td>
<td>x2</td>
<td>-&gt; see Q</td>
</tr>
<tr>
<td>Loss Ev. Bits</td>
<td></td>
<td>Downstream</td>
<td>x2</td>
<td>-&gt; see Q</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Others:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E2E</td>
<td>x2</td>
<td>-&gt; see L</td>
</tr>
<tr>
<td></td>
<td></td>
<td>!E2E</td>
<td>x2</td>
<td>Others:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Downstream</td>
<td></td>
<td>-&gt; see L</td>
</tr>
<tr>
<td>QR: sQuare +</td>
<td>2</td>
<td>Upstream</td>
<td>x2</td>
<td>Rate over N*ppa pkts</td>
</tr>
<tr>
<td>Ref. Sq. Bits</td>
<td></td>
<td>3/4 RT</td>
<td></td>
<td>(see Q bit for N)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>!E2E</td>
<td>x2</td>
<td>Others:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Downstream</td>
<td>E2E</td>
<td>N*ppa pk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Half RT</td>
<td></td>
<td>(see Q for N)</td>
</tr>
</tbody>
</table>

* All protocols
# Protocols employing loss detection
   (with or without pure ACK loss detection)
$ Require a working Spin bit
! Metric relative to the opposite channel
x2 Same metric for both directions
ppa Packets-Per-Ack
Q|L See Q if Upstream loss is significant; L otherwise

Figure 2: Loss Comparison

5. Protocol Ossification Considerations

Accurate loss and delay information is not critical to the operation of any protocol, though its presence for a sufficient number of flows is important for the operation of networks.

The delay and loss bits are amenable to "greasing" described in [RFC8701], if the protocol designers are not ready to dedicate (and
ossify) bits used for loss reporting to this function. The greasing could be accomplished similarly to the Latency Spin bit greasing in [QUIC-TRANSPORT]. Namely, implementations could decide that a fraction of flows should not encode loss and delay information and, instead, the bits would be set to arbitrary values. The observers would need to be ready to ignore flows with delay and loss information more resembling noise than the expected signal.

6. Examples of Application

6.1. QUIC

The binding of a delay signal to QUIC is partially described in [QUIC-TRANSPORT], which adds the Spin bit to the first byte of the short packet header, leaving two reserved bits for future use.

To implement the additional signals discussed in this document, the first byte of the short packet header can be modified as follows:

- the Delay bit (D) can be placed in the first reserved bit (i.e. the fourth most significant bit _0x10_) while the round trip loss bit (T) in the second reserved bit (i.e. the fifth most significant bit _0x08_); the proposed scheme is:

  0 1 2 3 4 5 6 7
  +---------------+
  |0|1|S|D|T|K|P|P|
  +---------------+

  Scheme 1

- alternatively, a two bits loss signal (QL or QR) can be placed in both reserved bits; the proposed schemes, in this case, are:

  0 1 2 3 4 5 6 7
  +---------------+
  |0|1|S|Q|L|K|P|P|
  +---------------+

  Scheme 2A

  0 1 2 3 4 5 6 7
  +---------------+
  |0|1|S|Q|R|K|P|P|
  +---------------+

  Scheme 2B
A further option would be to substitute the Spin bit with the Delay bit (or hidden Delay bit), leaving the two reserved bits for loss detection. The proposed schemes are:

```
0 1 2 3 4 5 6 7          0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|1|D|Q|L|K|P|P|   OR   |0|1|D^|Q|L|K|P|P|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Scheme 3A

```
0 1 2 3 4 5 6 7          0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|1|D|Q|R|K|P|P|   OR   |0|1|D^|Q|R|K|P|P|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Scheme 3B

6.2. TCP

The signals can be added to TCP by defining bit 4 of byte 13 of the TCP header to carry the Spin bit or the Delay bit, and possibly bits 5 and 6 to carry additional information, like the Delay bit and the round-trip loss bit (DT), or a two bits loss signal (QL or QR).

7. Security Considerations

Passive loss and delay observations have been a part of the network operations for a long time, so exposing loss and delay information to the network does not add new security concerns for protocols that are currently observable.

In the absence of packet loss, Q and R bits signals do not provide any information that cannot be observed by simply counting packets transiting a network path. In the presence of packet loss, Q and R bits will disclose the loss, but this is information about the environment and not the endpoint state. The L bit signal discloses internal state of the protocol’s loss detection machinery, but this state can often be gleamed by timing packets and observing congestion controller response.

Hence, loss bits do not provide a viable new mechanism to attack data integrity and secrecy.

The described techniques can generally apply to different communication protocols operating in different security environments. An implementation of these techniques for a particular protocol must consider specifics of the protocol and its expected operating
environment. For example, security considerations for QUIC, discussed in [QUIC-TRANSPORT] and [QUIC-TLS], consider a possibility of active and passive attackers in the network as well as attacks on specific QUIC mechanisms.

7.1. Optimistic ACK Attack

A defense against an Optimistic ACK Attack, described in [QUIC-TRANSPORT], involves a sender randomly skipping packet numbers to detect a receiver acknowledging packet numbers that have never been received. The Q bit signal may inform the attacker which packet numbers were skipped on purpose and which had been actually lost (and are, therefore, safe for the attacker to acknowledge). To use the Q bit for this purpose, the attacker must first receive at least an entire Q Block of packets, which renders the attack ineffective against a delay-sensitive congestion controller.

A protocol that is more susceptible to an Optimistic ACK Attack with the loss signal provided by Q bit and uses a loss-based congestion controller, should shorten the current Q Block by the number of skipped packets numbers. For example, skipping a single packet number will invert the square signal one outgoing packet sooner.

Similar considerations apply to the R bit, although a shortened R Block along with a matching skip in packet numbers does not necessarily imply a lost packet, since it could be due to a lost packet on the reverse path along with a deliberately skipped packet by the sender.

8. Privacy Considerations

To minimize unintentional exposure of information, loss bits provide an explicit loss signal - a preferred way to share information per [RFC8558].

New protocols commonly have specific privacy goals, and loss reporting must ensure that loss information does not compromise those privacy goals. For example, [QUIC-TRANSPORT] allows changing Connection IDs in the middle of a connection to reduce the likelihood of a passive observer linking old and new sub-flows to the same device. A QUIC implementation would need to reset all counters when it changes the destination (IP address or UDP port) or the Connection ID used for outgoing packets. It would also need to avoid incrementing Unreported Loss counter for loss of packets sent to a different destination or with a different Connection ID.
9. IANA Considerations

This document makes no request of IANA.

10. Contributors

The following people provided valuable contributions to this document:

- Marcus Ihlar, Ericsson, marcus.ihlar@ericsson.com
- Jari Arkko, Ericsson, jari.arkko@ericsson.com
- Emile Stephan, Orange, emile.stephan@orange.com

11. Acknowledgements

The authors would like to thank the QUIC WG for their contributions, Christian Huitema for implementing Q and L bits in his picoquic stack, and Ike Kunze for providing constructive reviews and helpful suggestions.

12. References

12.1. Normative References


12.2. Informative References

[ACCURATE]

[AltMark]

[ANRW19-PM-QUIC]

[I-D.trammell-ippm-spin]

[I-D.trammell-tsvwg-spin]
[IPv6AltMark]

[QUIC-TLS]

[QUIC-TRANSPORT]


[SPIN-BIT]

[UDP-OPTIONS]

[UDP-SURPLUS]
Herbert, T., "UDP Surplus Header", draft-herbert-udp-space-hdr-01 (work in progress), July 2019.
Authors’ Addresses

Mauro Cociglio
Telecom Italia - TIM
Via Reiss Romoli, 274
Torino 10148
Italy
EMail: mauro.cociglio@telecomitalia.it

Alexandre Ferrieux
Orange Labs
EMail: alexandre.ferrieux@orange.com

Giuseppe Fioccola
Huawei Technologies
Riesstrasse, 25
Munich 80992
Germany
EMail: giuseppe.fioccola@huawei.com

Igor Lubashev
Akamai Technologies
EMail: ilubashe@akamai.com

Fabio Bulgarella
Telecom Italia - TIM
Via Reiss Romoli, 274
Torino 10148
Italy
EMail: fabio.bulgarella@guest.telecomitalia.it

Isabelle Hamchaoui
Orange Labs
EMail: isabelle.hamchaoui@orange.com
Integrity of In-situ OAM Data Fields
draft-ietf-ippm-ioam-data-integrity-02

Abstract

In-situ Operations, Administration, and Maintenance (IOAM) records operational and telemetry information in the packet while the packet traverses a path in the network. IETF protocols require features to ensure their security. This document describes the integrity protection of IOAM-Data-Fields.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 6, 2023.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction ........................................................................................................ 2
2. Conventions ......................................................................................................... 3
3. Threat Analysis .................................................................................................... 4
   3.1. Modification: IOAM-Data-Fields ............................................................... 4
   3.2. Modification: IOAM Option-Type Headers .............................................. 5
   3.3. Injection: IOAM-Data-Fields .................................................................. 5
   3.4. Injection: IOAM Option-Type Headers .................................................... 6
   3.5. Management and Exporting ................................................................. 6
   3.6. Delay ................................................................................................. 6
   3.7. Threat Summary .................................................................................. 7
4. Integrity Protected Option-Types ................................................................. 7
   4.1. Integrity Protected Trace Option-Types .................................................. 8
   4.2. Integrity Protected POT Option-Type .................................................... 9
   4.3. Integrity Protected E2E Option-Type ..................................................... 9
5. Methods for space optimized integrity protection ....................................... 10
   5.1. Symmetric key based signature ............................................................ 11
   5.2. Asymmetric key based signature .......................................................... 11
6. IANA Considerations .......................................................................................... 11
   6.1. IOAM Option-Type Registry ............................................................... 11
   6.2. IOAM Integrity Protection Algorithm Suite Registry ....................... 12
7. Security Considerations .................................................................................... 12
   7.1. Replay protection ................................................................................ 13
8. Acknowledgements ............................................................................................ 13
9. References ........................................................................................................ 13
   9.1. Normative References ....................................................................... 13
   9.2. Informative References ..................................................................... 13
Authors’ Addresses .................................................................................................. 14

1. Introduction

"In-situ" Operations, Administration, and Maintenance (IOAM) records OAM information within the packet while the packet traverses a particular network domain. The term "in-situ" refers to the fact that the OAM data is added to the data packets rather than being sent within packets specifically dedicated to OAM. IOAM is to complement mechanisms such as Ping or Traceroute. In terms of "active" or "passive" OAM, "in-situ" OAM can be considered a hybrid OAM type. "In-situ" mechanisms do not require extra packets to be sent. IOAM adds information to the already available data packets and therefore cannot be considered passive. In terms of the classification given

Brockners, et al. Expires January 6, 2023
in [RFC7799], IOAM could be portrayed as Hybrid Type I. IOAM mechanisms can be leveraged where mechanisms using, e.g., ICMP do not apply or do not offer the desired results, such as proving that a certain traffic flow takes a pre-defined path, SLA verification for the data traffic, detailed statistics on traffic distribution paths in networks that distribute traffic across multiple paths, or scenarios in which probe traffic is potentially handled differently from regular data traffic by the network devices.

[RFC9197] assumes that IOAM is deployed in limited domains, where an operator has means to select, monitor, and control the access to all the networking devices, making the domain a trusted network. As such, IOAM-Data-Fields are carried in clear within packets and there are no protections against any node or middlebox tampering with the data. IOAM-Data-Fields collected in an untrusted or semi-trusted environment require integrity protection to support critical operational decisions.

The following considerations and requirements are to be taken into account in addition to addressing the problem of detectability of any integrity breach of the IOAM-Data-Fields collected:

1. IOAM data is processed by the data plane, hence viability of any method to prove integrity of the IOAM-Data-Fields must be feasible at data plane processing/forwarding rates (IOAM might be applied to all traffic a router forwards).

2. IOAM data is carried within packets. Additional space required to prove integrity of the IOAM-Data-Fields needs to be optimal, i.e. should not exceed the MTU or have adverse effect on packet processing.

3. Replay protection of older IOAM data should be possible. Without replay protection, a rogue node can present the old IOAM data, masking any ongoing network issues/activity and making the IOAM-Data-Fields collection useless.

This document defines the methods to protect the integrity of IOAM-Data-Fields, using the IOAM Option-Types specified in [RFC9197] as an example. The methods similarly apply to other IOAM Option-Types which contain IOAM-Data-Fields.

2. Conventions

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].
Abbreviations used in this document:

IOAM: In-situ Operations, Administration, and Maintenance
MTU: Maximum Transmit Unit
OAM: Operations, Administration, and Maintenance
POT: Proof of Transit
E2E: Edge to Edge

3. Threat Analysis

This section presents a threat analysis of integrity-related threats in the context of IOAM. The threats that are discussed are assumed to be independent of the lower layer protocols; it is assumed that threats at other layers are handled by security mechanisms that are deployed at these layers.

This document is focused on integrity protection for IOAM-Data-Fields. Thus the threat analysis includes threats that are related to or result from compromising the integrity of IOAM-Data-Fields. Other security aspects such as confidentiality are not within the scope of this document.

Throughout the analysis there is a distinction between on-path and off-path attackers. As discussed in [RFC9055], on-path attackers are located in a position that allows interception and modification of in-flight protocol packets, whereas off-path attackers can only attack by generating protocol packets.

The analysis also includes the impact of each of the threats. Generally speaking, the impact of a successful attack on an OAM protocol [RFC7276] is a false illusion of nonexistent failures or preventing the detection of actual ones; in both cases, the attack may result in denial of service (DoS). Furthermore, creating the false illusion of a nonexistent issue may trigger unnecessary processing in some of the IOAM nodes along the path, and may cause more IOAM-related data to be exported to the management plane than is conventionally necessary. Beyond these general impacts, threat-specific impacts are discussed in each of the subsections below.

3.1. Modification: IOAM-Data-Fields

Threat
An attacker can maliciously modify the IOAM-Data-Fields of in-transit packets. The modification can either be applied to all packets or selectively applied to a subset of the en route packets. This threat is applicable to on-path attackers.

Impact

By systematically modifying the IOAM-Data-Fields of some or all of the in-transit packets, an attacker can create a false picture of the paths in the network, the existence of faulty nodes and their location, and the network performance.

3.2. Modification: IOAM Option-Type Headers

Threat

An on-path attacker can modify the header in IOAM Option-Types in order to change or disrupt the behavior of nodes processing IOAM-Data-Fields along the path. This threat is not within the scope of this document.

Impact

Changing the header of IOAM Option-Types may have several implications. An attacker can maliciously increase the processing overhead in nodes that process IOAM-Data-Fields and increase the on-the-wire overhead of IOAM-Data-Fields, for example by modifying the IOAM-Trace-Type field in the IOAM Trace Option-Type header. An attacker can also prevent some of the nodes that process IOAM-Data-Fields from incorporating IOAM-Data-Fields, by modifying the RemainingLen field in the IOAM Trace Option-Type header.

3.3. Injection: IOAM-Data-Fields

Threat

An attacker can inject packets with IOAM Option-Types and IOAM-Data-Fields. This threat is applicable to both on-path and off-path attackers.

Impact

This attack and its impacts are similar to Section 3.1.
3.4. Injection: IOAM Option-Type Headers

Threat

An attacker can inject packets with IOAM Option-Type headers, thus manipulating other nodes that process IOAM-Data-Fields in the network. This threat is applicable to both on-path and off-path attackers. This threat is not within the scope of this document.

Impact

This attack and its impacts are similar to Section 3.2.

3.5. Management and Exporting

Threat

Attacks that compromise the integrity of IOAM-Data-Fields can be applied at the management plane, e.g., by manipulating network management packets. Furthermore, the integrity of IOAM-Data-Fields that are exported to a receiving entity can also be compromised. Management plane attacks are not within the scope of this document; the network management protocol is expected to include inherent security capabilities. The integrity of exported data is also not within the scope of this document. It is expected that the specification of the export format will discuss the relevant security aspects.

Impact

Malicious manipulation of the management protocol can cause nodes that process IOAM-Data-Fields to malfunction, to be overloaded, or to incorporate unnecessary IOAM-Data-Fields into user packets. The impact of compromising the integrity of exported IOAM-Data-Fields is similar to the impacts of previous threats that were described in this section.

3.6. Delay

Threat

An on-path attacker may delay some or all of the in-transit packets that include IOAM-Data-Fields in order to create the false illusion of congestion. Delay attacks are well known in the context of deterministic networks [RFC9055] and synchronization [RFC7384], and may be somewhat mitigated in these environments by using redundant paths in a way that is resilient to an attack along one of the paths. This approach does not address the threat.
in the context of IOAM, as it does not meet the requirement to
measure a specific path or to detect a problem along the path. It
is noted that this threat is not within the scope of the threats
that are mitigated in this document.

Impact

Since IOAM can be applied to a fraction of the traffic, an
attacker can detect and delay only the packets that include IOAM-
Data-Fields, thus preventing the authenticity of delay and load
measurements.

3.7. Threat Summary

<table>
<thead>
<tr>
<th>Threat</th>
<th>In scope</th>
<th>Out of scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modification: IOAM-Data-Fields</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Modification: IOAM Option-Type Headers</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Injection: IOAM-Data-Fields</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Injection: IOAM Option-Type Headers</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Management and Exporting</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Delay</td>
<td></td>
<td>+</td>
</tr>
</tbody>
</table>

Figure 1: Threat Analysis Summary

4. Integrity Protected Option-Types

This section defines new IOAM Option-Types to be allocated in the
IOAM Option-Type Registry. Their purpose is to carry IOAM-Data-
Fields with integrity protection. Each of the IOAM Option-Types
defined in [RFC9197] is extended as follows:

64 IOAM Pre-allocated Trace Integrity Protected Option-Type:
corresponds to the IOAM Pre-allocated Trace Option-Type with
integrity protection.

65 IOAM Incremental Trace Integrity Protected Option-Type:
corresponds to the IOAM Incremental Trace Option-Type with
integrity protection.
66 IOAM POT Integrity Protected Option-Type: corresponds to the IOAM POT Option-Type with integrity protection.

67 IOAM E2E Integrity Protected Option-Type: corresponds to the IOAM E2E Option-Type with integrity protection.

The Integrity Protection subheader follows the IOAM Option-Type header when the IOAM Option-Type is an Integrity Protected Option-Type. It is defined as follows:

```
+-------------------------------------------+
<table>
<thead>
<tr>
<th>Signature-suite</th>
<th>Nonce length</th>
<th>Reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonce</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Signature</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
+-------------------------------------------+
```

Signature-suite: 8-bit unsigned integer. This field defines the algorithms used to compute the digest and the signature over the IOAM-Data-Fields.

Nonce length: 8-bit unsigned integer. This field specifies the length of the Nonce in octets.

Reserved: 16-bit Reserved field. MUST be set to zero upon transmission and ignored upon receipt.

Nonce: Variable length field with length specified in Nonce length.

Signature: Digital signature value generated by the method and algorithm specified by Signature-suite.

4.1. Integrity Protected Trace Option-Types

Both the IOAM Pre-allocated Trace Option-Type header and the IOAM Incremental Trace Option-Type header, as defined in [RFC9197], are followed by the Integrity Protection subheader when the IOAM Option-Type is respectively set to the IOAM Pre-allocated Trace Integrity Protected Option-Type or the IOAM Incremental Trace Integrity Protected Option-Type:
4.2. Integrity Protected POT Option-Type

The IOAM POT Option-Type header, as defined in [RFC9197], is followed by the Integrity Protection subheader when the IOAM Option-Type is set to the IOAM POT Integrity Protected Option-Type:

    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Namespace-ID | IOAM-POT-Type | IOAM-POT-Flags |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Signature-suite | Nonce length | Reserved |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Nonce |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Signature |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

4.3. Integrity Protected E2E Option-Type

The IOAM E2E Option-Type header, as defined in [RFC9197], is followed by the Integrity Protection subheader when the IOAM Option-Type is set to the IOAM E2E Integrity Protected Option-Type:

    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Namespace-ID | IOAM-POT-Type | IOAM-POT-Flags |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Signature-suite | Nonce length | Reserved |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Nonce |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Signature |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
5. Methods for space optimized integrity protection

Methods for space optimized integrity protection can leverage symmetric or asymmetric key based signatures, as described in the subsections below. The Signature consumes 32 octets and is carried only once for the entire packet. In case of performance concerns, such method can be applied to a subset of the traffic by using sampling of data to enable IOAM with integrity protection. Both symmetric and asymmetric signature methods work similarly, as follows:

1. The encapsulating node creates a nonce and stores it in the Nonce field of the Integrity Protection subheader. The signature is generated over the Nonce field and the hash of IOAM-Data-Fields it has inserted, i.e., sign(Nonce || hash(IOAM-Data-Fields)). IOAM-Data-Fields supposed to be modified by other IOAM nodes on the path MUST be excluded from the signature (e.g., the POT Cumulative field). The signature is stored in the Signature field of the Integrity Protection subheader. Important note: if all the inserted IOAM-Data-Fields are supposed to be modified by other IOAM nodes on the path, or if there is no IOAM-Data-Field inserted at all, then the encapsulating node MUST NOT use an Integrity Protected Option-Type.

2. A transit node generates a signature over the Signature field and the hash of IOAM-Data-Fields it has inserted, i.e., sign(Signature || hash(IOAM-Data-Fields)). IOAM-Data-Fields modified in-place by the transit node MUST be excluded from the signature (e.g., the POT Cumulative field). The signature is stored in the Signature field of the Integrity Protection subheader. Important note: if the transit node does not insert IOAM-Data-Fields (e.g., it only modifies IOAM-Data-Fields in-place, or does nothing), then the transit node MUST NOT generate a signature and MUST NOT update the Signature field.
3. The decapsulating node (aka the Validator) is responsible for the integrity verification of the IOAM-Data-Fields collected. Serving as the Validator, the decapsulating node MUST NOT generate a signature based on IOAM-Data-Fields it has inserted, if any, and therefore MUST NOT update the Signature field. To validate the IOAM-Data-Fields integrity, the Validator recomputes the signature by iteratively following the same procedure as for the encapsulating and transit nodes, in that order, using their respective keys (see Section 5.1 or Section 5.2 depending on the approach, i.e., symmetric or asymmetric). The recomputed signature is then compared to the Signature field. It is trivial in some cases (e.g., with POT Type-0 or E2E Option-Types), where only the encapsulating node generates a signature, as specified by the method described in this section. For other cases where transit nodes also generate a signature (e.g., with Trace Option-Types), node-ids MUST be included in IOAM-Data-Fields. Details on how the mapping between node-ids and keys is implemented on the Validator are outside the scope of this document.

5.1. Symmetric key based signature

This method assumes that symmetric keys have been distributed to the respective nodes as well as the Validator (the Validator receives all the keys). The details of the mechanisms responsible for key distribution are outside the scope of this document.

This method MUST use an algorithm pair defined in Section 6.2 and the approach MUST be symmetric.

5.2. Asymmetric key based signature

This method assumes that asymmetric keys have been generated per IOAM node and the respective nodes can access their keys (the Validator receives all the public keys). The details of the mechanisms responsible for key distribution are outside the scope of this document.

This method MUST use an algorithm pair defined in Section 6.2 and the approach MUST be asymmetric.

6. IANA Considerations

6.1. IOAM Option-Type Registry

This draft defines the following new code points in the IOAM Option-Type Registry:

64 IOAM Pre-allocated Trace Integrity Protected Option-Type
65 IOAM Incremental Trace Integrity Protected Option-Type

66 IOAM POT Integrity Protected Option-Type

67 IOAM E2E Integrity Protected Option-Type

6.2.  IOAM Integrity Protection Algorithm Suite Registry

"IOAM Integrity Protection Algorithm Suite Registry" in the "In-Situ OAM (IOAM) Protocol Parameters" group. The one-octet "IOAM Integrity Protection Algorithm Suite Registry" identifiers assigned by IANA identify the digest algorithm and signature algorithm used in the Signature Suite Identifier field. IANA has registered the following algorithm suite identifiers for the digest algorithm and for the signature algorithm.

<table>
<thead>
<tr>
<th>Algorithm Suite Identifier</th>
<th>Digest Algorithm</th>
<th>Signature Algorithm</th>
<th>Specification Pointer</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Reserved</td>
<td>Reserved</td>
<td>This document</td>
<td>None</td>
</tr>
<tr>
<td>0x01</td>
<td>SHA-256</td>
<td>ECDSA P-256</td>
<td>[SHS] [DSS] [RFC6090]</td>
<td>Asymmetric</td>
</tr>
<tr>
<td>0x02</td>
<td>SHA-256</td>
<td>AES-256</td>
<td>[AES] [NIST.800-38D]</td>
<td>Symmetric</td>
</tr>
<tr>
<td>0x03-0xFF</td>
<td>Unassigned</td>
<td>Unassigned</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Future assignments are to be made using the Standards Action process defined in [RFC8126]. Assignments consist of the one-octet algorithm suite identifier value and the associated digest algorithm name and signature algorithm name.

7.  Security Considerations

This section discusses additional security aspects.
7.1. Replay protection

The nonce makes a signature chain unique but does not necessarily prevent replay attacks. To enable replay protection, the encapsulating node and the Validator MUST use a common, unique nonce.

8. Acknowledgements

The authors would like to thank Santhosh N, Rakesh Kandula, Saiprasad Muchala, Al Morton, Greg Mirsky, Benjamin Kaduk and Martin Duke for their comments and advice.

9. References

9.1. Normative References


9.2. Informative References


Authors’ Addresses

Brockners, et al. Expires January 6, 2023
Frank Brockners  
Cisco Systems, Inc.  
Hansaallee 249, 3rd Floor  
DUESSELDORF, NORDRHEIN-WESTFALEN 40549  
Germany  
Email: fbrockne@cisco.com

Shwetha Bhandari  
Thoughtspot  
3rd Floor, Indiqube Orion, 24th Main Rd, Garden Layout, HSR Layout  
Bangalore, KARNATAKA 560 102  
India  
Email: shwetha.bhandari@thoughtspot.com

Tal Mizrahi  
Huawei  
8-2 Matam  
Haifa 3190501  
Israel  
Email: tal.mizrahi.phd@gmail.com

Justin Iurman  
Universite de Liege  
10, Allee de la decouverte (B28)  
Sart-Tilman, LIEGE 4000  
Belgium  
Email: justin.iurman@uliege.be
Abstract

In-situ Operations, Administration, and Maintenance (IOAM) collects operational and telemetry information in the packet while the packet traverses a path between two points in the network. This document provides a framework for IOAM deployment and provides best current practices.

Status of This Memo

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

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

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

This Internet-Draft will expire on October 13, 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents.
1. Introduction

"In-situ" Operations, Administration, and Maintenance (IOAM) collects OAM information within the packet while the packet traverses a particular network domain. The term "in-situ" refers to the fact

<table>
<thead>
<tr>
<th>Table of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
</tr>
<tr>
<td>2. Conventions</td>
</tr>
<tr>
<td>3. IOAM Deployment: Domains And Nodes</td>
</tr>
<tr>
<td>4. Types of IOAM</td>
</tr>
<tr>
<td>4.1. Per-hop Tracing IOAM</td>
</tr>
<tr>
<td>4.2. Proof of Transit IOAM</td>
</tr>
<tr>
<td>4.3. Edge to Edge IOAM</td>
</tr>
<tr>
<td>4.4. Direct Export IOAM</td>
</tr>
<tr>
<td>5. IOAM Encapsulations</td>
</tr>
<tr>
<td>5.1. IPv6</td>
</tr>
<tr>
<td>5.2. NSH</td>
</tr>
<tr>
<td>5.3. BIER</td>
</tr>
<tr>
<td>5.4. GRE</td>
</tr>
<tr>
<td>5.5. Geneve</td>
</tr>
<tr>
<td>5.6. Segment Routing</td>
</tr>
<tr>
<td>5.7. Segment Routing for IPv6</td>
</tr>
<tr>
<td>5.8. VXLAN-GPE</td>
</tr>
<tr>
<td>6. IOAM Data Export</td>
</tr>
<tr>
<td>7. IOAM Deployment Considerations</td>
</tr>
<tr>
<td>7.1. IOAM Namespaces</td>
</tr>
<tr>
<td>7.2. IOAM Layering</td>
</tr>
<tr>
<td>7.3. IOAM Trace Option Types</td>
</tr>
<tr>
<td>7.4. Traffic-sets That IOAM Is Applied To</td>
</tr>
<tr>
<td>7.5. IOAM Loopback Mode</td>
</tr>
<tr>
<td>7.6. IOAM Active Mode</td>
</tr>
<tr>
<td>7.7. Brown Field Deployments: IOAM Unaware Nodes</td>
</tr>
<tr>
<td>8. IOAM Manageability</td>
</tr>
<tr>
<td>9. IANA Considerations</td>
</tr>
<tr>
<td>10. Security Considerations</td>
</tr>
<tr>
<td>11. Acknowledgements</td>
</tr>
<tr>
<td>12. References</td>
</tr>
<tr>
<td>12.1. Normative References</td>
</tr>
<tr>
<td>12.2. Informative References</td>
</tr>
<tr>
<td>Authors’ Addresses</td>
</tr>
</tbody>
</table>
that the OAM data is added to the data packets rather than is being sent within packets specifically dedicated to OAM. IOAM is to complement mechanisms such as Ping, Traceroute, or other active probing mechanisms. In terms of "active" or "passive" OAM, "in-situ" OAM can be considered a hybrid OAM type. "In-situ" mechanisms do not require extra packets to be sent. IOAM adds information to the already available data packets and therefore cannot be considered passive. In terms of the classification given in [RFC7799] IOAM could be portrayed as Hybrid Type I. IOAM mechanisms can be leveraged where mechanisms using e.g. ICMP do not apply or do not offer the desired results, such as proving that a certain traffic flow takes a pre-defined path, SLA verification for the live data traffic, detailed statistics on traffic distribution paths in networks that distribute traffic across multiple paths, or scenarios in which probe traffic is potentially handled differently from regular data traffic by the network devices.

2. Conventions

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.

Abbreviations used in this document:

BIER: Bit Index Explicit Replication
E2E: Edge to Edge
Geneve: Generic Network Virtualization Encapsulation [I-D.ietf-nvo3-geneve]
GRE: Generic Routing Encapsulation
IOAM: In-situ Operations, Administration, and Maintenance
MTU: Maximum Transmit Unit
NSH: Network Service Header [RFC8300]
OAM: Operations, Administration, and Maintenance
POT: Proof of Transit
SFC: Service Function Chain
3. IOAM Deployment: Domains And Nodes

IOAM is focused on "limited domains" as defined in [RFC8799]. IOAM is not targeted for a deployment on the global Internet. The part of the network which employs IOAM is referred to as the "IOAM-Domain". For example, an IOAM-domain can include an enterprise campus using physical connections between devices or an overlay network using virtual connections / tunnels for connectivity between said devices. An IOAM-domain is defined by its perimeter or edge. The operator of an IOAM-domain is expected to put provisions in place to ensure that packets which contain IOAM data fields do not leak beyond the edge of an IOAM domain, e.g. using for example packet filtering methods. The operator should consider the potential operational impact of IOAM to mechanisms such as ECMP load-balancing schemes (e.g., load-balancing schemes based on packet length could be impacted by the increased packet size due to IOAM), path MTU (i.e., ensure that the MTU of all links within a domain is sufficiently large to support the increased packet size due to IOAM) and ICMP message handling.

An IOAM-Domain consists of "IOAM encapsulating nodes", "IOAM decapsulating nodes" and "IOAM transit nodes". The role of a node (i.e., encapsulating, transit, decapsulating) is defined within an IOAM-Namespace (see below). A node can have different roles in different IOAM-Namespaces.

An "IOAM encapsulating node" incorporates one or more IOAM-Option-Types into packets that IOAM is enabled for. If IOAM is enabled for a selected subset of the traffic, the IOAM encapsulating node is responsible for applying the IOAM functionality to the selected subset.

An "IOAM transit node" updates one or more of the IOAM-Data-Fields. If both the Pre-allocated and the Incremental Trace Option-Types are present in the packet, each IOAM transit node will update at most one of these Option-Types. A transit node does not add new IOAM-Option-Types to a packet, and does not change the IOAM-Data-Fields of an IOAM Edge-to-Edge Option-Type.

An "IOAM decapsulating node" removes IOAM-Option-Type(s) from packets.
The role of an IOAM-encapsulating, IOAM-transit or IOAM-decapsulating node is always performed within a specific IOAM-Namespace. This means that an IOAM node which is e.g., an IOAM-decapsulating node for IOAM-Namespace "A" but not for IOAM-Namespace "B" will only remove the IOAM-Option-Types for IOAM-Namespace "A" from the packet. An IOAM decapsulating node situated at the edge of an IOAM domain removes all IOAM-Option-Types and associated encapsulation headers for all IOAM-Namespace from the packet.

IOAM-Namespace allow for a namespace-specific definition and interpretation of IOAM-Data-Fields. An interface-id could for example point to a physical interface (e.g., to understand which physical interface of an aggregated link is used when receiving or transmitting a packet) whereas in another case it could refer to a logical interface (e.g., in case of tunnels). Please refer to Section 7.1 for a discussion of IOAM-Namespace.

<table>
<thead>
<tr>
<th>Export of IOAM data (optional)</th>
<th>Export of IOAM data (optional)</th>
<th>Export of IOAM data (optional)</th>
<th>Export of IOAM data (optional)</th>
</tr>
</thead>
<tbody>
<tr>
<td>User packets</td>
<td>Encapsulating Node</td>
<td>Transit Node</td>
<td>Transit Node</td>
</tr>
<tr>
<td>+----+</td>
<td>A</td>
<td>B</td>
<td>Decapsulating Node</td>
</tr>
</tbody>
</table>

Figure 1: Roles of IOAM nodes

IOAM nodes which add or remove the IOAM-Data-Fields can also update the IOAM-Data-Fields at the same time. Or in other words, IOAM encapsulating or decapsulating nodes can also serve as IOAM transit nodes at the same time. Note that not every node in an IOAM domain needs to be an IOAM transit node. For example, a deployment might require that packets traverse a set of firewalls which support IOAM. In that case, only the set of firewall nodes would be IOAM transit nodes rather than all nodes.

4. Types of IOAM

IOAM supports different modes of operation, which are differentiated by the type of IOAM data fields being carried in the packet, the data being collected, the type of nodes which collect or update data as well as whether and how nodes export IOAM information.
Per-hop tracing: OAM information about each IOAM node a packet traverses is collected and stored within the user data packet as the packet progresses through the IOAM domain. Potential uses of IOAM per-hop tracing include:

* Understand the different paths different packets traverse between an IOAM encapsulating and an IOAM decapsulating node in a network that uses load balancing such as Equal Cost Multi-Path (ECMP). This information could be used to tune the algorithm for ECMP for optimized network resource usage.

* Operations/Troubleshooting: Understand which path a particular packet or set of packets take as well as what amount of jitter and delay different IOAM nodes in the path contribute to the overall delay and jitter between the IOAM encapsulating node and the IOAM decapsulating node.

Proof-of-transit: Information that a verifier node can use to verify whether a packet has traversed all nodes that is supposed to traverse is stored within the user data packet. Proof-of-transit could for example be used to verify that a packet indeed passes through all services of a service function chain (e.g., verify whether a packet indeed traversed the set of firewalls that it is expected to traverse), or whether a packet indeed took the expected path.

Edge-to-edge: OAM information which is specific to the edges of an IOAM domain is collected and stored within the user data packet. Edge-to-Edge OAM could be used to gather operational information about a particular network domain, such as the delay and jitter incurred by that network domain or the traffic matrix of the network domain.

Direct export: OAM information about each IOAM node a packet traverses is collected and immediately exported to a collector. Direct export could be used in situations where per-hop tracing information is desired, but gathering the information within the packet - as with per-hop tracing - is not feasible. Rather than automatically correlating the per-hop tracing information, as done with per-hop tracing, direct export requires a collector to correlate the information from the individual nodes. In addition, all nodes enabled for direct export need to be capable to export the IOAM information to the collector.
4.1. Per-hop Tracing IOAM

"IOAM tracing data" is expected to be collected at every IOAM transit node that a packet traverses to ensure visibility into the entire path a packet takes within an IOAM-Domain. I.e., in a typical deployment all nodes in an IOAM-Domain would participate in IOAM and thus be IOAM transit nodes, IOAM encapsulating or IOAM decapsulating nodes. If not all nodes within a domain are IOAM capable, IOAM tracing information (i.e., node data, see below) will only be collected on those nodes which are IOAM capable. Nodes which are not IOAM capable will forward the packet without any changes to the IOAM-Data-Fields. The maximum number of hops and the minimum path MTU of the IOAM domain is assumed to be known.

IOAM offers two different trace Option-Types, the "incremental" Option-Type as well as the "pre-allocated" Option-Type. For a discussion which of the two option types is the most suitable for an implementation and/or deployment, see Section 7.3.

Every node data entry holds information for a particular IOAM transit node that is traversed by a packet. The IOAM decapsulating node removes the IOAM-Option-Type(s) and processes and/or exports the associated data. All IOAM-Data-Fields are defined in the context of an IOAM-Namespace.

IOAM tracing can collect the following types of information:

- Identification of the IOAM node. An IOAM node identifier can match to a device identifier or a particular control point or subsystem within a device.
- Identification of the interface that a packet was received on, i.e. ingress interface.
- Identification of the interface that a packet was sent out on, i.e. egress interface.
- Time of day when the packet was processed by the node as well as the transit delay. Different definitions of processing time are feasible and expected, though it is important that all devices of an in-situ OAM domain follow the same definition.
- Generic data: Format-free information where syntax and semantic of the information is defined by the operator in a specific deployment. For a specific IOAM-Namespace, all IOAM nodes should interpret the generic data the same way. Examples for generic IOAM data include geo-location information (location of the node at the time the packet was processed), buffer queue fill level or
o Information to detect whether IOAM trace data was added at every hop or whether certain hops in the domain weren’t IOAM transit nodes.

o Data that relates to how the packet traversed a node (transit delay, buffer occupancy in case the packet was buffered, queue depth in case the packet was queued)

The Option-Types of incremental tracing and pre-allocated tracing are defined in [I-D.ietf-ippm-ioam-data].

4.2. Proof of Transit IOAM

IOAM Proof of Transit Option-Type is to support path or service function chain [RFC7665] verification use cases. Proof-of-transit uses methods like nested hashing or nested encryption of the IOAM data or mechanisms such as Shamir’s Secret Sharing Schema (SSSS).

The IOAM Proof of Transit Option-Type consist of a fixed size "IOAM proof of transit option header" and "IOAM proof of transit option data fields". For details see [I-D.ietf-ippm-ioam-data].

4.3. Edge to Edge IOAM

The IOAM Edge-to-Edge Option-Type is to carry data that is added by the IOAM encapsulating node and interpreted by IOAM decapsulating node. The IOAM transit nodes may process the data but must not modify it.

The IOAM Edge-to-Edge Option-Type consist of a fixed size "IOAM Edge-to-Edge Option-Type header" and "IOAM Edge-to-Edge Option-Type data fields". For details see [I-D.ietf-ippm-ioam-data].

4.4. Direct Export IOAM

Direct Export is an IOAM mode of operation within which IOAM data to be directly exported to a collector rather than being collected within the data packets. The IOAM Direct Export Option-Type consist of a fixed size "IOAM direct export option header". Direct Export for IOAM is defined in [I-D.ietf-ippm-ioam-direct-export].
5. IOAM Encapsulations

IOAM data fields and associated data types for in-situ OAM are defined in [I-D.ietf-ippm-ioam-data]. The in-situ OAM data field can be transported by a variety of transport protocols, including NSH, Segment Routing, Geneve, BIER, IPv6, etc.

5.1. IPv6

IOAM encapsulation for IPv6 is defined in [I-D.ietf-ippm-ioam-ipv6-options], which also discussed IOAM deployment considerations for IPv6 networks.

5.2. NSH

IOAM encapsulation for NSH is defined in [I-D.ietf-sfc-ioam-nsh].

5.3. BIER

IOAM encapsulation for BIER is defined in [I-D.xzlnp-bier-ioam].

5.4. GRE

IOAM encapsulation for GRE is outlined as part of the "EtherType Protocol Identification of In-situ OAM Data" in [I-D.weis-ippm-ioam-eth], though no example protocol header stacks are provided in the document. When used with GRE, the IOAM-Option-Types (the below diagram uses "IOAM" as shorthand for IOAM-Option-Types) are sequenced in behind the GRE header that follows the "outer" IP header. Figure 2 shows two example protocol header stacks that use GRE along with IOAM.
5.5. Geneve

IOAM encapsulation for Geneve is defined in [I-D.brockners-ippm-ioam-geneve].

5.6. Segment Routing

IOAM encapsulation for Segment Routing is defined in [I-D.gandhi-spring-ioam-sr-mpls].

5.7. Segment Routing for IPv6

IOAM encapsulation for Segment Routing over IPv6 is defined in [I-D.ali-spring-ioam-srv6].

5.8. VXLAN-GPE

IOAM encapsulation for VXLAN-GPE is defined in [I-D.brockners-ippm-ioam-vxlan-gpe].

6. IOAM Data Export

IOAM nodes collect information for packets traversing a domain that supports IOAM. IOAM decapsulating nodes as well as IOAM transit nodes can choose to retrieve IOAM information from the packet,
process the information further and export the information using e.g., IPFIX.

Raw data export of IOAM data using IPFIX is discussed in [I-D.spiegel-ippm-ioam-rawexport]. "Raw export of IOAM data" refers to a mode of operation where a node exports the IOAM data as it is received in the packet. The exporting node neither interprets, aggregates nor reformats the IOAM data before it is exported. Raw export of IOAM data is to support an operational model where the processing and interpretation of IOAM data is decoupled from the operation of encapsulating/updating/decapsulating IOAM data, which is also referred to as IOAM data-plane operation. The figure below shows the separation of concerns for IOAM export: Exporting IOAM data is performed by the "IOAM node" which performs IOAM data-plane operation, whereas the interpretation of IOAM data is performed by one or several IOAM data processing systems. The separation of concerns is to off-load interpretation, aggregation and formatting of IOAM data from the node which performs data-plane operations. In other words, a node which is focused on data-plane operations, i.e. forwarding of packets and handling IOAM data will not be tasked to also interpret the IOAM data, but can leave this task to another system or a set of systems. For scalability reasons, a single IOAM node could choose to export IOAM data to several IOAM data processing systems. Similarly, there several monitoring systems or analytics systems can be used to further process the data received from the IOAM preprocessing systems. Figure 3 shows an overview of IOAM export, including IOAM data processing systems and monitoring/analytics systems.
7. IOAM Deployment Considerations

This section discusses several aspects of an IOAM deployment, including IOAM Namespaces, IOAM Layering, traffic-sets that IOAM is applied to and IOAM loopback mode.

7.1. IOAM Namespaces

IOAM-Namespaces add further context to IOAM-Option-Types and associated IOAM-Data-Fields. IOAM-Namespaces support several different uses:

- IOAM-Namespaces can be used by an operator to distinguish different operational domains. Devices at domain edges can filter on Namespace-IDs to provide for proper IOAM-Domain isolation.
o IOAM-Namespaces provide additional context for IOAM-Data-Fields and thus ensure that IOAM-Data-Fields are unique and can be interpreted properly by management stations or network controllers. While, for example, the node identifier field does not need to be unique in a deployment (e.g., an operator may wish to use different node identifiers for different IOAM layers, even within the same device; or node identifiers might not be unique for other organizational reasons, such as after a merger of two formerly separated organizations), the combination of node_id and Namespace-ID should always be unique. Similarly, IOAM-Namespaces can be used to define how certain IOAM-Data-Fields are interpreted: IOAM offers three different timestamp format options. The Namespace-ID can be used to determine the timestamp format. IOAM-Data-Fields (e.g., buffer occupancy) which do not have a unit associated are to be interpreted within the context of a IOAM-Namespace.

o IOAM-Namespaces can be used to identify different sets of devices (e.g., different types of devices) in a deployment: If an operator desires to insert different IOAM-Data-Fields based on the device, the devices could be grouped into multiple IOAM-Namespaces. This could be due to the fact that the IOAM feature set differs between different sets of devices, or it could be for reasons of optimized space usage in the packet header. It could also stem from hardware or operational limitations on the size of the trace data that can be added and processed, preventing collection of a full trace for a flow.

* Assigning different IOAM Namespace-IDs to different sets of nodes or network partitions and using the Namespace-ID as a selector at the IOAM encapsulating node, a full trace for a flow could be collected and constructed via partial traces in different packets of the same flow. Example: An operator could choose to group the devices of a domain into two IOAM-Namespaces, in a way that at average, only every second hop would be recorded by any device. To retrieve a full view of the deployment, the captured IOAM-Data-Fields of the two IOAM-Namespaces need to be correlated.

* Assigning different IOAM Namespace-IDs to different sets of nodes or network partitions and using a separate instance of an IOAM-Option-Type for each Namespace-ID, a full trace for a flow could be collected and constructed via partial traces from each IOAM-Option-Type in each of the packets in the flow. Example: An operator could choose to group the devices of a domain into two IOAM-Namespaces, in a way that each IOAM-Namespace is represented by one of two IOAM-Option-Types in the packet. Each node would record data only for the IOAM-Namespace that it
belongs to, ignoring the other IOAM-Option-Type with a IOAM-
Namespace to which it doesn't belong. To retrieve a full view
of the deployment, the captured IOAM-Data-Fields of the two
IOAM-Namespace need to be correlated.

7.2. IOAM Layering

If several encapsulation protocols (e.g., in case of tunneling) are
stacked on top of each other, IOAM-Data-Fields could be present in
different protocol fields at different layers. Layering allows
operators to instrument the protocol layer they want to measure. The
behavior follows the ships-in-the-night model, i.e. IOAM-Data-Fields
in one layer are independent from IOAM-Data-Fields in another layer.
Or in other words: Even though the term "layering" often implies some
form of hierarchy and relationship, in IOAM, layers are independent
from each other and don’t assume any relationship among them. The
different layers could, but do not have to share the same IOAM
encapsulation mechanisms. Similarly, the semantics of the IOAM-Data-
Fields, can, do not have to be associated to across different layers.
For example, a node which inserts node-id information into two
different layers could use "node-id=10" for one layer and "node-
id=1000" for the second layer.

Figure 4 shows an example of IOAM layering. The figure shows a
Geneve tunnel carried over IPv6 which starts at node A and ends at
node D. IOAM information is encapsulated in IPv6 as well as in
Geneve. At the IPv6 layer, node A is IOAM encapsulating node (into
IPv6), node D is the IOAM decapsulating node and node B and node C
are IOAM transit nodes. At the Geneve layer, node A is IOAM
encapsulating node (into Geneve) and node D is IOAM decapsulating
node (from Geneve). The use of IOAM at both layers as shown in the
example here could be used to reveal which nodes of an underlay (here
the IPv6 network) are traversed by tunneled packet in an overlay
(here the Geneve network) – which assumes that the IOAM information
encapsulated by nodes A and D into Geneve and IPv6 is associated to
each other.
7.3. IOAM Trace Option Types

IOAM offers two different IOAM Option-Types for tracing: "Incremental" Trace-Option-Type and "Pre-allocated" Trace-Option-Type. "Incremental" refers to a mode of operation where the packet is expanded at every IOAM node that adds IOAM-Data-Fields. "Pre-Allocated" describes a mode of operation where the IOAM encapsulating node allocates room for all IOAM-Data-Fields in the entire IOAM domain. More specifically:

Pre-allocated Trace-Option: This trace option is defined as a container of node data fields with pre-allocated space for each node to populate its information. This option is useful for implementations where it is efficient to allocate the space once and index into the array to populate the data during transit (e.g., software forwarders often fall into this class).

Incremental Trace-Option: This trace option is defined as a container of node data fields where each node allocates and pushes its node data immediately following the option header.

A deployment can choose to configure and support one or both of the IOAM Trace-Option-Types. The operator decides by means of configuration which Trace-Option-Type(s) will be used for a particular domain. Deployments can mix devices which include either the Incremental Trace-Option-Type or the Pre-allocated Trace-Option-Type, e.g., in case different types of packet forwarders and Brockners, et al. Expires October 13, 2022 [Page 15]
associated different types of IOAM implementations exist in a deployment. As a result, both Option-Types can be present in a packet. IOAM decapsulating nodes remove both types of Trace-Option-Types from the packet.

The two different Option-Types cater to different packet forwarding infrastructures and are to allow an optimized implementation of IOAM tracing:

Pre-allocated Trace-Option: For some implementations of packet forwarders it is efficient to allocate the space for the maximum number of nodes that IOAM Trace Data-Fields should be collected from and insert/update information in the packet at flexible locations, based on a pointer retrieved from a field in the packet. The IOAM encapsulating node allocates an array of the size of the maximum number of nodes that IOAM Trace Data-Fields should be collected from. IOAM transit nodes index into the array to populate the data during transit. Software forwarders often fall into this class of packet forwarder implementations. The maximum number of nodes that IOAM information could be collected from is configured by the operator on the IOAM encapsulating node. The operator has to ensure that the packet with the pre-allocated array that carries the IOAM Data-Fields does not exceed the MTU of any of the links in the IOAM domain.

Incremental Trace-Option: Looking up a pointer contained in the packet and inserting/updating information at a flexible location in the packet as a result of the pointer lookup is costly for some forwarding infrastructures. Hardware-based packet forwarding infrastructures often fall into this category. Consequently, hardware-based packet forwarders could choose to support the incremental IOAM-Trace-Option-Type. The incremental IOAM-Trace-Option-Type eliminates the need for the IOAM transit nodes to read the full array in the Trace-Option-Type and allows packets to grow to the size of the MTU of the IOAM domain. IOAM transit nodes will expand the packet and insert the IOAM-Data-Fields as long as there is space available in the packet, i.e. as long as the size of the packet stays within the bounds of the MTU of any of the links in the IOAM domain. There is no need for the operator to configure the IOAM encapsulation node with the maximum number of nodes that IOAM information could be collected from. The operator has to ensure that the minimum MTU of any of the links in the IOAM domain is known to all IOAM transit nodes.
7.4. Traffic-sets That IOAM Is Applied To

IOAM can be deployed on all or only on subsets of the live user traffic, e.g., per interface, based on an access control list or flow specification defining a specific set of traffic, etc.

7.5. IOAM Loopback Mode

IOAM Loopback is used to trigger each transit device along the path of a packet to send a copy of the data packet back to the source. Loopback allows an IOAM encapsulating node to trace the path to a given destination, and to receive per-hop data about both the forward and the return path. Loopback is enabled by the encapsulating node setting the loopback flag. Looped-back packets use the source address of the original packet as destination address and the address of the node which performs the loopback operation as source address. Nodes which loop back a packet clear the loopback flag before sending the copy back towards the source. Loopack applies to IOAM deployments where the encapsulating node is either a host or the start of a tunnel: For details on IOAM loopback, please refer to [I-D.ietf-ippm-ioam-flags].

7.6. IOAM Active Mode

The IOAM active mode flag indicates that a packet is an active OAM packet as opposed to regular user data traffic. Active mode is expected to be used for active measurement using IOAM. For details on IOAM active mode, please refer to [I-D.ietf-ippm-ioam-flags].

Example use-cases for IOAM Active Mode include:

- Endpoint detailed active measurement: Synthetic probe packets are sent between the source and destination. These probe packets include a Trace Option-Type (i.e., either incremental or pre-allocated). Since the probe packets are sent between the endpoints, these packets are treated as data packets by the IOAM domain, and do not require special treatment at the IOAM layer. The source, which is also the IOAM encapsulating node can choose to set the Active flag, providing an explicit indication that these probe packets are meant for telemetry collection.

- IOAM active measurement using probe packets: Probe packets are generated and transmitted by an IOAM encapsulating node towards a destination which is also the IOAM decapsulating node. Probe packets include a Trace Option-Type (i.e., either incremental or pre-allocated) which has its Active flag set.
o IOAM active measurement using replicated data packets: Probe packets are created by an IOAM encapsulating node by selecting some or all of the en route data packets and replicating them. A selected data packet that is replicated, and its (possibly truncated) copy is forwarded with one or more IOAM option, while the original packet is forwarded normally, without IOAM options. To the extent possible, the original data packet and its replica are forwarded through the same path. The replica includes a Trace Option-Type that has its Active flag set, indicating that the IOAM decapsulating node should terminate it. In this case the IOAM Active flag ensures that the replicated traffic is not forwarded beyond the IOAM domain.

7.7. Brown Field Deployments: IOAM Unaware Nodes

A network can consist of a mix of IOAM aware and IOAM unaware nodes. The encapsulation of IOAM-Data-Fields into different protocols (see also Section 5) are defined such that data packets that include IOAM-Data-Fields do not get dropped by IOAM unaware nodes. For example, packets which contain the IOAM-Trace-Option-Types in IPv6 Hop by Hop extension headers are defined with bits to indicate "00 - skip over this option and continue processing the header". This will ensure that when a node that is IOAM unaware receives a packet with IOAM-Data-Fields included, does not drop the packet.

Deployments which leverage the IOAM-Trace-Option-Type(s) could benefit from the ability to detect the presence of IOAM unaware nodes, i.e. nodes which forward the packet but do not update/add IOAM-Data-Fields in IOAM-Trace-Option-Type(s). The node data that is defined as part of the IOAM-Trace-Option-Type(s) includes a Hop_Lim field associated to the node identifier to detect missed nodes, i.e. "holes" in the trace. Monitoring/Analytics system(s) could utilize this information to account for the presence of IOAM unaware nodes in the network.

8. IOAM Manageability

The YANG model for configuring IOAM in network nodes which support IOAM is defined in [I-D.zhou-ippm-ioam-yang].

A deployment can leverage IOAM profiles is to limit the scope of IOAM features, allowing simpler implementation, verification, and interoperability testing in the context of specific use cases that do not require the full functionality of IOAM. An IOAM profile defines a use case or a set of use cases for IOAM, and an associated set of rules that restrict the scope and features of the IOAM specification, thereby limiting it to a subset of the full functionality. IOAM profiles are defined in [I-D.mizrahi-ippm-ioam-profile].
9. IANA Considerations

This document does not request any IANA actions.

10. Security Considerations

As discussed in [RFC7276], a successful attack on an OAM protocol in general, and specifically on IOAM, can prevent the detection of failures or anomalies, or create a false illusion of nonexistent ones.

The Proof of Transit Option-Type (Section Section 4.2) is used for verifying the path of data packets. The security considerations of POT are further discussed in [I-D.ietf-sfc-proof-of-transit].

Security considerations related to the use of IOAM flags, in particular the loopback flag are found in [I-D.ietf-ippm-ioam-flags].

IOAM data can be subject to eavesdropping. Although the confidentiality of the user data is not at risk in this context, the IOAM data elements can be used for network reconnaissance, allowing attackers to collect information about network paths, performance, queue states, buffer occupancy and other information. Recon is an improbable security threat in an IOAM deployment that is within a confined physical domain. However, in deployments that are not confined to a single LAN, but span multiple inter-connected sites (for example, using an overlay network), the inter-site links can be secured (e.g., by IPsec) in order to avoid external eavesdropping. Another possible mitigation approach is to use the "direct exporting" mode [I-D.ietf-ippm-ioam-direct-export]. In this case the IOAM related trace information would not be available in the customer data packets, but would trigger exporting of (secured) packet-related IOAM information at every node. IOAM data export and securing IOAM data export is outside the scope of this document.

IOAM can be used as a means for implementing Denial of Service (DoS) attacks, or for amplifying them. For example, a malicious attacker can add an IOAM header to packets or modify an IOAM header in en route packets in order to consume the resources of network devices that take part in IOAM or collectors that analyze the IOAM data. Another example is a packet length attack, in which an attacker pushes headers associated with IOAM Option-Types into data packets, causing these packets to be increased beyond the MTU size, resulting in fragmentation or in packet drops. Such DoS attacks can be mitigated by deploying IOAM in confined administrative domains, and by defining performance limits on IOAM encapsulation and IOAM exporting. By limiting the rate and/or percentage of packets that
are subject to IOAM encapsulation and the rate of exported traffic, it is possible to confine the impact of such attacks.

Since IOAM options may include timestamps, if network devices use synchronization protocols then any attack on the time protocol [RFC7384] can compromise the integrity of the timestamp-related data fields. Synchronization attacks can be mitigated by combining a secured time distribution scheme, e.g., [RFC8915], and by using redundant clock sources [RFC5905] and/or redundant network paths for the time distribution protocol [RFC8039].

At the management plane, attacks may be implemented by misconfiguring or by maliciously configuring IOAM-enabled nodes in a way that enables other attacks. Thus, IOAM configuration should be secured in a way that authenticates authorized users and verifies the integrity of configuration procedures.

Notably, IOAM is expected to be deployed in specific network domains, thus confining the potential attack vectors to within the network domain. Indeed, in order to limit the scope of threats to within the current network domain the network operator is expected to enforce policies that prevent IOAM traffic from leaking outside of the IOAM domain, and prevent IOAM data from outside the domain to be processed and used within the domain. Note that the Immediate Export mode (reference to be added in a future revision) can mitigate the potential threat of IOAM data leaking through data packets.

11. Acknowledgements

The authors would like to thank Tal Mizrahi, Eric Vyncke, Nalini Elkins, Srihari Raghavan, Ranganathan T S, Barak Gafni, Karthik Babu Harichandra Babu, Akshaya Nadahalli, LJ Wobker, Erik Nordmark, Vengada Prasad Govindan, Andrew Yourtchenko, Aviv Kfir, Tianran Zhou, Zhenbin (Robin), Joe Clarke, Al Morton, Tom Herbet, Haoyu song, and Mickey Spiegel for the comments and advice on IOAM.

12. References

12.1. Normative References


12.2. Informative References

[I-D.ali-spring-ioam-srv6]
Ali, Z., Gandhi, R., Filsfils, C., Brockners, F., Nainar, N., Pignataro, C., Li, C., Chen, M., and G. Dawra,
"Segment Routing Header encapsulation for In-situ OAM Data", draft-ali-spring-ioam-srv6-05 (work in progress),
January 2022.

[I-D.brockners-ippm-ioam-geneve]
Brockners, F., Bhandari, S., Govindan, V. P., Pignataro, C., Nainar, N. K., Gredler, H., Leddy, J., Youell, S.,
Mizrahi, T., Lapukhov, P., Gafni, B., Kfir, A., and M. Spiegel, "Geneve encapsulation for In-situ OAM Data",
draft-brockners-ippm-ioam-geneve-05 (work in progress), November 2020.

[I-D.brockners-ippm-ioam-vxlan-gpe]
Brockners, F., Bhandari, S., Govindan, V. P., Pignataro, C., Gredler, H., Leddy, J., Youell, S., Mizrahi, T.,
Kfir, A., Gafni, B., Lapukhov, P., and M. Spiegel, "VXLAN-GPE Encapsulation for In-situ OAM Data",
draft-brockners-ippm-ioam-vxlan-gpe-03 (work in progress), November 2019.

[I-D.gandhi-spring-ioam-sr-mpls]
Encapsulation for In-situ OAM Data", draft-gandhi-spring-ioam-sr-mpls-02 (work in progress), August 2019.

[I-D.ietf-ippm-ioam-data]
Brockners, F., Bhandari, S., and T. Mizrahi, "Data Fields for In-situ OAM", draft-ietf-ippm-ioam-data-17 (work in
progress), December 2021.

[I-D.ietf-ippm-ioam-direct-export]
Song, H., Gafni, B., Zhou, T., Li, Z., Brockners, F., Bhandari, S., Sivakolundu, R., and T. Mizrahi, "In-situ

[I-D.ietf-ippm-ioam-flags]
Mizrahi, T., Brockners, F., Bhandari, S., Sivakolundu, R., Pignataro, C., Kfir, A., Gafni, B., Spiegel, M., and J.
Lemon, "In-situ OAM Loopback and Active Flags", draft-ietf-ippm-ioam-flags-07 (work in progress), October 2021.
[I-D.ietf-ippm-ioam-ipv6-options]
Bhandari, S. and F. Brockners, "In-situ OAM IPv6 Options",
draft-ietf-ippm-ioam-ipv6-options-07 (work in progress),
February 2022.

[I-D.ietf-nvo3-geneve]
Gross, J., Ganga, I., and T. Sridhar, "Geneve: Generic
Network Virtualization Encapsulation", draft-ietf-
nvo3-geneve-16 (work in progress), March 2020.

[I-D.ietf-nvo3-vxlan-gpe]
(Editor), F. M., (editor), L. K., and U. E. (editor),
"Generic Protocol Extension for VXLAN (VXLAN-GPE)", draft-
ietf-nvo3-vxlan-gpe-12 (work in progress), September 2021.

[I-D.ietf-sfc-ioam-nsh]
Brockners, F. and S. Bhandari, "Network Service Header
(NSH) Encapsulation for In-situ OAM (IOAM) Data",
draft-ietf-sfc-ioam-nsh-08 (work in progress), April 2022.

[I-D.ietf-sfc-proof-of-transit]
Brockners, F., Bhandari, S., Mizrahi, T., Dara, S., and S.
Youell, "Proof of Transit", draft-ietf-sfc-proof-of-
transit-08 (work in progress), November 2020.

[I-D.mizrahi-ippm-ioam-profile]
Mizrahi, T., Brockners, F., Bhandari, S., Sivakolundu, R.,
Pignataro, C., Kfir, A., Gafni, B., Spiegel, M., and T.
Zhou, "In Situ OAM Profiles", draft-mizrahi-ippm-ioam-
profile-06 (work in progress), February 2022.

[I-D.spiegel-ippm-ioam-rawexport]
Spiegel, M., Brockners, F., Bhandari, S., and R.
Sivakolundu, "In-situ OAM raw data export with IPFIX",
draft-spiegel-ippm-ioam-rawexport-06 (work in progress),
February 2022.

[I-D.weis-ippm-ioam-eth]
Weis, B., Brockners, F., Hill, C., Bhandari, S., Govindan,
V. P., Pignataro, C., Nainar, N. K., Gredler, H., Leddy,
J., Youell, S., Mizrahi, T., Kfir, A., Gafni, B.,
Lapukhov, P., and M. Spiegel, "EtherType Protocol
Identification of In-situ OAM Data", draft-weis-ippm-ioam-
eth-05 (work in progress), February 2022.


Authors’ Addresses

Frank Brockners (editor)
Cisco Systems, Inc.
Hansaallee 249, 3rd Floor
DUESSELDORF, NORDRHEIN-WESTFALEN  40549
Germany

Email: fbrockne@cisco.com

Shwetha Bhandari (editor)
Thoughtspot
3rd Floor, Indiqube Orion, 24th Main Rd, Garden Layout, HSR Layout
Bangalore, KARNATAKA 560 102
India

Email: shwetha.bhandari@thoughtspot.com

Daniel Bernier
Bell Canada
Canada

Email: daniel.bernier@bell.ca

Tal Mizrahi (editor)
Huawei
8-2 Matam
Haifa  3190501
Israel

Email: tal.mizrahi.phd@gmail.com
Abstract

In-situ Operations, Administration, and Maintenance (IOAM) records operational and telemetry information in user packets while the packets traverse a path between two points in the network. This document defines a YANG module for the IOAM function.

Status of This Memo

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

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

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

This Internet-Draft will expire on 8 January 2023.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

In-situ Operations, Administration, and Maintenance (IOAM) [I-D.ietf-ippm-ioam-data] records OAM information within user packets while the packets traverse a network. The data types and data formats for IOAM data records have been defined in [I-D.ietf-ippm-ioam-data]. The IOAM data can be embedded in many protocol encapsulations such as Network Services Header (NSH) and IPv6.

This document defines a data model for IOAM capabilities using the YANG data modeling language [RFC7950]. This YANG model supports five IOAM options, which are:

* Incremental Tracing Option [I-D.ietf-ippm-ioam-data]
* Pre-allocated Tracing Option [I-D.ietf-ippm-ioam-data]
* Direct Export Option [I-D.ietf-ippm-ioam-direct-export]
* Proof of Transit (PoT) Option [I-D.ietf-ippm-ioam-data]
* Edge-to-Edge Option [I-D.ietf-ippm-ioam-data]

2. Conventions used in this document

The keywords "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 BCP14, [RFC2119], [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are defined in [RFC7950] and are used in this specification:

* augment
* data model
* data node

The terminology for describing YANG data models is found in [RFC7950].

2.1. Tree Diagrams

Tree diagrams used in this document follow the notation defined in [RFC8340].

3. Design of the IOAM YANG Data Model

3.1. Overview

The IOAM model is organized as list of profiles as shown in the following figure. Each profile associates with one flow and the corresponding IOAM information.

The "ioam-info" is a container for all the read only assistant information, so that monitoring systems can interpret the IOAM data.
module: ietf-ioam
  +--rw ioam
    +--ro ioam-info
      +--ro timestamp-type? identityref
      +--ro available-interface* [if-name]
        |  +--ro if-name if:interface-ref
    +--rw ioam-profiles
    +--rw admin-config
      +--rw enabled? boolean
    +--rw ioam-profile* [profile-name]
      +--rw profile-name string
      +--rw filter
        |  +--rw filter-type? ioam-filter-type
        |  +--rw ace-name? -> acl:acls/acl/aces/ace/name
        +--rw protocol-type? ioam-protocol-type
      +--rw incremental-tracing-profile {incremental-trace}?
        |  ...
      +--rw preallocated-tracing-profile {preallocated-trace}?
        |  ...
      +--rw direct-export-profile {direct-export}?
        |  ...
      +--rw pot-profile {proof-of-transit}?
        |  ...
      +--rw e2e-profile {edge-to-edge}?
        ...

In the "ioam-profiles", the "enabled" is an administrative configuration. When it is set to true, IOAM configuration is enabled for the system. Meanwhile, the IOAM data-plane functionality is enabled.

The "filter" is used to identify a flow, where the IOAM profile can apply. There may be multiple filter types. ACL [RFC8519] is a common way to specify a flow. Each IOAM profile can associate with an ACE (Access Control Entry). IOAM actions MUST be driven by the accepted packets, when the matched ACE "forwarding" action is "accept".

The IOAM data can be encapsulated into multiple protocols, e.g., IPv6 [I-D.ietf-ippm-ioam-ipv6-options] and NSH [I-D.ietf-sfc-ioam-nsh]. The "protocol-type" is used to indicate where the IOAM is applied. For example, if the "protocol-type" is IPv6, the IOAM ingress node will encapsulate the associated flow with the IPv6-IOAM [I-D.ietf-ippm-ioam-ipv6-options] format.

IOAM data includes five encapsulation types, i.e., incremental tracing data, preallocated tracing data, direct export data, proof of transit data and end to end data. In practice, multiple IOAM data
types can be encapsulated into the same IOAM header. The "ioam-profile" contains a set of sub-profiles, each of which relates to one encapsulation type. The configured object may not support all the sub-profiles. The supported sub-profiles are indicated by 5 defined features, i.e., "incremental-trace", "preallocated-trace", "direct export", "proof-of-transit", "edge-to-edge".

3.2. Preallocated Tracing Profile

The IOAM tracing data is expected to be collected at every node that a packet traverses to ensure visibility into the entire path a packet takes within an IOAM domain. The preallocated tracing option will create pre-allocated space for each node to populate its information. The "preallocated-tracing-profile" contains the detailed information for the preallocated tracing data. The information includes:

* enabled: indicates whether the preallocated tracing profile is enabled.

* node-action: indicates the operation (e.g., encapsulate IOAM header, transit the IOAM data, or decapsulate IOAM header) applied to the dedicated flow.

* use-namespace: indicate the namespace used for the trace types.

* trace-type: indicates the per-hop data to be captured by the IOAM enabled nodes and included in the node data list.

* Loopback mode is used to send a copy of a packet back towards the source.

* Active mode indicates that a packet is used for active measurement.

```yaml
---rw preallocated-tracing-profile {preallocated-trace}?
  ---rw enabled?        boolean
  ---rw node-action?    ioam-node-action
  ---rw trace-types
    | ---rw use-namespace?  ioam-namespace
    | ---rw trace-type*    ioam-trace-type
  ---rw enable-loopback-mode? boolean
  ---rw enable-active-mode? boolean
```
3.3. Incremental Tracing Profile

The incremental tracing option contains a variable node data fields where each node allocates and pushes its node data immediately following the option header. The "incremental-tracing-profile" contains the detailed information for the incremental tracing data. The detailed information is the same as the Preallocated Tracing Profile, but with one more variable, "max-length", which restricts the length of the IOAM header.

```
  +--rw incremental-tracing-profile {incremental-trace}?
    +--rw enabled?                boolean
    +--rw node-action?            ioam-node-action
    +--rw trace-types
      |  +--rw use-namespace?   ioam-namespace
      |    +--rw trace-type*   ioam-trace-type
    +--rw enable-loopback-mode?   boolean
    +--rw enable-active-mode?     boolean
    +--rw max-length?             uint32
```

3.4. Direct Export Profile

The direct export option is used as a trigger for IOAM nodes to export IOAM data to a receiving entity (or entities). The "direct-export-profile" contains the detailed information for the direct export data. The detailed information is the same as the Preallocated Tracing Profile, but with one more optional variable, "flow-id", which is used to correlate the exported data of the same flow from multiple nodes and from multiple packets.

```
  +--rw direct-export-profile {direct-export}?
    +--rw enabled?                boolean
    +--rw node-action?            ioam-node-action
    +--rw trace-types
      |  +--rw use-namespace?   ioam-namespace
      |    +--rw trace-type*   ioam-trace-type
    +--rw enable-loopback-mode?   boolean
    +--rw enable-active-mode?     boolean
    +--rw flow-id?               uint32
```

3.5. Proof of Transit Profile

The IOAM Proof of Transit data is to support the path or service function chain verification use cases. The "pot-profile" contains the detailed information for the proof of transit data. "pot-type" indicates a particular POT variant that specifies the POT data that is included. There may be several POT types, which have different configuration data. To align with [I-D.ietf-ippm-ioam-data], this
document only defines IOAM POT type 0. User need to augment this module for the configuration of a specific POT type.

```yang
++-rw pot-profile {proof-of-transit}?
    +-rw enabled? boolean
    +-rw pot-type? ioam-pot-type
```

3.6. Edge to Edge Profile

The IOAM edge to edge option is to carry data that is added by the IOAM encapsulating node and interpreted by IOAM decapsulating node. The "e2e-profile" contains the detailed information for the edge to edge data. The detailed information includes:

* enabled: indicates whether the edge to edge profile is enabled.
* node-action is the same semantic as in Section 2.2.
* use-namespace: indicate the namespace used for the edge to edge types.
* e2e-type indicates data to be carried from the ingress IOAM node to the egress IOAM node.

```yang
++-rw e2e-profile {edge-to-edge}?
    +-rw enabled? boolean
    +-rw node-action? ioam-node-action
    +-rw e2e-types
        +-rw use-namespace? ioam-namespace
        +-rw e2e-type* ioam-e2e-type
```

4. IOAM YANG Module

<CODE BEGINS> file "ietf-ioam@2022-07-07.yang"
module ietf-ioam {
    yang-version 1.1;
    prefix "ioam";

    import ietf-access-control-list {
        prefix "acl";
        reference
            "RFC 8519: YANG Data Model for Network Access Control Lists (ACLs)";
    }

    import ietf-interfaces {
```
prefix "if";
reference
  "RFC 8343: A YANG Data Model for Interface Management";
}

import ietf-lime-time-types {
  prefix "lime";
  reference
    "RFC 8532: Generic YANG Data Model for the Management of Operations, Administration, and Maintenance (OAM) Protocols That Use Connectionless Communications";
}

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

contact
  "WG Web: <https://datatracker.ietf.org/wg/ippm>
  WG List: <ippm@ietf.org>
  Editor: zhoutianran@huawei.com
  Editor: james.n.guichard@futurewei.com
  Editor: fbrockne@cisco.com
  Editor: srihari@cisco.com";

description
  "This YANG module specifies a vendor-independent data model for the In Situ OAM (IOAM).

Copyright (c) 2021 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info).

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

revision 2022-07-07 {
  description "First revision.";
  reference "RFC XXXX: A YANG Data Model for In-Situ OAM";
}

/*
 * FEATURES

feature incremental-trace
{
    description
        "This feature indicated that the incremental tracing option is supported";
    reference "RFC 9197: Data Fields for In-situ OAM";
}

feature preallocated-trace
{
    description
        "This feature indicated that the preallocated tracing option is supported";
    reference "RFC 9197: Data Fields for In-situ OAM";
}

feature direct-export
{
    description
        "This feature indicated that the direct export option is supported";
    reference "RFC XXXX: In-situ OAM Direct Exporting";
}

feature proof-of-transit
{
    description
        "This feature indicated that the proof of transit option is supported";
    reference "RFC 9197: Data Fields for In-situ OAM";
}

feature edge-to-edge
{
    description
        "This feature indicated that the edge to edge option is supported";
    reference "RFC 9197: Data Fields for In-situ OAM";
}

/*
* IDENTITIES
*/

identity filter {
    description
        "Base identity to represent a filter. A filter is used to
specify the flow to apply the IOAM profile. ";
}

identity acl-filter {
  base filter;
  description
    "Apply ACL rules to specify the flow."
}

identity protocol {
  description
    "Base identity to represent the carrier protocol. It’s used to
    indicate what layer and protocol the IOAM data is embedded.";
}

identity ipv6 {
  base protocol;
  description
    "The described IOAM data is embedded in IPv6 protocol.";
    reference "RFC XXXX: In-situ OAM IPv6 Options";
}

identity nsh {
  base protocol;
  description
    "The described IOAM data is embedded in NSH.";
    reference
      "RFC XXXX: Network Service Header (NSH) Encapsulation
       for In-situ OAM (IOAM) Data";
}

identity node-action {
  description
    "Base identity to represent the node actions. It’s used to
    indicate what action the node will take.";
}

identity action-encapsulate {
  base node-action;
  description
    "It indicates the node is to encapsulate the IOAM packet";
}

identity action-decapsulate {
  base node-action;
  description
    "It indicates the node is to decapsulate the IOAM packet";
}
identity trace-type {
  description
    "Base identity to represent trace types.";
}

identity trace-hop-lim-node-id {
  base trace-type;
  description
    "It indicates the presence of Hop_Lim and node_id in the
    node data.";
}

identity trace-if-id {
  base trace-type;
  description
    "It indicates presence of ingress_if_id and egress_if_id
    (short format) in the node data.";
}

identity trace-timestamp-seconds {
  base trace-type;
  description
    "It indicates presence of timestamp seconds in the node data.";
}

identity trace-timestamp-fraction {
  base trace-type;
  description
    "It indicates presence of timestamp fraction in the node data.";
}

identity trace-transit-delay {
  base trace-type;
  description
    "It indicates presence of transit delay in the node data.";
}

identity trace-namespace-data {
  base trace-type;
  description
    "It indicates presence of name space specific data (short format)
    in the node data.";
}

identity trace-queue-depth {
  base trace-type;
  description
    "It indicates presence of queue depth in the node data.";
}
identity trace-checksum-complement {
  base trace-type;
  description
      "It indicates presence of the Checksum Complement node data.";
}

identity trace-hop-lim-node-id-wide {
  base trace-type;
  description
      "It indicates presence of Hop_Lim and node_id in wide format
       in the node data.";
}

identity trace-if-id-wide {
  base trace-type;
  description
      "It indicates presence of ingress_if_id and egress_if_id in
       wide format in the node data.";
}

identity trace-namespace-data-wide {
  base trace-type;
  description
      "It indicates presence of IOAM-Namespace specific data in wide
       format in the node data.";
}

identity trace-buffer-occupancy {
  base trace-type;
  description
      "It indicates presence of buffer occupancy in the node data.";
}

identity trace-opaque-state-snapshot {
  base trace-type;
  description
      "It indicates presence of variable length Opaque State Snapshot
       field.";
}

identity pot-type {
  description
      "Base identity to represent Proof of Transit (PoT) types.";
}

identity pot-type-0 {
base pot-type;
description
"The IOAM POT Type field value is 0, and POT data is a 16
Octet field to carry data associated to POT procedures.";
}

identity e2e-type {
  description
  "Base identity to represent e2e types";
}

identity e2e-seq-num-64 {
  base e2e-type;
  description
  "It indicates presence of a 64-bit sequence number.";
}

identity e2e-seq-num-32 {
  base e2e-type;
  description
  "It indicates the presence of a 32-bit sequence number.";
}

identity e2e-timestamp-seconds {
  base e2e-type;
  description
  "It indicates the presence of timestamp seconds representing
  the time at which the packet entered the IOAM-domain";
}

identity e2e-timestamp-fraction {
  base e2e-type;
  description
  "It indicates the presence of timestamp fraction representing
  the time at which the packet entered the IOAM-domain";
}

identity namespace {
  description
  "Base identity to represent the Namespace-ID.";
}

identity default-namespace {
  base namespace;
  description
  "The Namespace-ID value of 0x0000 is defined as the
  Default-Namespace-ID and must be known to all the nodes
  implementing IOAM.";
}
typedef ioam-filter-type {
    type identityref {
        base filter;
    }
    description "It specifies a known type of filter.";
}

typedef ioam-protocol-type {
    type identityref {
        base protocol;
    }
    description "It specifies a known type of carrier protocol for the IOAM data.";
}

typedef ioam-node-action {
    type identityref {
        base node-action;
    }
    description "It specifies a known type of node action.";
}

typedef ioam-trace-type {
    type identityref {
        base trace-type;
    }
    description "It specifies a known trace type.";
}

typedef ioam-pot-type {
    type identityref {
        base pot-type;
    }
    description "It specifies a known pot type.";
}

typedef ioam-e2e-type {
    type identityref {
        base e2e-type;
    }
    description "It specifies a known e2e type.";
}
typedef ioam-namespace {  
type identityref {  
  base namespace;  
}  
description  
"It specifies the supported namespace.";  
}

/*
 * GROUP DEFINITIONS
 */

grouping ioam-filter {  
description "A grouping for IOAM filter definition";  

leaf filter-type {  
type ioam-filter-type;  
description "filter type";  
}  

leaf ace-name {  
  when "derived-from-or-self(../filter-type, 'ioam:acl-filter')";  
type leafref {  
    path "/acl:acls/acl:acl/acl:aces/acl:ace/acl:name";  
}  
description "The Access Control Entry name is used to refer to an ACL specification.";  
}

}  
grouping encap-tracing {  
description  
"A grouping for the generic configuration for tracing profile.";

container trace-types {  
description  
"It indicates the list of trace types for encapsulation";

leaf use-namespace {  
type ioam-namespace;  
description
"It indicates the name space used for encapsulation";
}

leaf-list trace-type {
    type ioam-trace-type;
    description
        "The trace type is only defined at the encapsulation node."
}

leaf enable-loopback-mode {
    type boolean;
    default false;
    description
        "Loopback mode is used to send a copy of a packet back towards
        the source. The loopback mode is only defined at the
        encapsulation node."
}

leaf enable-active-mode {
    type boolean;
    default false;
    description
        "Active mode indicates that a packet is used for active
        measurement. An IOAM decapsulating node that receives a
        packet with the Active flag set in one of its Trace options
        must terminate the packet."
}

grouping ioam-incremental-tracing-profile {
    description
        "A grouping for incremental tracing profile."

    leaf node-action {
        type ioam-node-action;
        description
            "This object indicates the action the node need to
            take, e.g. encapsulation."
    }

    uses encap-tracing {
        when "derived-from-or-self(node-action,
            'ioam:action-encapsulate')"
    }

    leaf max-length {
        when "derived-from-or-self(../node-action,
'ioam:action-encapsulate');

type uint32;
units bytes;
description
"This field specifies the maximum length of the node data list
in octets. The max-length is only defined at the
encapsulation node, and it's only used for the incremental
tracing mode."
}
}


grouping ioam-preallocated-tracing-profile {
  description
  "A grouping for incremental tracing profile.";

  leaf node-action {
    type ioam-node-action;
    description "This indicates what action the node will take,
e.g. encapsulation.";
  }

  uses encap-tracing {
    when "derived-from-or-self(node-action,
      'ioam:action-encapsulate')";
  }
}


grouping ioam-direct-export-profile {
  description
  "A grouping for direct export profile.";

  leaf node-action {
    type ioam-node-action;
    description "This indicates what action the node will take,
e.g. encapsulation.";
  }

  uses encap-tracing {
    when "derived-from-or-self(node-action,
      'ioam:action-encapsulate')";
  }

  leaf flow-id {
    when "derived-from-or-self(/node-action,
      'ioam:action-encapsulate')";
    type uint32;
    description
    "This field specifies the max-length of the node data list
    in octets. The max-length is only defined at the
    encapsulation node, and it’s only used for the incremental
    tracing mode."
  }
}
"A 32-bit flow identifier. The field is set at the encapsulating node. The Flow ID can be uniformly assigned by a central controller or algorithmically generated by the encapsulating node. The latter approach cannot guarantee the uniqueness of Flow ID, yet the conflict probability is small due to the large Flow ID space. flow-id is used to correlate the exported data of the same flow from multiple nodes and from multiple packets."
}
)
)

grouping ioam-e2e-profile {
    description
        "A grouping for edge-to-edge profile."
;
leaf node-action {
    type ioam-node-action;
    description
        "It indicates how the node acts for this profile";
}

container e2e-types {
    when "derived-from-or-self(/node-action,
        'ioam:action-encapsulate')"
;
    description
        "It indicates the list of e2e types for encapsulation"
;
leaf use-namespace {
    type ioam-namespace;
    description
        "It indicates the name space used for encapsulation"
;
}

leaf-list e2e-type {
    type ioam-e2e-type;
    description
        "The e2e type is only defined at the encapsulation node."
;
}
)

grouping ioam-admin-config {
    description
        "IOAM top-level administrative configuration."
;
leaf enabled {
    type boolean;
default false;

description
  "This object is to control the availability of configuration. It must be true before anything in the
  /ioam/ioam-profiles/ioam-profile subtree can be edited. If false, any configuration in place is not used.";
}
}

/*
* DATA NODES
*/

container ioam {
  description "IOAM top level container";
}

container ioam-info {
  config false;
  description
    "Describes assistant information such as units or timestamp format. So that monitoring systems can interpret the IOAM data.";

  leaf timestamp-type {
    type identityref {
      base lime:timestamp-type;
    }
    description
      "Type of timestamp, such as Truncated PTP or NTP."
  }
}

list available-interface {
  key "if-name";
  description
    "A list of available interfaces that support IOAM.";
  leaf if-name {
    type if:interface-ref;
    description "This is a reference to the Interface name.";
  }
}

container ioam-profiles {
  description
    "Contains a list of IOAM profiles.";
}

container admin-config {
  description
"Contains all the administrative configurations related to
the IOAM functionalities and all the IOAM profiles."

uses ioam-admin-config;
}

list ioam-profile {
  key "profile-name";
  description
    "A list of IOAM profiles that configured on the node.
    There is no mandatory type of profile (e.g.,
    incremental-trace, preallocated-trace.) in the list.
    But at least one profile should be added.";

  leaf profile-name {
    type string{
      length "1 .. max";
    }
    description
      "Unique identifier for each IOAM profile";
  }

  container filter {
    uses ioam-filter;
    description
      "The filter which is used to indicate the flow to apply
      IOAM.";
  }

  leaf protocol-type {
    type ioam-protocol-type;
    description
      "This item is used to indicate the carrier protocol where
      the IOAM is applied.";
  }

  container incremental-tracing-profile {
    if-feature incremental-trace;
    description
      "It describes the profile for incremental tracing option";

    leaf enabled {
      type boolean;
      default false;
      description
        "When true, apply incremental tracing option to the
        specified flow identified by the filter.";
    }
  }

uses ioam-incremental-tracing-profile;
}

container preallocated-tracing-profile {
  if-feature preallocated-trace;
  description
  "It describes the profile for preallocated tracing option";

  leaf enabled {
    type boolean;
    default false;
    description
    "When true, apply preallocated tracing option to the
     specified flow identified by the following filter.";
  }
}

uses ioam-preallocated-tracing-profile;
}

container direct-export-profile {
  if-feature direct-export;
  description
  "It describes the profile for direct-export option";

  leaf enabled {
    type boolean;
    default false;
    description
    "When true, apply direct-export option to the
     specified flow identified by the following filter.";
  }
}

uses ioam-direct-export-profile;
}

container pot-profile {
  if-feature proof-of-transit;
  description
  "It describes the profile for PoT option";

  leaf enabled {
    type boolean;
    default false;
    description
    "When true, apply Proof of Transit option to the
     specified flow identified by the following filter.";
  }
}
leaf pot-type {
    type ioam-pot-type;
    description
        "The type of a particular POT variant that specifies
        the POT data that is included.";
}

container e2e-profile {
    if-feature edge-to-edge;
    description
        "It describes the profile for e2e option";

    leaf enabled {
        type boolean;
        default false;
        description
            "When true, apply edge-to-edge option to the
             specified flow identified by the following filter.";
    }

    uses ioam-e2e-profile;
}

5. Security Considerations

The YANG module specified in this document defines a schema for data
that is designed to be accessed via network management protocols such
as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer
is the secure transport layer, and the mandatory-to-implement secure
transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer
is HTTPS, and the mandatory-to-implement secure transport is TLS
[RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341]
provides the means to restrict access for particular NETCONF or
RESTCONF users to a preconfigured subset of all available NETCONF or
RESTCONF protocol operations and content.

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

* /ioam/ioam-profiles/admin-config

The items in the container above include the top level administrative configurations related to the IOAM functionalities and all the IOAM profiles. Unexpected changes to these items could lead to the IOAM function disruption and/or misbehavior of all the IOAM profiles.

* /ioam/ioam-profiles/ioam-profile

The entries in the list above include the whole IOAM profile configurations which indirectly create or modify the device configurations. Unexpected changes to these entries could lead to the mistake of the IOAM behavior for the corresponding flows.

6. IANA Considerations

RFC Ed.: In this section, replace all occurrences of ’XXXX’ with the actual RFC number (and remove this note).

IANA is requested to assign a new URI from the IETF XML Registry [RFC3688]. The following URI is suggested:

Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

This document also requests a new YANG module name in the YANG Module Names registry [RFC7950] with the following suggestion:

name: ietf-ioam
prefix: ioam
reference: RFC XXXX

7. Acknowledgements

For their valuable comments, discussions, and feedback, we wish to acknowledge Greg Mirsky, Reshad Rahman, Tom Petch and Mickey Spiegel.

8. References

8.1. Normative References


8.2. Informative References


Appendix A. Examples

This appendix is non-normative.

tbd

Authors’ Addresses

Tianran Zhou
Huawei
156 Beiqing Rd.
Abstract

For many years, a lack of responsiveness, variously called lag, latency, or bufferbloat, has been recognized as an unfortunate, but common, symptom in today’s networks. Even after a decade of work on standardizing technical solutions, it remains a common problem for the end users.

Everyone "knows" that it is "normal" for a video conference to have problems when somebody else at home is watching a 4K movie or uploading photos from their phone. However, there is no technical reason for this to be the case. In fact, various queue management solutions (fq_codel, cake, PIE) have solved the problem.

Our networks remain unresponsive, not from a lack of technical solutions, but rather a lack of awareness of the problem and its solutions. We believe that creating a tool whose measurement matches people’s everyday experience will create the necessary awareness, and result in a demand for products that solve the problem.

This document specifies the "RPM Test" for measuring responsiveness. It uses common protocols and mechanisms to measure user experience specifically when the network is under working conditions. The measurement is expressed as "Round-trips Per Minute" (RPM) and should be included with throughput (up and down) and idle latency as critical indicators of network quality.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 12, 2023.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction ........................................... 3
1.1. Terminology ......................................... 3
2. Design Constraints .................................... 4
3. Goals ................................................... 6
4. Measuring Responsiveness Under Working Conditions .... 6
4.1. Working Conditions .................................. 6
4.1.1. From single-flow to multi-flow ................... 7
4.1.2. Parallel vs Sequential Uplink and Downlink ....... 7
4.1.3. Reaching full link utilization .................. 8
4.1.4. Final "Working Conditions" Algorithm ............ 8
4.2. Measuring Responsiveness ............................ 10
4.2.1. Aggregating the Measurements .................... 11
5. Interpreting responsiveness results ................... 11
5.1. Elements influencing responsiveness ................ 11
5.1.1. Client side influence ............................ 12
5.1.2. Network influence ................................ 12
5.1.3. Server side influence ............................ 13
5.2. Root-causing Responsiveness ......................... 13
6. RPM Test Server API .................................... 13
7. RPM Test Server Discovery ............................. 15
8. Security Considerations ................................ 16
For many years, a lack of responsiveness, variously called lag, latency, or bufferbloat, has been recognized as an unfortunate, but common, symptom in today’s networks [Bufferbloat]. Solutions like fq_codel [RFC8290] or PIE [RFC8033] have been standardized and are to some extent widely implemented. Nevertheless, people still suffer from bufferbloat.

Although significant, the impact on user experience can be transitory — that is, its effect is not always visible to the user. Whenever a network is actively being used at its full capacity, buffers can fill up and create latency for traffic. The duration of those full buffers may be brief: a medium-sized file transfer, like an email attachment or uploading photos, can create bursts of latency spikes. An example of this is lag occurring during a videoconference, where a connection is briefly shown as unstable.

These short-lived disruptions make it hard to narrow down the cause. We believe that it is necessary to create a standardized way to measure and express responsiveness.

Existing network measurement tools could incorporate a responsiveness measurement into their set of metrics. Doing so would also raise the awareness of the problem and would help establish a new expectation that the standard measures of network quality should — in addition to throughput and idle latency — also include latency under load, or, as we prefer to call it, responsiveness under working conditions.

1.1. Terminology

A word about the term "bufferbloat" — the undesirable latency that comes from a router or other network equipment buffering too much data. This document uses the term as a general description of bad latency, using more precise wording where warranted.

"Latency" is a poor measure of responsiveness, since it can be hard for the general public to understand. The units are unfamiliar ("What is a millisecond?") and counterintuitive ("100 msec — that sounds good — it’s only a tenth of a second!").
Instead, we create the term "Responsiveness under working conditions" to make it clear that we are measuring all, not just idle, conditions, and use "round-trips per minute" as the metric. The advantage of round-trips per minute are two-fold: First, it allows for a metric that is "the higher the better". This kind of metric is often more intuitive for end-users. Second, the range of the values tends to be around the 4-digit integer range which is also a value easy to compare and read, again allowing for a more intuitive use. Finally, we abbreviate the measurement to "RPM", a wink to the "revolutions per minute" that we use for car engines.

This document defines an algorithm for the "RPM Test" that explicitly measures responsiveness under working conditions.

2.  Design Constraints

There are many challenges around measurements on the Internet. They include the dynamic nature of the Internet, the diverse nature of the traffic, the large number of devices that affect traffic, and the difficulty of attaining appropriate measurement conditions.

Internet paths are changing all the time. Daily fluctuations in the demand make the bottlenecks ebb and flow. To minimize the variability of routing changes, it’s best to keep the test duration relatively short.

TCP and UDP traffic, or traffic on ports 80 and 443, may take significantly different paths on the Internet and be subject to entirely different Quality of Service (QoS) treatment. A good test will use standard transport-layer traffic - typical for people’s use of the network - that is subject to the transport’s congestion control that might reduce the traffic’s rate and thus its buffering in the network.

Traditionally, one thinks of bufferbloat happening on the routers and switches of the Internet. However, the networking stacks of the clients and servers can have huge buffers. Data sitting in TCP sockets or waiting for the application to send or read causes artificial latency, and affects user experience the same way as "traditional" bufferbloat.

Finally, it is crucial to recognize that significant queueing only happens on entry to the lowest-capacity (or "bottleneck") hop on a network path. For any flow of data between two communicating devices, there is always one hop along the path where the capacity available to that flow at that hop is the lowest among all the hops of that flow’s path at that moment in time. It is important to understand that the existence of a lowest-capacity hop on a network
path is not itself a problem. In a heterogeneous network like the Internet it is inevitable that there must necessarily be some hop along the path with the lowest capacity for that path. If that hop were to be improved to make it no longer the lowest-capacity hop, then some other hop would become the new lowest-capacity hop for that path. In this context a "bottleneck" should not be seen as a problem to be fixed, because any attempt to "fix" the bottleneck is futile - such a "fix" can never remove the existence of a bottleneck on a path; it just moves the bottleneck somewhere else. Arguably, this heterogeneity of the Internet is one of its greatest strengths. Allowing individual technologies to evolve and improve at their own pace, without requiring the entire Internet to change in lock-step, has enabled enormous improvements over the years in technologies like DSL, cable modems, Ethernet, and Wi-Fi, each advancing independently as new developments became ready. As a result of this flexibility we have moved incrementally, one step at a time, from 56kb/s dial-up modems in the 1990s to Gb/s home Internet service and Gb/s wireless connectivity today.

Note that in a shared datagram network, conditions do not remain static. The hop that is the current bottleneck may change from moment to moment. For example, changes in other traffic may result in changes to a flow’s share of a given hop. A user moving around may cause the Wi-Fi transmission rate to vary widely, from a few Mb/s when far from the Access Point, all the way up to Gb/s or more when close to the Access Point.

Consequently, if we wish to enjoy the benefits of the Internet’s great flexibility, we need software that embraces and celebrates this diversity and adapts intelligently to the varying conditions it encounters.

Because significant queueing only happens on entry to the bottleneck hop, the queue management at this critical hop of the path almost entirely determines the responsiveness of the entire flow. If the bottleneck hop’s queue management algorithm allows an excessively large queue to form, this results in excessively large delays for packets sitting in that queue awaiting transmission, significantly degrading overall user experience.

In order to discover the depth of the buffer at the bottleneck hop, the RPM Test mimics normal network operations and data transfers, to cause this bottleneck buffer to fill to capacity, and then measures the resulting end-to-end latency under these operating conditions. A well managed bottleneck queue keeps its queue occupancy under control, resulting in consistently low round-trip time and consistently good responsiveness. A poorly managed bottleneck queue will not.
3. Goals

The algorithm described here defines an RPM Test that serves as a good proxy for user experience. This means:

1. Today's Internet traffic primarily uses HTTP/2 over TLS. Thus, the algorithm should use that protocol.

   As a side note: other types of traffic are gaining in popularity (HTTP/3) and/or are already being used widely (RTP). Traffic prioritization and QoS rules on the Internet may subject traffic to completely different paths: these could also be measured separately.

2. The Internet is marked by the deployment of countless middleboxes like transparent TCP proxies or traffic prioritization for certain types of traffic. The RPM Test must take into account their effect on TCP-handshake [RFC0793], TLS-handshake, and request/response.

3. The test result should be expressed in an intuitive, nontechnical form.

4. Finally, to be useful to a wide audience, the measurement should finish within a short time frame. Our target is 20 seconds.

4. Measuring Responsiveness Under Working Conditions

To make an accurate measurement, the algorithm must reliably put the network in a state that represents those "working conditions". During this process, the algorithm measures the responsiveness of the network. The following explains how the former and the latter are achieved.

4.1. Working Conditions

There are many different ways to define the state of "working conditions" to measure responsiveness. There is no one true answer to this question. It is a tradeoff between using realistic traffic patterns and pushing the network to its limits.

In this document we aim to generate a realistic traffic pattern by using standard HTTP transactions but exploring the worst-case scenario by creating multiple of these transactions and using very large data objects in these HTTP transactions.

This allows to create a stable state of working conditions during which the network is used at its nearly full capacity, without
generating DoS-like traffic patterns (e.g., intentional UDP flooding). This creates a realistic traffic mix representative of what a typical user’s network experiences in normal operation.

Finally, as end-user usage of the network evolves to newer protocols and congestion control algorithms, it is important that the working conditions also can evolve to continuously represent a realistic traffic pattern.

4.1.1. From single-flow to multi-flow

A single TCP connection may not be sufficient to reach the capacity of a path quickly. Using a 4MB receive window, over a network with a 32 ms round-trip time, a single TCP connection can achieve up to 1Gb/s throughput. For higher throughput and/or networks with higher round-trip time, TCP allows larger receive window sizes, up to 1 GB. For most applications there is little reason to open multiple parallel TCP connections in an attempt to achieve higher throughput.

However, it may take some time for a single TCP connection to ramp up to full speed, and one of the goals of the RPM test is to quickly load the network to capacity, take its measurements, and then finish. Additionally, traditional loss-based TCP congestion control algorithms react aggressively to packet loss by reducing the congestion window. This reaction (intended by the protocol design) decreases the queueing within the network, making it harder to determine the depth of the bottleneck queue reliably.

The purpose of the RPM Test is not to productively move data across the network in a useful way, the way a normal application does. The purpose of the RPM Test is, as quickly as possible, to simulate a representative traffic load as if real applications were doing sustained data transfers, measure the resulting round-trip time occurring under those realistic conditions, and then end the test. Because of this, using multiple simultaneous parallel connections allows the RPM test to complete its task more quickly, in a way that overall is less disruptive and less wasteful of network capacity than a test using a single TCP connection that would take longer to bring the bottleneck hop to a stable saturated state.

4.1.2. Parallel vs Sequential Uplink and Downlink

Poor responsiveness can be caused by queues in either (or both) the upstream and the downstream direction. Furthermore, both paths may differ significantly due to access link conditions (e.g., 5G downstream and LTE upstream) or the routing changes within the ISPs. To measure responsiveness under working conditions, the algorithm must explore both directions.
One approach could be to measure responsiveness in the uplink and downlink in parallel. It would allow for a shorter test run-time.

However, a number of caveats come with measuring in parallel:

- Half-duplex links may not permit simultaneous uplink and downlink traffic. This means the test might not reach the path’s capacity in both directions at once and thus not expose all the potential sources of low responsiveness.

- Debuggability of the results becomes harder: During parallel measurement it is impossible to differentiate whether the observed latency happens in the uplink or the downlink direction.

Thus, we recommend testing uplink and downlink sequentially. Parallel testing is considered a future extension.

4.1.3. Reaching full link utilization

The RPM Test gradually increases the number of TCP connections and measures "goodput" - the sum of actual data transferred across all connections in a unit of time. When the goodput stops increasing, it means that the network is used at its full capacity. At this point we are creating the worst-case scenario within the limits of the realistic traffic pattern.

The algorithm notes that throughput increases rapidly until TCP connections complete their TCP slow-start phase. At that point, throughput eventually stalls, often due to receive window limitations, particularly in cases of high network bandwidth, high network round-trip time, low receive window size, or a combination of all three. The only means to further increase throughput is by adding more TCP connections to the pool of load-generating connections. If new connections leave the throughput the same, full link utilization has been reached and - more importantly - the working condition is stable.

4.1.4. Final "Working Conditions" Algorithm

The following algorithm reaches working conditions of a network by using HTTP/2 upload (POST) or download (GET) requests of infinitely large files. The algorithm is the same for upload and download and uses the same term "load-generating connection" for each. The actions of the algorithm take place at regular intervals. For the current draft the interval is defined as one second.

Where
o i: The index of the current interval. The variable i is initialized to 0 when the algorithm begins and increases by one for each interval.

o instantaneous aggregate goodput at interval p: The number of total bytes of data transferred within interval p, divided by the interval duration. If p is negative (i.e., a time interval logically prior to the start of the test beginning, used in moving average calculations), the number of total bytes of data transferred within that interval is considered to be 0.

o moving average aggregate goodput at interval p: The number of total bytes of data transferred within interval p and the three immediately preceding intervals, divided by four times the interval duration.

o moving average stability during the period between intervals b and e: Whether or not, for all b<=x<e, the absolute difference is less than 5% between the moving average aggregate goodput at interval x and the moving average aggregate goodput at interval x+1. If all absolute differences are below 5% then the moving average has achieved stability. If any of the absolute differences are 5% or more then the moving average has not achieved stability.

the steps of the algorithm are:

o Create four load-generating connections.

o At each interval:
  * Compute the instantaneous aggregate goodput at interval i.
  * Compute the moving average aggregate goodput at interval i.
  * If the moving average aggregate goodput at interval i is more than a 5% increase over the moving average aggregate goodput at interval i - 1, the network has not yet reached full link utilization.
    + If no load-generating connections have been added within the last four intervals, add four more load-generating connections.
  * Else, the network has reached full link utilization with the existing load-generating connections. The current state is a candidate for stable working conditions.
+ If a) there have been load-generating connections added in the past four intervals and b) there has been moving average stability during the period between intervals i-4 and i, then the network has reached full link utilization and the algorithm terminates.

+ Otherwise, add four more load-generating connections.

In Section 3, it is mentioned that one of the goals is that the test finishes within 20 seconds. It is left to the implementation what to do when full link utilization is not reached within that time-frame. For example, an implementation might gather a provisional responsiveness measurement or let the test run for longer.

4.2. Measuring Responsiveness

Measuring responsiveness during the previously explained working conditions creation is a continuous process during the duration of the test. It requires a sufficiently large sample-size to have confidence in the results.

The measurement of the responsiveness happens by sending probe-requests for a small object. The probe requests are being sent in two ways:

1. A HTTP GET request on a separate connection. This test mimics the time it takes for a web browser to connect to a new web server and request the first element of a web page (e.g., "index.html"), or the startup time for a video streaming client to launch and begin fetching media.

2. A HTTP GET request multiplexed on the load-generating connections. This test mimics the time it takes for a video streaming client to skip ahead to a different chapter in the same video stream, or for a navigation client to react and fetch new map tiles when the user scrolls the map to view a different area. In a well functioning system fetching new data over an existing connection should take less time than creating a brand new TLS connection from scratch to do the same thing.

The former will provide 3 set of data-points. First, the duration of the TCP-handshake (noted hereafter as tcp_foreign). Second, the TLS round-trip-time (noted tls_foreign). For this, it is important to note that different TLS versions have a different number of round-trips. Thus, the TLS establishment time needs to be normalized to the number of round-trips the TLS handshake takes until the connection is ready to transmit data. And third, the HTTP latency
between issuing the GET request for a 1-byte object until the entire response has been received (noted http_foreign).

The latter will provide a single data-point between the time the HTTP GET request for the 1-byte object is issued on the load-generating connection until the full HTTP response has been received (noted http_self).

It is important to issue multiple of these probes. To have a large dataset, the methodology requires a client to issue these probes every 100 milli-seconds. For the probes on the load-generating connections, the client needs to use one of the initial load-generating connections. This means that every 100ms, 2 probes are being evaluated. The total amount of data used for these probes would be no more than about 50KB worth of data within one second.

4.2.1. Aggregating the Measurements

The algorithm produces sets of 4 times for each probe, namely: tcp_foreign, tls_foreign, http_foreign, http_self (fromm the previous section). Each of these sets will have a large number of sample. To aggregate the methodology proposes the following:

Among each set, we take the 90th percentile, thus resulting in 4 individual numbers. To aggregate these individual numbers into a single responsiveness number, we suggest the following weighted mean:

\[
\text{Responsiveness} = \frac{60000}{((1/3 \times \text{tcp}_\text{foreign} + 1/3 \times \text{tls}_\text{foreign} + 1/3 \times \text{http}_\text{foreign} + \text{http}_\text{self})/2)}
\]

This responsiveness value presents round-trips per minute (RPM).

5. Interpreting responsiveness results

The described methodology uses a high-level approach to measure responsiveness. By executing the test with regular HTTP requests a number of elements come into play that will influence the result. Contrary to more traditional measurement methods the responsiveness metric is not only influenced by the properties of the network but can significantly be influenced by the properties of the client and the server implementations. This section describes how the different elements influence responsiveness and how a user may differentiate them when debugging a network.

5.1. Elements influencing responsiveness

Due to the HTTP-centric approach of the measurement methodology a number of factors come into play that influence the results. Namely, the client-side networking stack (from the top of the HTTP-layer all
the way down to the physical layer), the network (including potential transparent HTTP "accelerators"), and the server-side networking stack. The following outlines how each of these contributes to the responsiveness.

5.1.1. Client side influence

As the driver of the measurement, the client-side networking stack can have a large influence on the result. The biggest influence of the client comes when measuring the responsiveness in the uplink direction. Load-generation will cause queue-buildup in the transport layer as well as the HTTP layer. Additionally, if the network’s bottleneck is on the first hop, queue-buildup will happen at the layers below the transport stack (e.g., NIC firmware).

Each of these queue build-ups may cause latency and thus low responsiveness. A well designed networking stack would ensure that queue-buildup in the TCP layer is kept at a bare minimum with solutions like TCP_NOTSENT_LOWAT [draft-ietf-tcpm-rfc793bis]. At the HTTP/2 layer it is important that the load-generating data is not interfering with the latency-measuring probes. For example, the different streams should not be stacked one after the other but rather be allowed to be multiplexed for optimal latency. The queue-buildup at these layers would only influence latency on the probes that are sent on the load-generating connections.

Below the transport layer many places have a potential queue build-up. It is important to keep these queues at reasonable sizes or that they implement techniques like FQ-Codel. Depending on the techniques used at these layers, the queue build-up can influence latency on probes sent on load-generating connections as well as separate connections. If flow-queuing is used at these layers, the impact on separate connections will be negligible.

5.1.2. Network influence

The network obviously is a large driver for the responsiveness result. Propagation delay from the client to the server as well as queuing in the bottleneck node will cause latency. Beyond these traditional sources of latency, other factors may influence the results as well. Many networks deploy transparent TCP Proxies, firewalls doing deep packet-inspection, HTTP "accelerators",... As the methodology relies on the use of HTTP/2, the responsiveness metric will be influenced by such devices as well.

The network will influence both kinds of latency probes that the responsiveness tests sends out. Depending on the network’s use of Smart Queue Management and whether this includes flow-queuing or not,
the latency probes on the load-generating connections may be influenced differently than the probes on the separate connections.

5.1.3. Server side influence

Finally, the server-side introduces the same kind of influence on the responsiveness as the client-side, with the difference that the responsiveness will be impacted during the downlink load generation.

5.2. Root-causing Responsiveness

Once an RPM result has been generated one might be tempted to try to localize the source of a potential low responsiveness. The responsiveness measurement is however aimed at providing a quick, top-level view of the responsiveness under working conditions the way end-users experience it. Localizing the source of low responsiveness involves however a set of different tools and methodologies.

Nevertheless, the responsiveness test allows to gain some insight into what the source of the latency is. The previous section described the elements that influence the responsiveness. From there it became apparent that the latency measured on the load-generating connections and the latency measured on separate connections may be different due to the different elements.

For example, if the latency measured on separate connections is much less than the latency measured on the load-generating connections, it is possible to narrow down the source of the additional latency on the load-generating connections. As long as the other elements of the network don’t do flow-queueing, the additional latency must come from the queue build-up at the HTTP and TCP layer. This is because all other bottlenecks in the network that may cause a queue build-up will be affecting the load-generating connections as well as the separate latency probing connections in the same way.

6. RPM Test Server API

The RPM measurement is built upon a foundation of standard protocols: IP, TCP, TLS, HTTP/2. On top of this foundation, a minimal amount of new "protocol" is defined, merely specifying the URLs that used for GET and PUT in the process of executing the test.

Both the client and the server MUST support HTTP/2 over TLS. The client MUST be able to send a GET request and a POST. The server MUST be able to respond to both of these HTTP commands. The server MUST have the ability to provide content upon a GET request. Both client and server SHOULD use loss-based congestion controls like Cubic. The server MUST use a packet scheduling algorithm that
minimizes internal queueing to avoid affecting the client’s measurement.

The server MUST respond to 4 URLs:

1. A "small" URL/response: The server must respond with a status code of 200 and 1 byte in the body. The actual message content is irrelevant. The server SHOULD specify the content-type as application/octet-stream. The server SHOULD minimize the size, in bytes, of the response fields that are encoded and sent on the wire.

2. A "large" URL/response: The server must respond with a status code of 200 and a body size of at least 8GB. The server SHOULD specify the content-type as application/octet-stream. The body can be bigger, and may need to grow as network speeds increases over time. The actual message content is irrelevant. The client will probably never completely download the object, but will instead close the connection after reaching working condition and making its measurements.

3. An "upload" URL/response: The server must handle a POST request with an arbitrary body size. The server should discard the payload. The actual POST message content is irrelevant. The client will probably never completely upload the object, but will instead close the connection after reaching working condition and making its measurements.

4. A configuration URL that returns a JSON [RFC8259] object with the information the client uses to run the test (sample below). The server SHOULD specify the content-type as application/json.

Sample JSON:

```json
{
  "version": 1,
  "urls": {
    "large_https_download_url": "https://nq.example.com/api/v1/large",
    "small_https_download_url": "https://nq.example.com/api/v1/small",
    "https_upload_url": "https://nq.example.com/api/v1/upload"
  }
  "test_endpoint": "hostname123.provider.com"
}
```

All of the fields in the sample configuration are required except "test_endpoint". If the test server provider can pin all of the requests for a test run to a specific host in the service (for a particular run), they can specify that host name in the "test_endpoint" field.
The client begins the responsiveness measurement by querying for the JSON configuration. This supplies the URLs for creating the load-generating connections in the upstream and downstream direction as well as the small object for the latency measurements.

7. RPM Test Server Discovery

It makes sense to host RPM Test Server instances in Internet Data Centers where they can be accessed easily by users wishing to test the quality of their Internet connection. However, when a user performs an RPM test and determines that they are suffering from poor RPM during download, the logical next question might be, "What’s causing my poor performance? Is it poor buffer management by my ISP? Is it poor buffer management in my home Wi-Fi Access point? Something else?"

To help an end user answer this question, it will be useful for home gateway equipment to host RPM Test Server instances. In an example configuration, a user may have cable modem service offering 100 Mb/s download speed, connected via gigabit Ethernet to one or more Wi-Fi access points in the home, which then offer service to Wi-Fi client devices at different rates depending on distance, interference from other traffic, etc. By having the cable modem itself host an RPM Test Server instance, the user can then run a test between the cable modem and their computer or smartphone, to help isolate whether bufferbloat they are experiencing is occurring in equipment inside the home (like their Wi-Fi access points) or somewhere outside the home.

To aid in discoverability of these facilities, local RPM Test Server instances SHOULD advertise the availability of service type [RFC6335] "_nq._tcp" (Network Quality), via DNS-Based Service Discovery [RFC6763], using Multicast DNS on its local link(s) [RFC6762]. Where applicable, an RPM Test Server instance SHOULD also advertise the availability of its service via unicast discovery, for discovery by client devices not directly attached to the same link. Population of the appropriate DNS zone with the relevant unicast discovery records can be performed automatically using a Discovery Proxy [RFC8766], or in some scenarios simply by having a human administrator manually enter the required records. Similarly, a "cloud" service, providing Internet hosting service for "example.com" could choose to include the relevant DNS-SD records within the "example.com" domain [RFC6763] to communicate to clients the list of available RPM Test Server instances.
8. Security Considerations

TBD

9. IANA Considerations

IANA has been requested to record the service type ".nq._tcp" (Network Quality) for advertising and discovery of RPM Test Server instances.

10. Acknowledgments

We would like to thank Rich Brown for his editorial pass over this I-D. We also thank Erik Auerswald and Will Hawkins for their constructive feedback on the I-D.

11. Informative References

[Bufferbloat]

[draft-ietf-tcpm-rfc793bis]

[RFC793]

[RFC6335]

[RFC6762]

[RFC6763]
Appendix A.  Example Server Configuration

This section shows fragments of sample server configurations to host an responsiveness measurement endpoint.

A.1.  Apache Traffic Server

Apache Traffic Server starting at version 9.1.0 supports configuration as a responsiveness server. It requires the generator and the statichit plugin.

The sample remap configuration file then is:

```text

```
map https://nq.example.com/api/v1/config \ 
http://localhost/ \ 
@plugin=statichit.so \ 
@pparam=--file-path=config.example.com.json \ 
@pparam=--mime-type=application/json

map https://nq.example.com/api/v1/large \ 
http://localhost/cache/8589934592/ \ 
@plugin=generator.so

map https://nq.example.com/api/v1/small \ 
http://localhost/cache/1/ \ 
@plugin=generator.so

map https://nq.example.com/api/v1/upload \ 
http://localhost/ \ 
@plugin=generator.so

Authors’ Addresses

Christoph Paasch
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: cpaasch@apple.com

Randall Meyer
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: rrm@apple.com

Stuart Cheshire
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: cheshire@apple.com
Omer Shapira  
Apple Inc.  
One Apple Park Way  
Cupertino, California 95014  
United States of America  

Email: oesh@apple.com

Matt Mathis  
Google, Inc  
1600 Amphitheatre Parkway  
Mountain View, CA 94043  
United States of America

Email: mattmathis@google.com
Simple TWAMP (STAMP) Extensions for Segment Routing Networks
draft-ietf-ippm-stamp-srpm-04

Abstract

Segment Routing (SR) leverages the source routing paradigm. SR is applicable to both Multiprotocol Label Switching (SR-MPLS) and IPv6 (SRv6) forwarding planes. This document specifies RFC 8762 (Simple Two-Way Active Measurement Protocol (STAMP)) extensions for SR networks, for both SR-MPLS and SRv6 forwarding planes by augmenting the optional extensions defined in RFC 8972.

Status of This Memo

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

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

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

This Internet-Draft will expire on 6 January 2023.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

Segment Routing (SR) leverages the source routing paradigm for Software Defined Networks (SDNs). SR is applicable to both Multiprotocol Label Switching (SR-MPLS) and IPv6 (SRv6) forwarding planes [RFC8402]. SR Policies as defined in [I-D.ietf-spring-segment-routing-policy] are used to steer traffic through a specific, user-defined paths using a stack of Segments. A comprehensive SR Performance Measurement (PM) toolset is one of the essential requirements to measure network performance to provide Service Level Agreements (SLAs).
The Simple Two-Way Active Measurement Protocol (STAMP) provides capabilities for the measurement of various performance metrics in IP networks [RFC8762] without the use of a control channel to pre-signal session parameters. [RFC8972] defines optional extensions, in the form of TLVs, for STAMP. Note that the YANG data model defined in [I-D.ietf-ippm-stamp-yang] can be used to provision the STAMP Session-Sender and STAMP Session-Reflector.

The STAMP test packets are transmitted along an IP path between a Session-Sender and a Session-Reflector to measure performance delay and packet loss along that IP path. It may be desired in SR networks that the same path (same set of links and nodes) between the Session-Sender and Session-Reflector is used for the STAMP test packets in both directions. This is achieved by using the STAMP [RFC8762] extensions for SR-MPLS and SRv6 networks specified in this document by augmenting the optional extensions defined in [RFC8972].

2. Conventions Used in This Document

2.1. 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 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.2. Abbreviations

MPLS: Multiprotocol Label Switching.

PM: Performance Measurement.

SID: Segment ID.

SL: Segment List.

SR: Segment Routing.

SR-MPLS: Segment Routing with MPLS forwarding plane.

SRv6: Segment Routing with IPv6 forwarding plane.

SSID: STAMP Session Identifier.

2.3. Reference Topology

In the reference topology shown below, the STAMP Session-Sender S1 initiates a STAMP test packet and the STAMP Session-Reflector R1 transmits a reply STAMP test packet. The reply test packet may be transmitted to the Session-Sender S1 on the same path (same set of links and nodes) or a different path in the reverse direction from the path taken towards the Session-Reflector R1.

The nodes S1 and R1 may be connected via a link or an SR path [RFC8402]. The link may be a physical interface, virtual link, or Link Aggregation Group (LAG) [IEEE802.1AX], or LAG member. The SR path may be an SR Policy [I-D.ietf-spring-segment-routing-policy] on node S1 (called head-end) with destination to node R1 (called tail-end).

```
+-------+     Test Packet     +-------+  
|       | - - - - - - - - - ->|       |
|   S1  |=====================|   R1  |
|       |<- - - - - - - - - -|       |
+-------+  Reply Test Packet  +-------+  
            /                  \  
          T4                    T3  
```

STAMP Session-Sender  STAMP Session-Reflector

Reference Topology

3. Verification Check Flag in TLV

The STAMP TLV option in [RFC8972] defines the use of the 8-bit flags field common to all STAMP TLVs.

A one-bit flag called Verification Check (V) flag is defined at position (TBA3) in the flags field of the STAMP TLV. A Session-Sender MUST set the V flag to 0 before transmitting an extended STAMP test packet when reply test packet is required. A Session-Reflector MUST set the V flag to 1 for any STAMP TLV that it supports that includes a requirement that cannot be met or is in conflict with the Session-Reflector processing or capability. The V flag MUST be set to 0 by the Session-Reflector when the requirement from the request is met.

A Session-Sender MUST set the V flag to 1 before transmitting an extended STAMP test packet when test packet reply is not required. A Session-Reflector MUST NOT reply and MUST drop the test packet if the
Session-Reflector determined for any STAMP TLV that is supports that includes a requirement that cannot be met or is in conflict with the Session-Reflector processing or capability.

4. Destination Node Address TLV

The Session-Sender may need to transmit test packets to the Session-Reflector with a different destination address that is not matching an address on the Session-Reflector e.g. when the STAMP test packet is encapsulated by a tunneling protocol or an MPLS Segment List with destination IPv4 address from 127/8 range or Segment Routing Header (SRH) with destination IPv6 address ::1/128. When using IPv4 destination address from 127/8 range (e.g. for testing ECMPs), the STAMP test packet may not reach the intended Session-Reflector in an error condition, and an un-intended node may transmit reply test packet resulting in reporting of invalid measurement metrics.

[RFC8972] defines STAMP test packets that can include one or more optional TLVs. In this document, Destination Node Address TLV (Type TBA1) is defined for STAMP test packet [RFC8972] and has the following format shown in Figure 1:

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|STAMP TLV Flags| Type=TBA1     |         Length                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
.                           Address                             .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: Destination Node Address TLV Format

The Length field is used to decide the Address Family of the Address.

The STAMP TLV Flags are set using the procedures described in [RFC8972].

The Destination Node Address TLV is optional. The Destination Node Address TLV indicates the address of the intended Session-Reflector node of the test packet. The Session-Reflector MUST add the received Destination Node Address TLV in the reply test packet to ensure the symmetric reply test packet size and to transmit the STAMP TLV Flags to the Session-Sender.

A Session-Sender MUST set the V flag to 0 before transmitting an extended STAMP test packet when test packet reply is required. A Session-Reflector that supports this TLV, MUST set the V flag in the reply test packet to 1 if the Session-Reflector determined that it is
not the intended Destination as identified in the Destination Node Address TLV. Otherwise, the Session-Reflector MUST set the V flag in the reply test packet to 0.

A Session-Sender MUST set the V flag to 1 before transmitting an extended STAMP test packet when test packet reply is not required. A Session-Reflector that supports this TLV, MUST NOT reply and MUST drop the test packet if the Session-Reflector determined that it is not the intended Destination as identified in the Destination Node Address TLV.

5. Return Path TLV

For end-to-end SR paths, the Session-Reflector may need to transmit the reply test packet on a specific return path. The Session-Sender can request this in the test packet to the Session-Reflector using a Return Path TLV. With this TLV carried in the Session-Sender test packet, signaling and maintaining dynamic SR network state for the STAMP sessions on the Session-Reflector are avoided.

For links, the Session-Reflector may need to transmit the reply test packet on the same incoming link in the reverse direction. The Session-Sender can request this in the test packet to the Session-Reflector using a Return Path TLV.

[RFC8972] defines STAMP test packets that can include one or more optional TLVs. In this document, the TLV Type (value TBA2) is defined for the Return Path TLV that carries the return path for the Session-Sender test packet. The format of the Return Path TLV is shown in Figure 2:

<table>
<thead>
<tr>
<th>STAMP TLV Flags</th>
<th>Type=TBA2</th>
<th>Length</th>
<th>Return Path Sub-TLVs</th>
</tr>
</thead>
</table>
+------------------+-+------------------+-+------------------+-+------------------+
| 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 |
| +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| | STAMP TLV Flags | Type=TBA2 | Length |
| +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| | Return Path Sub-TLVs |
| . |
| +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Figure 2: Return Path TLV

The STAMP TLV Flags are set using the procedures described in [RFC8972].

The Return Path TLV is optional. The Session-Sender MUST only insert one Return Path TLV in the STAMP test packet. The Session-Reflector that supports this TLV, MUST only process the first Return Path TLV.
in the test packet and ignore other Return Path TLVs if present, and it MUST add the received Return Path TLV (including all Sub-TLVs) in the reply test packet to ensure the symmetric reply test packet size and to transmit the STAMP TLV Flags to the Session-Sender. The Session-Reflector that supports this TLV MUST reply using the Return Path received in the Session-Sender test packet. In the case where the Session-Reflector does not support this TLV, the procedure defined in [RFC8762] is followed by the Session-Reflector.

A Session-Sender MUST set the V flag to 0 before transmitting an extended STAMP test packet when test packet reply is required. A Session-Reflector that supports this TLV, MUST set the V flag in the reply test packet to 1 if the Session-Reflector determined that it cannot use the return path in the test packet to transmit the reply test packet. Otherwise, the Session-Reflector MUST set the V flag in the reply test packet to 0.

A Session-Sender MUST set the V flag to 1 before transmitting an extended STAMP test packet when test packet reply is not required. A Session-Reflector that supports this TLV, MUST NOT reply and MUST drop the test packet if the Session-Reflector determined that it cannot use the return path in the test packet to transmit the reply test packet.

5.1. Return Path Sub-TLVs

The Return Path TLV contains one or more Sub-TLVs to carry the information for the requested return path. A Return Path Sub-TLV can carry Return Path Control Code, Return Path IP Address or Return Path Segment List.

The STAMP Sub-TLV Flags are set using the procedures described in [RFC8972].

When Return Path Sub-TLV is present in the Session-Sender test packet, the Session-Reflector that supports this TLV, MUST transmit reply test packet using the return path information specified in the Return Path Sub-TLV.

A Return Path TLV MUST NOT contain both Control Code Sub-TLV as well as Return Address or Return Segment List Sub-TLV.

5.1.1. Return Path Control Code Sub-TLV

The format of the Return Path Control Code Sub-TLV is shown in Figure 3. The Type of the Return Path Control Code Sub-TLV is defined as following:
* Type (value 1): Return Path Control Code. The Session-Sender can request the Session-Reflector to transmit the reply test packet based on the flags defined in the Control Code field.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|STAMP TLV Flags|   Type=1      |         Length                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Control Code                                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: Control Code Sub-TLV in Return Path TLV

Control Code Flags (32-bit): Defined as follows.

- 0x0: No Reply Requested.
- 0x1: Reply Requested on the Same Link.

When Control Code flag is set to 0x0 in the Session-Sender test packet, the Session-Reflector does not transmit reply test packet to the Session-Sender and terminates the STAMP test packet. Only the one-way measurement is applicable in this case. Optionally, the Session-Reflector may locally stream performance metrics via telemetry using the information from the received test packet. All other Return Path Sub-TLVs MUST be ignored in this case.

When Control Code flag is set to 0x1 in the Session-Sender test packet, the Session-Reflector transmits the reply test packet over the same incoming link where the test packet is received in the reverse direction towards the Session-Sender. All other Return Path Sub-TLVs MUST be ignored in this case.

5.1.2. Return Address Sub-TLV

The STAMP reply test packet may be transmitted to the Session-Sender to a different destination address on the Session-Sender using Return Path TLV. For this, the Session-Sender can specify in the test packet the receiving destination node address for the Session-Reflector reply test packet. When transmitting the STAMP test packet to a different destination address, the Session-Sender MUST follow the procedure defined in Section 4.3 of [RFC8762].
The format of the Return Address Sub-TLV is shown in Figure 4. The Address Family field indicates the type of the address, and it SHALL be set to one of the assigned values in the "IANA Address Family Numbers" registry. The Type of the Return Address Sub-TLV is defined as following:

* Type (value 2): Return Address. Destination node address of the Session-Reflector reply test packet different than the Source Address in the Session-Sender test packet.

```
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
  +---------------------------------------------+
  | STAMP TLV Flags|     Type=2   |         Length                   |
  +---------------------------------------------+
  |     Reserved                        | Address Family |
  +---------------------------------------------+
  |                                         .  Address                         |
  +---------------------------------------------+
```

Figure 4: Return Address Sub-TLV in Return Path TLV

5.1.3. Return Segment List Sub-TLVs

The format of the Segment List Sub-TLVs in the Return Path TLV is shown in Figures 5, 6, and 7. The segment entries MUST be in network order. The Segment List Sub-TLV can be one of the following Types:

* Type (value 3): SR-MPLS Label Stack of the Return Path

* Type (value 4): SRv6 Segment List of the Return Path

* Type (value 5): Structured SRv6 Segment List of the Return Path

```
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
  +---------------------------------------------+
  | STAMP TLV Flags|     Type=3   |         Length                   |
  +---------------------------------------------+
  |     Segment(1)                                  |
  +---------------------------------------------+
  |                                           .  Segment(n) (bottom of stack) |
  +---------------------------------------------+
```
Figure 5: SR-MPLS Segment List Sub-TLV in Return Path TLV

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|STAMP TLV Flags|     Type=4    |         Length                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Segment(1)

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

Segment(n) (bottom of stack)

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

Figure 6: SRv6 Segment List Sub-TLV in Return Path TLV

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|STAMP TLV Flags|     Type=5    |         Length                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    LB Length  |  LN Length    | Fun. Length   |  Arg. Length  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Segment(1)

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

Segment(n) (bottom of stack)

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

An SR-MPLS Label Stack Sub-TLV may carry only Binding SID [I-D.ietf-pce-binding-label-sid] of the Return SR-MPLS Policy.

An SRv6 Segment List Sub-TLV and Structured SRv6 Segment List Sub-TLV may carry only Binding SID [I-D.ietf-pce-binding-label-sid] of the Return SRv6 Policy.

A Structured SRv6 Segment List Sub-TLV is used to carry the structure and behavior for SRv6 SIDs [RFC8986] used in the Return SRv6 path as shown in Figure 7. The structure is intended for informational use by the control and management planes. The fields in the structure of the Sub-TLV are defined as follows [RFC8986]:

* LB Length: 1 octet. SRv6 SID Locator Block (LB) length in bits.
* LN Length: 1 octet. SRv6 SID Locator Node (LN) length in bits.
* Arg. Length: 1 octet. SRv6 SID Arguments length in bits.

In Structured SRv6 Segment List Sub-TLV, the sum of all four sizes MUST be less than or equal to 128 bits. If the sum of all four sizes is larger than 128 bits, the Sub-TLV MUST NOT be used by the Session-Reflector.

The Session-Sender MUST only insert one Segment List Return Path Sub-TLV in the test packet. The Session-Reflector MUST only process the first Segment List Return Path Sub-TLV in the test packet and ignore other Segment List Return Path Sub-TLVs if present.

Note that in addition to P2P SR paths, the Return Segment List Sub-TLV is also applicable to P2MP SR paths. For example, for P2MP SR paths, it may only carry the Node Segment Identifier of the Session-Sender in order for the reply test packet to follow an SR path to the Session-Sender.

6. Security Considerations

The usage of STAMP protocol is intended for deployment in limited domains [RFC8799]. As such, it assumes that a node involved in STAMP protocol operation has previously verified the integrity of the path and the identity of the far-end Session-Reflector.
If desired, attacks can be mitigated by performing basic validation and sanity checks, at the Session-Sender, of the timestamp fields in received reply test packets. The minimal state associated with these protocols also limits the extent of measurement disruption that can be caused by a corrupt or invalid test packet to a single test cycle.

The security considerations specified in [RFC8762] and [RFC8972] also apply to the extensions defined in this document. Specifically, the message integrity protection using HMAC, as defined in [RFC8762] Section 4.4, also apply to the procedure described in this document.

STAMP uses the well-known UDP port number that could become a target of denial of service (DoS) or could be used to aid man-in-the-middle (MITM) attacks. Thus, the security considerations and measures to mitigate the risk of the attack documented in Section 6 of [RFC8545] equally apply to the STAMP extensions in this document.

The STAMP extensions defined in this document may be used for potential "proxying" attacks. For example, a Session-Sender may specify a return path that has a destination different from that of the Session-Sender. But normally, such attacks will not happen in an SR domain where the Session-Senders and Session-Reflectors belong to the same domain. In order to prevent using the extension defined in this document for proxying any possible attacks, the return path has destination to the same node where the forward path is from. The Session-Reflector may drop the Session-Sender test packet when it cannot determine whether the Return Path has the destination to the Session-Sender. That means, the Session-Sender should choose a proper source address according to the specified Return Path to help the Session-Reflector to make that decision.

7. IANA Considerations

IANA has created the "STAMP TLV Types" registry for [RFC8972]. IANA is requested to allocate a value for the Destination Address TLV Type and a value for the Return Path TLV Type from the IETF Review TLV range of the same registry.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBA1</td>
<td>Destination Node Address TLV</td>
<td>This document</td>
</tr>
<tr>
<td>TBA2</td>
<td>Return Path TLV</td>
<td>This document</td>
</tr>
</tbody>
</table>

Table 1: STAMP TLV Types
NOTE: The following experimental values will be used until IANA early allocation is received:

- Destination Node Address TLV Value 240
- Return Path TLV Value 241

IANA is requested to create a sub-registry for "Return Path Sub-TLV Type". All code points in the range 1 through 175 in this registry shall be allocated according to the "IETF Review" procedure as specified in [RFC8126]. Code points in the range 176 through 239 in this registry shall be allocated according to the "First Come First Served" procedure as specified in [RFC8126]. Remaining code points are allocated according to Table 2:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 175</td>
<td>IETF Review</td>
<td>This document</td>
</tr>
<tr>
<td>176 - 239</td>
<td>First Come First Served</td>
<td>This document</td>
</tr>
<tr>
<td>240 - 251</td>
<td>Experimental Use</td>
<td>This document</td>
</tr>
<tr>
<td>252 - 254</td>
<td>Private Use</td>
<td>This document</td>
</tr>
</tbody>
</table>

Table 2: Return Path Sub-TLV Type Registry

IANA is requested to allocate the values for the following Sub-TLV Types from this registry.
Table 3: Return Path Sub-TLV Types

IANA has created the "STAMP TLV Flags" subregistry. IANA is requested to allocate the following bit position in the "STAMP TLV Flags" subregistry.

Table 4: STAMP TLV Flags

8. References

8.1. Normative References


8.2. Informative References


[I-D.ietf-pce-binding-label-sid]
Sivabalan, S., Filsfils, C., Tantsura, J., Previdi, S.,
and C. L. (editor), "Carrying Binding Label/Segment
Identifier in PCE-based Networks.", Work in Progress,
Internet-Draft, draft-ietf-pce-binding-label-sid-15, 20
March 2022, <https://www.ietf.org/archive/id/draft-ietf-
pce-binding-label-sid-15.txt>.

[I-D.ietf-ippm-stamp-yang]
Mirsky, G., Min, X., and W. S. Luo, "Simple Two-way Active
Measurement Protocol (STAMP) Data Model", Work in
Progress, Internet-Draft, draft-ietf-ippm-stamp-yang-09,
12 July 2021, <https://www.ietf.org/archive/id/draft-ietf-
ippm-stamp-yang-09.txt>.

[IEEE802.1AX]
IEEE Std. 802.1AX, "IEEE Standard for Local and
metropolitan area networks - Link Aggregation", November
2008.

Acknowledgments

The authors would like to thank Thierry Couture for the discussions
on the use-cases for Performance Measurement in Segment Routing. The
authors would also like to thank Greg Mirsky, Mike Koldychev, Gyan
Mishra, Tianran Zhou, Al Mortons, Reshad Rahman, Zhengliang Li, Frank
Brockners, and Cheng Li for providing comments and suggestions.

Authors’ Addresses

Rakesh Gandhi (editor)
Cisco Systems, Inc.
Canada
Email: rgandhi@cisco.com

Clarence Filsfils
Cisco Systems, Inc.
Email: cfilsfils@cisco.com

Daniel Voyer
Bell Canada
Email: daniel.voyer@bell.ca

Mach(Guoyi) Chen
Huawei

Gandhi, et al. Expires 6 January 2023
Simple Two-way Active Measurement Protocol (STAMP) Data Model
draft-ietf-ippm-stamp-yang-10

Abstract

This document specifies the data model for implementations of Session-Sender and Session-Reflector for Simple Two-way Active Measurement Protocol (STAMP) 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 https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on 11 January 2023.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

1. Introduction .................................................. 2
   1.1. Conventions used in this document .................... 2
       1.1.1. Requirements Language ......................... 3
2. Scope, Model, and Applicability .............................. 3
   2.1. Data Model Parameters ................................ 3
       2.1.1. STAMP-Sender .................................... 3
       2.1.2. STAMP-Reflector ................................ 4
3. Data Model .................................................... 4
   3.1. Tree Diagrams ............................................ 5
   3.2. YANG Module ............................................. 10
4. IANA Considerations .......................................... 34
5. Security Considerations ...................................... 34
6. Acknowledgments .............................................. 35
7. References .................................................... 35
   7.1. Normative References ................................... 35
   7.2. Informative References ................................ 36
Appendix A. Example of STAMP Session Configuration ............ 37
Authors’ Addresses .............................................. 38

1. Introduction

The Simple Two-way Active Measurement Protocol (STAMP) [RFC8762] 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 STAMP protocol
[RFC8762] in unauthenticated mode is on-wire compatible with TWAMP
Light, discussed in Appendix I [RFC5357]. 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. As one of the goals of STAMP is to support these
variations, this document presents their analysis; describes the data
model of the base STAMP specification. The defined STAMP data model
can be augmented to include STAMP extensions, for example, described
in [RFC8972]. This document defines the STAMP data model and
specifies it formally, using the YANG data modeling language
[RFC7950].

This version of the interfaces data model conforms to the Network
Management Datastore Architecture (NMDA) defined in [RFC8342].

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 a model of the STAMP as defined in [RFC8762] and Section 3 [RFC8972].

```
+----------------------+                +-------------------------+
| STAMP Session-Sender | <--- STAMP---> | STAMP Session-Reflector |
+----------------------+                +-------------------------+

Figure 1: STAMP Reference Model
```

2.1. Data Model Parameters

This section describes containers within the STAMP data model.

2.1.1. STAMP-Sender

The stamp-session-sender container holds items that are related to the configuration of the stamp Session-Sender logical entity.

The stamp-session-sender-state container holds information about the state of the particular STAMP test session.

RPCs stamp-sender-start and stamp-sender-stop respectively start and stop the referenced session by the stamp-session-id of the STAMP.
2.1.1.1. Controls for Test Session and Performance Metric Calculation

The data model supports several scenarios for a STAMP Session-Sender to execute test sessions and calculate performance metrics:

* The test mode in which the test packets are sent unbound in time as defined by the parameter ‘interval’ in the stamp-session-sender container frequency is referred to as continuous mode. Performance metrics in the continuous mode are calculated at a period defined by the parameter ‘measurement-interval’.

* The test mode that has a specific number of the test packets configured for the test session in the ‘number-of-packets’ parameter is referred to as a periodic mode. The STAMP-Sender MAY repeat the test session with the same parameters. The ‘repeat’ parameter defines the number of tests and the ‘repeat-interval’ - the interval between the consecutive tests. The performance metrics are calculated after each test session when the interval defined by the ‘session-timeout’ expires.

2.1.2. STAMP-Reflector

The stamp-session-reflector container holds items that are related to the configuration of the STAMP Session-Reflector logical entity.

The stamp-session-refl-state container holds Session-Reflector state data for the particular STAMP test session.

3. Data Model

Creating the STAMP data model presents several challenges, and among them is the identification of a test-session at Session-Reflector. A Session-Reflector MAY require only as little as the STAMP Session Identifier (SSID) and the source IP address in received STAMP-Test packet to spawn a 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 perform STAMP 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 the use of Class of Service TLV (Section 4.4 [RFC8972]) that will go undetected, but by using SSID and the source IP address as a key, we can ensure that STAMP test packets that are considered as different test sessions follow the same path even in ECMP environments.
3.1. Tree Diagrams

This section presents a simplified graphical representation of the STAMP data model using a YANG tree diagram [RFC8340].

module: ietf-stamp
  +--rw stamp
    |    +--rw stamp-session-sender {session-sender}?
    |         +--rw sender-enable? boolean
    |         +--rw test-session-enable? [stamp-session-id] boolean
    |         +--rw number-of_packets? union
    |         +--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 session_sender_udp_port inet:port-number
    |         +--rw stamp_session_id uint16
    |         +--rw session_reflector_ip inet:ip-address
    |         +--rw session_reflector_udp_port? inet:port-number
    |         +--rw sender_timeout? timestamp-format
    |         +--rw security! {stamp-security}?
    |             | +--rw key_chain? kc:key-chain-ref
    |             +--rw first_percentile? percentile
    |             +--rw second_percentile? percentile
    |             +--rw third_percentile? percentile
    +--rw stamp_session_reflector {session-reflector}?
        +--rw reflector_enable? boolean
        +--rw ref_wait? uint32
        +--rw reflector_mode_state? session-reflector-mode
        +--rw test_session-enable? [stamp-session-id] boolean
        +--rw stamp_session_id union
        +--rw dscp_handling_mode? session-dscp-mode
        +--rw dscp_value? inet:dscp
        +--rw sender_ip? union
        +--rw sender_udp_port? union
        +--rw reflector_ip? union
        +--rw reflector_udp_port? inet:port-number
        +--rw reflector_timeout? timestamp-format
        +--rw security! {stamp-security}?
            +--rw key_chain? kc:key-chain-ref
Figure 2: STAMP Configuration Tree Diagram

module: ietf-stamp
  +--ro stamp-state
  |  +--ro stamp-session-sender-state {session-sender}?
  |  |  +--ro test-session-state* [stamp-session-id]
  |  |  |  +--ro stamp-session-id              uint16
  |  |  +--ro sender-session-state?   enumeration
  |  +--ro current-stats
  |  |  +--ro start-time                    yang:date-and-time
  |  |  +--ro interval?                     uint32
  |  |  +--ro duplicate-packets?            uint32
  |  |  +--ro reordered-packets?            uint32
  |  |  +--ro sender-timestamp-format?      timestamp-format
  |  |  +--ro reflector-timestamp-format?   timestamp-format
  |  +--ro dscp?                         inet:dscp
  +--ro two-way-delay
  |  +--ro delay
  |  |  +--ro min?   yang:gauge64
  |  |  +--ro max?   yang:gauge64
  |  |  +--ro avg?   yang:gauge64
  |  +--ro delay-variation
  |  |  +--ro min?   yang:gauge32
  |  |  +--ro max?   yang:gauge32
  |  |  +--ro avg?   yang:gauge32
  +--ro one-way-delay-far-end
  |  +--ro delay
  |  |  +--ro min?   yang:gauge64
  |  |  +--ro max?   yang:gauge64
  |  |  +--ro avg?   yang:gauge64
  |  +--ro delay-variation
  |  |  +--ro min?   yang:gauge32
  |  |  +--ro max?   yang:gauge32
  |  |  +--ro avg?   yang:gauge32
  +--ro one-way-delay-near-end
  |  +--ro delay
  |  |  +--ro min?   yang:gauge64
  |  |  +--ro max?   yang:gauge64
  |  |  +--ro avg?   yang:gauge64
  |  +--ro delay-variation
  |  |  +--ro min?   yang:gauge32
  |  |  +--ro max?   yang:gauge32
  |  |  +--ro avg?   yang:gauge32
  +--ro low-percentile
  |  +--ro delay-percentile
  |  |  +--ro rtt-delay?        yang:gauge64
  |  |  +--ro near-end-delay?   yang:gauge64
|   | ro far-end-delay? yang:gauge64
|   | delay-variation-percentile
|   |   ro rtt-delay-variation? yang:gauge32
|   |   ro near-end-delay-variation? yang:gauge32
|   |   ro far-end-delay-variation? yang:gauge32
|   | mid-percentile
|   |   ro delay-percentile
|   |     ro rtt-delay? yang:gauge64
|   |     ro near-end-delay? yang:gauge64
|   |     ro far-end-delay? yang:gauge64
|   |   delay-variation-percentile
|   |     ro rtt-delay-variation? yang:gauge32
|   |     ro near-end-delay-variation? yang:gauge32
|   |     ro far-end-delay-variation? yang:gauge32
|   | high-percentile
|   |   ro delay-percentile
|   |     ro rtt-delay? yang:gauge64
|   |     ro near-end-delay? yang:gauge64
|   |     ro far-end-delay? yang:gauge64
|   |   delay-variation-percentile
|   |     ro rtt-delay-variation? yang:gauge32
|   |     ro near-end-delay-variation? yang:gauge32
|   |     ro far-end-delay-variation? yang:gauge32
|   | 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
|   | 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
|   | 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
|   | sender-ip inet:ip-address
|   | session-sender-udp-port inet:port-number
|   | session-reflector-ip inet:ip-address
|   | session-reflector-udp-port? inet:port-number
|   | sent-packets? uint32
|   | rcv-packets? uint32
|   | sent-packets-error? uint32

++-ro history-stats* [stamp-session-id]
  ++-ro stamp-session-id          uint16
  ++-ro end-time                  yang:date-and-time
  ++-ro interval?                 uint32
  ++-ro duplicate-packets?        uint32
  ++-ro reordered-packets?        uint32
  ++-ro sender-timestamp-format?  timestamp-format
  ++-ro reflector-timestamp-format?  timestamp-format
  ++-ro dscp?                     inet:dscp
  ++-ro two-way-delay
    ++-ro delay
      |  ++-ro min?   yang:gauge64
      |  ++-ro max?   yang:gauge64
      |  ++-ro avg?   yang:gauge64
      ++-ro delay-variation
        |  ++-ro min?   yang:gauge32
        |  ++-ro max?   yang:gauge32
        |  ++-ro avg?   yang:gauge32
  ++-ro one-way-delay-far-end
    ++-ro delay
      |  ++-ro min?   yang:gauge64
      |  ++-ro max?   yang:gauge64
      |  ++-ro avg?   yang:gauge64
      ++-ro delay-variation
        |  ++-ro min?   yang:gauge32
        |  ++-ro max?   yang:gauge32
        |  ++-ro avg?   yang:gauge32
  ++-ro one-way-delay-near-end
    ++-ro delay
      |  ++-ro min?   yang:gauge64
      |  ++-ro max?   yang:gauge64
      |  ++-ro avg?   yang:gauge64
      ++-ro delay-variation
        |  ++-ro min?   yang:gauge32
        |  ++-ro max?   yang:gauge32
        |  ++-ro avg?   yang:gauge32
++-ro low-percentile
  ++-ro delay-percentile
    |  ++-ro rtt-delay?    yang:gauge64
    |  ++-ro near-end-delay?  yang:gauge64
    |  ++-ro far-end-delay?  yang:gauge64
    ++-ro delay-variation-percentile
      |  ++-ro rtt-delay-variation?    yang:gauge32
      |  ++-ro near-end-delay-variation?  yang:gauge32
      |  ++-ro far-end-delay-variation?  yang:gauge32
++-ro mid-percentile
    +++-ro delay-percentile
        +++-ro rtt-delay?  yang:gauge64
        +++-ro near-end-delay?  yang:gauge64
        +++-ro far-end-delay?  yang:gauge64
    +++-ro delay-variation-percentile
        +++-ro rtt-delay-variation?  yang:gauge32
        +++-ro near-end-delay-variation?  yang:gauge32
        +++-ro far-end-delay-variation?  yang:gauge32
++-ro high-percentile
    +++-ro delay-percentile
        +++-ro rtt-delay?  yang:gauge64
        +++-ro near-end-delay?  yang:gauge64
        +++-ro far-end-delay?  yang:gauge64
    +++-ro delay-variation-percentile
        +++-ro rtt-delay-variation?  yang:gauge32
        +++-ro near-end-delay-variation?  yang:gauge32
        +++-ro far-end-delay-variation?  yang:gauge32
++-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 sender-ip  inet:ip-address
++-ro session-sender-udp-port  inet:port-number
++-ro session-reflector-ip  inet:ip-address
++-ro session-reflector-udp-port?  inet:port-number
++-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 stamp-session-refl-state {session-reflector}?
+++ro test-session-state* [stamp-session-id]
  +++ro stamp-session-id    uint16
  +++ro reflector-timestamp-format?   timestamp-format
  +++ro sender-ip         inet:ip-address
  +++ro session-sender-udp-port   inet:port-number
  +++ro session-reflector-ip       inet:ip-address
  +++ro session-reflector-udp-port?   inet:port-number
  +++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

Figure 3: STAMP State Tree Diagram

rpcs:
  +++x stamp-sender-start
   |  +++w input
   |  |  +++w stamp-session-id    uint16
  +++x stamp-sender-stop
   +++w input
   |  +++w stamp-stamp-session-id    uint16

Figure 4: STAMP RPC Tree Diagram

3.2. YANG Module

<CODE BEGINS> file "ietf-stamp@2022-06-15.yang"
module ietf-stamp {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-stamp";
  prefix "ietf-stamp";

  import ietf-inet-types {
    prefix inet;
    reference "RFC 6991: Common YANG Types.";
  } 
  import ietf-yang-types {
    prefix yang;
    reference "RFC 6991: Common YANG Types.";
  } 
  import ietf-key-chain {
    prefix kc;
    reference "RFC 8177: YANG Data Model for Key Chains.";
  } 

This YANG module specifies a vendor-independent model for the Simple Two-way Active Measurement Protocol (STAMP). The data model covers two STAMP logical entities – Session-Sender and Session-Reflector; characteristics of the STAMP test session, as well as measured and calculated performance metrics.

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved. Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info).

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

revision "2022-06-15" {
  description
    "Initial Revision. Base STAMP specification is covered";
  reference
    "RFC XXXX: STAMP YANG Data Model."
}

typedef session-reflector-mode {
  type enumeration {
    enum stateful {
      stateful
      }
    }
  }

/*
 * Typedefs
 */

typedef session-reflector-mode {
  type enumeration {
    enum stateful {
      stateful
      }
    }
  }

description
 "When the Session-Reflector is stateful,
i.e. is aware of STAMP-Test session state.";
}
enum stateless {
    description
    "When the Session-Reflector is stateless,
i.e. is not aware of the state of
    STAMP-Test session.";
}
}
description "State of the Session-Reflector";
reference
 "RFC 8762 Simple Two-way Active
Measurement Protocol (STAMP) Section 4.";
}
typedef session-dscp-mode {
    type enumeration {
        enum copy-received-value {
            description
            "Use DSCP value copied from received
            STAMP 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 timestamp-format {
    type enumeration {
        enum ntp-format {
            description
            "NTP 64 bit format of a timestamp";
        }
        enum ptp-format {
            description
            "PTPv2 truncated format of a timestamp";
        }
    }
    description
    "Timestamp format used by Session-Sender
or Session-Reflector.";
}
typedef percentage {
type decimal64 {
  fraction-digits 5;
}
description "Percentage";
}

typedef percentile {
type decimal64 {
  fraction-digits 5;
}
description "Percentile is a measure used in statistics indicating the value below which a given percentage of observations in a group of observations fall.";
}

/ * Feature definitions. */
feature session-sender {
description "This feature relates to the device functions as the STAMP Session-Sender";
reference "RFC 8762 Simple Two-way Active Measurement Protocol (STAMP) Section 4.2.";
}

feature session-reflector {
description "This feature relates to the device functions as the STAMP Session-Reflector";
reference "RFC 8762 Simple Two-way Active Measurement Protocol (STAMP) Section 4.3.";
}

feature stamp-security {
description "Secure STAMP supported";
reference
"RFC 8762 Simple Two-way Active Measurement Protocol (STAMP) Section 4.4."
}

feature extra-padding-tlv {
  description
    "An extra padding to be used by the Session-Sender";
  reference
    "RFC 8972 STAMP Optional Extensions Section 4.1."
}

feature location-tlv {
  description
    "Request the location information from the Session-Reflector";
  reference
    "RFC 8972 STAMP Optional Extensions Section 4.2."
}

feature timestamp-information-tlv {
  description
    "Request from the Session-Reflector to report on the method obtaining a timestamp";
  reference
    "RFC 8972 STAMP Optional Extensions Section 4.3."
}

feature class-of-service-tlv {
  description
    "Request the Session-Reflector to report on the DSCP value received";
  reference
    "RFC 8972 STAMP Optional Extensions Section 4.4."
}

feature direct-measurement-tlv {
  description
    "Request the Session-Reflector to report the current result of a direct loss measurement";
  reference
    "RFC 8972 STAMP"
Optional Extensions Section 4.5.

feature location-tlv {
    description
    "Request the Session-Reflector to report on the 3GPP access";
    reference
    "RFC 8972 STAMP Optional Extensions Section 4.6."
}

feature followup-telemetry-tlv {
    description
    "Request the Session-Reflector to report the time of transmission of the previous STAMP test packet";
    reference
    "RFC 8972 STAMP Optional Extensions Section 4.7."
}

feature hmac-tlv {
    description
    "A TLV to provide the authentication protection for STAMP extensions.";
    reference
    "RFC 8972 STAMP Optional Extensions Section 4.8."
}

/*
 * Groupings
 */

grouping extra-padding {
    description "Extra padding in a STAMP test packet added at the Session-Sender as per RFC 8972 Section 4.1."
    leaf extra-padding-length {
        type uint32;
        default "0";
        description "Number of octets to be added to a test packet at the Session-Sender."
    }
}

grouping class-of-service {

description "Request the Session-Reflector to report the DSCP value in the received and, optionally, use the intended DSCP value for the reflected STAMP test packet as per RFC 8972 Section 4.4.";
leaf dscp-intended {
    type inet:dscp;
    default "0";
    description "The DSCP value to be used by the Session-Reflector in the reflected STAMP test packet.";
}

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 test-session-statistics {
    description "Performance metrics calculated for a STAMP test session.";
    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 sender-timestamp-format {
    type timestamp-format;
    description "Sender Timestamp format";
}

leaf reflector-timestamp-format {
    type timestamp-format;
    description "Reflector Timestamp format";
}

leaf dscp {
    type inet:dscp;
    description
       "The DSCP value that was placed in the header of
       STAMP UDP test packets by the Session-Sender.";
}

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 "/stamp/stamp-session-sender/"
        +"sender-test-session[stamp-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 "/stamp/stamp-session-sender/"
        +"sender-test-session[stamp-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 "/stamp/stamp-session-sender/"
        +"sender-test-session[stamp-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 "/stamp/stamp-session-sender/
      +"sender-test-session[stamp-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 "/stamp/stamp-session-sender/
      +"sender-test-session[stamp-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;
}

uses session-parameters;
uses maintenance-statistics;

grouping stamp-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:gauge64;
            units nanoseconds;
            description
                "Min of Packets transmitted delay";
        }
        leaf max {
            type yang:gauge64;
            units nanoseconds;
            description
                "Max of Packets transmitted delay";
        }
        leaf avg {
            type yang:gauge64;
            units nanoseconds;
            description
                "Avg of Packets transmitted delay";
        }
    }
    container delay-variation {
        description "Packets transmitted delay variation";
        leaf min {
            type yang:gauge32;
            units nanoseconds;
            description
                "Min of Packets transmitted delay variation";
        }
        leaf max {
            type yang:gauge32;
            units nanoseconds;
            description
                "Max of Packets transmitted delay variation";
        }
    }
}
leaf avg {
  type yang:gauge32;
  units nanoseconds;
  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 yang:gauge64;
      units nanoseconds;
      description
        "Percentile of round-trip delay";
    }
    leaf near-end-delay {
      type yang:gauge64;
      units nanoseconds;
      description
        "Percentile of near-end delay";
    }
    leaf far-end-delay {
      type yang:gauge64;
      units nanoseconds;
      description
        "Percentile of far-end delay";
    }
  }
  container delay-variation-percentile {
    description
      "Report round-trip, near- and far-end delay variation";
    leaf rtt-delay-variation {
      type yang:gauge32;
      units nanoseconds;
      description
        "Percentile of round-trip delay-variation";
    }
    leaf near-end-delay-variation {
      type yang:gauge32;
      units nanoseconds;
      description
        "Percentile of near-end delay-variation";
    }
  }
}
leaf far-end-delay-variation {
    type yang:gauge32;
    units nanoseconds;
    description
    "Percentile of far-end delay-variation";
}

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 consecutively lost packets during the test interval.";
    }
    leaf loss-burst-min {
        type int32;
        description
        "Minimum number of consecutively 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-parameters {
    description
"Parameters Session-Sender"

leaf sender-ip {
    type inet:ip-address;
    mandatory true;
    description "Sender IP address";
}

leaf session-sender-udp-port {
    type inet:port-number {
        range "49152..65535";
    }
    mandatory true;
    description "Sender UDP port number";
    reference
    "RFC 8762 Simple Two-Way Active Measurement Protocol Section 4.1.";
}

leaf stamp-session-id {
    type uint16;
    description
    "A STAMP test session identifier assigned by the Session-Sender.";
    reference
    "RFC 8972 Simple Two-Way Active Measurement Protocol Optional Extensions Section 3.";
}

leaf session-reflector-ip {
    type inet:ip-address;
    mandatory true;
    description "Reflector IP address";
}

leaf session-reflector-udp-port {
    type inet:port-number{
        range "862 | 1024..49151 | 49152..65535";
    }
    default 862;
    description
    "Reflector UDP port number";
    reference
    "RFC 8762 Simple Two-Way Active Measurement Protocol Section 4.1.";
}

}

grouping session-security {
    description
    "Grouping for STAMP security and related parameters";
    container security {

if-feature stamp-security;
presence "Enables secure STAMP";
description
  "Parameters for STAMP authentication";
leaf key-chain {
  type kc:key-chain-ref;
  description "Name of key-chain";
}
}
reference
  "RFC 8762 Simple Two-Way Active Measurement Protocol Section 4.4.";

/*==============================================================================*
 * Configuration Data
 /*
 container stamp {
  description
    "Top level container for STAMP configuration";

container stamp-session-sender {
  if-feature session-sender;
  description "STAMP Session-Sender container";

  leaf sender-enable {
    type boolean;
    default "true";
    description
      "Whether this network element is enabled to act as STAMP Session-Sender";
    reference
      "RFC 8762 Simple Two-Way Active Measurement Protocol Section 4.2.";
  }

  list sender-test-session {
    key "stamp-session-id";
    unique "stamp-session-id";
    description
      "This structure is a container of test session managed objects";

    leaf test-session-enable {
      type boolean;
      default "true";
      description
        "Whether this STAMP 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 STAMP-Test session is
   bound by number of test packets or not.";
}

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 STAMP-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
 STAMP-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 STAMP-Reflector for the test session.";
    reference
       "RFC 8762 STAMP Section 4."
 }

uses session-parameters;
leaf sender-timestamp-format {
    type timestamp-format;
    default ntp-format;
    description "Sender Timestamp format";
 }

uses session-security;
uses stamp-session-percentile;
}

container stamp-session-reflector {
    if-feature session-reflector;
    description
       "STAMP Session-Reflector container";
    leaf reflector-enable {
    type boolean;
    default "true";
    description
       "Whether this network element is enabled to
        act as STAMP Session-Reflector";
    }

    leaf ref-wait {
    type uint32 {
         range 1..604800;
    }
units seconds;
default 900;
description
"REFWAIT(STAMP test session timeout in seconds),
the default value is 900";
}

leaf reflector-mode-state {
type session-reflector-mode;
default stateless;
description
"The state of the mode of the STAMP
Session-Reflector";
reference
"RFC 8762 STAMP Section 4."
}

list reflector-test-session {
key "session-index";
unique "sender-ip stamp-session-id";
description
"This structure is a container of test session
managed objects";

leaf session-index {
type uint32;
description "Session index";
}

leaf stamp-session-id {
type union {
type uint16;
type enumeration {
enum any {
description
"Indicates that the Session-Reflector
accepts STAMP test packets from
a Session-Sender with any SSID
value";
reference
"RFC 8972 Simple Two-Way Active
Measurement Protocol Optional
Extensions Section 3.";
}
}
}

leaf dscp-handling-mode {
type session-dscp-mode;
default copy-received-value;
description
  "Session-Reflector handling of DSCP:
  - use value copied from received STAMP-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."
}

leaf sender-ip {
  type union {
    type inet:ip-address;
    type enumeration {
      enum any {
        description
        "Indicates that the Session-Reflector
        accepts STAMP test packets from
        any Session-Sender";
      }
    }
  }
}
default any;
description
  "This value determines whether specific
  IPv4/IPv6 address of the Session-Sender
  or the wildcard, i.e. any address";
}

leaf sender-udp-port {
  type union {
    type inet:port-number {
      range "49152..65535";
    }
    type enumeration {
      enum any {
        description
        "Indicates that the Session-Reflector
        accepts STAMP test packets from
        any Session-Sender";
      }
    }
  }
}
leaf reflector-ip {
    type union {
        type inet:ip-address;
        type enumeration {
            enum any {
                description "This value determines whether specific IPv4/IPv6 address of the Session-Reflector or the wildcard, i.e. any address";
            }
        }
    }
    default any;
    description "This value determines whether specific port number of the Session-Sender or the wildcard, i.e. any";
}

leaf reflector-udp-port {
    type inet:port-number{
        range "862 | 1024..49151 | 49152..65535";
    }
    default 862;
    description "Reflector UDP port number";
    reference "RFC 8762 Simple Two-Way Active Measurement Protocol Section 4.1.";
}

leaf reflector-timestamp-format {
    type timestamp-format;
    default ntp-format;
    description "Reflector Timestamp format";
}

uses session-security;
container stamp-state {
    config false;
    description
        "Top level container for STAMP state data";
}

/*
 * Operational state data nodes
 */

container stamp-state {
    config false;
    description
        "Top level container for STAMP state data";
}

container stamp-session-sender-state {
    if-feature session-sender;
    description
        "Session-Sender container for state data";
    list test-session-state {
        key "session-index";
        description
            "This structure is a container of test session managed objects";
        leaf session-index {
            type uint32;
            description "Session index";
        }
    }
    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 STAMP 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";
        }
    }
}
uses test-session-statistics;
}
}

list history-stats {
  key session-index;
  description
    "This container contains the results for the history Measurement Interval in a Measurement session ";
  leaf session-index {
    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";
  }
  uses test-session-statistics;
}
}

container stamp-session-refl-state {
  if-feature session-reflector;
  description
    "STAMP Session-Reflector container for state data";
  leaf reflector-light-admin-status {
    type boolean;
    description
      "Whether this network element is enabled to act as STAMP Session-Reflector";
  }
}

list test-session-state {
  key "session-index";
  description
    "This structure is a container of test session managed objects";
leaf session-index {
  type uint32;
  description "Session index";
}

leaf reflector-timestamp-format {
  type timestamp-format;
  description "Reflector Timestamp format";
}
uses session-parameters;
uses maintenance-statistics;
}
}
}

rpc stamp-sender-start {
    description "start the configured sender session";
    input {
      leaf stamp-session-id {
        type uint16;
        mandatory true;
        description "The STAMP session to be started";
      }
    }
}
}

rpc stamp-sender-stop {
    description "stop the configured sender session";
    input {
      leaf stamp-session-id {
        type uint16;
        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.


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 [RFC7950].

name: ietf-stamp
prefix: stamp
reference: RFC XXXX

5. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The NETCONF access control model [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a pre-configured subset of all available NETCONF or RESTCONF protocol operations and content.

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

TBD
Unauthorized access to any data node of these subtrees can adversely affect the routing subsystem of both the local device and the network. This may lead to corruption of the measurement that may result in false corrective action, e.g., false negative or false positive. That could be, for example, prolonged and undetected deterioration of the quality of service or actions to improve the quality unwarranted by the real network conditions.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

TBD

Unauthorized access to any data node of these subtrees can disclose the operational state information of VRRP on this device.

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

TBD

6. Acknowledgments

Authors recognize and appreciate valuable comments provided by Adrian Pan and Henrik Nydell.

7. References

7.1. Normative References


7.2. Informative References

Appendix A. Example of STAMP Session Configuration

Figure 5 shows a configuration example of a STAMP-Sender.

```xml
<?xml version="1.0" encoding="utf-8"?>
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <stamp xmlns="urn:ietf:params:xml:ns:yang:ietf-stamp">
    <stamp-session-sender>
      <session-enable>enable</session-enable>
      <stamp-session-id>10</stamp-session-id>
      <test-session-enable>enable</test-session-enable>
      <number-of-packets>forever</number-of-packets>
      <interval>10</interval> <!-- 10 microseconds -->
      <measurement-interval/> <!-- use default 60 seconds -->
      <repeat/> <!-- use default 0 repetitions, i.e. do not repeat this session -->
      <dscp-value/> <!-- use deafult 0 (CS0) -->
      <test-session-reflector-mode/>
      <sender-ip></sender-ip>
      <session-sender-udp-port></session-sender-udp-port>
      <session-reflector-ip></session-reflector-ip>
      <session-reflector-udp-port/> <!-- use default 862 -->
      <sender-timestamp-format/>
      <first-percentile/> <!-- use default 95 -->
      <second-percentile/> <!-- use default 99 -->
      <third-percentile/> <!-- use default 99.9 -->
    </stamp-session-sender>
  </stamp>
</data>
```

Figure 5: XML instance of STAMP Session-Sender configuration
<?xml version="1.0" encoding="utf-8"?>
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <stamp xmlns="urn:ietf:params:xml:ns:yang:ietf-stamp">
    <stamp-session-reflector>
      <session-enable>enable</session-enable>
      <ref-wait/> <!-- use default 900 seconds -->
      <!-- use default 'stateless' -->
      <reflector-mode-state/>
      <stamp-session-id/> <!-- use default 'any' -->
      <!-- use default 'copy-received-value' -->
      <dscp-handling-mode/>
      <!-- not used because of dscp-hanling-mode
          being 'copy-received-value' -->
      <dscp-value/>
      <sender-ip/> <!-- use default 'any' -->
      <sender-udp-port/> <!-- use default 'any' -->
      <reflector-ip/> <!-- use default 'any' -->
      <reflector-udp-port/> <!-- use default 862 -->
      <reflector-timestamp-format/>
      <!-- No authentication -->
    </stamp-session-reflector>
  </stamp>
</data>

Figure 6: XML instance of STAMP Session-Reflector configuration

Authors’ Addresses

Greg Mirsky
Ericsson
Email: gregimirsky@gmail.com

Xiao Min
ZTE Corp.
Email: xiao.min2@zte.com.cn

Wei S Luo
Ericsson
Email: wei.s.luo@ericsson.com

Mirsky, et al. Expires 11 January 2023 [Page 38]
Abstract

This document defines a set of metrics for networking services with performance requirements expressed as Service Level Objectives (SLO). These metrics, referred to as Precision Availability Metrics (PAM), can be used to assess the service levels that are being delivered. Specifically, PAM can be used to assess whether a service is provided in compliance with its specified quality, i.e., in accordance with its defined SLOs.
1. Introduction

Network operators and network users often need to assess the quality with which network services are being provided and delivered. In particular in cases where service level guarantees are given and service level objectives (SLOs) are defined, it is essential to provide a measure of the degree with which actual service levels that are delivered comply with SLOs that were agreed, typically in a contract or agreement. Examples of service levels include service latency and packet loss. Simple examples of SLOs associated with such service levels would be target values for the maximum packet delay (one-way and/or round trip) or maximum packet loss ratio that would be deemed acceptable.

An example of an SLO is one that characterizes the continued ability of a particular set of nodes to communicate. Essentially, the absence of what is, in other contexts, is called a defect. The SLO would include the various time and measurement aspects that would be
interpreted as a defect or failure to communicate. It is important
to note that it is being defined as a state, and thus, it has
conditions that define entry into it and exit out of it. It is
expected that an SLA includes a defect-related SLO, possibly in
addition to other SLOs.

To express the perceived quality of delivered networking services
versus their SLOs, a set of metrics are needed to characterize the
quality of the service being provided. Of concern is not so much the
absolute service level (for example, actual latency experienced), but
whether the service is provided in accordance with the negotiated,
and eventually contracted, service levels. For instance, this may
include whether the packet delay that is experienced falls within an
acceptable range that has been contracted for the service. The
specific quality of service depends on the SLO that is in effect. A
non-conformance to an SLO might result in degradation of the quality
of experience for gamers or even jeopardize the safety of a large
geographical area. However, as those applications represent clear
business opportunities, they demand dependable technical solutions.

The same service level may be deemed acceptable for one application,
while unacceptable for another, depending on the needs of the
application. Hence it is not sufficient to simply measure service
levels per se over time, but to assess the quality of the service
being provided with the applicable SLO in mind. However, at this
point, there are no standard metrics in place that can be used to
account for the quality with which services are delivered relative to
their SLOs, and whether their SLOs are being met at all times. Such
metrics and the instrumentation to support them are essential for a
number of purposes, including monitoring (to ensure that networking
services are performing according to their objectives) as well as
accounting (to maintain a record of service levels delivered,
important for monetization of such services as well as for triaging
of problems).

The current state-of-the-art of metrics available today includes, for
example, interface metrics, useful to obtain data on traffic volume
and behavior that can be observed at an interface [RFC2863] and
[RFC8343], but agnostic of actual service levels and not specific to
distinct flows. Flow records [RFC7011] and [RFC7012] maintain
statistics about flows, including flow volume and flow duration, but
again, contain very little information about end-to-end service
levels, let alone whether the service levels delivered to meet their
targets, i.e., their associated SLOs.

This specification introduces a new set of metrics, Precision
Availability Metrics (PAM), aimed at capturing end-to-end service
levels for a flow, specifically the degree to which flows comply with
the SLOs that are in effect. PAM can be used to assess whether a service is provided in compliance with its specified quality, i.e., in accordance with its defined SLOs. This information can be used in multiple ways, for example, to optimize service delivery, take timely counteractions in the event of service degradation, or account for the quality of services being delivered.

Availability is discussed in Section 3.4 of [RFC7297]. In this document, the term "availability" reflects that a service which is characterized by its SLOs is considered unavailable whenever those SLOs are violated, even if basic connectivity is still working. "Precision" refers to the fact that services whose end-to-end service levels are governed by SLOs, and which must therefore be precisely delivered according to the associated quality and performance requirements. It should be noted that "precision" refers to what is being assessed, not to the mechanism used to measure it; in other words, it does not refer to the precision of the mechanism with which actual service levels are measured. The specification and implementation of methods that provide for accurate measurements is a separate topic independent of the definition of the metrics in which the results of such measurements would be expressed.

[Ed.note: It should be noted that at this point, the set of metrics proposed here is intended as a "starter set" that is intended to spark further discussion. Other metrics are certainly conceivable; we expect that the list of metrics will evolve as part of the Working Group discussions.]

2. Conventions and Terminology

2.1. Terminology

In this document, SLA and SLO are used as defined in Section 4.1 [I-D.ietf-teas-ietf-network-slices].

2.2. Acronyms

[Ed>Note: needs updating.]

PAM Precision Availability Metric
OAM Operations, Administration, and Maintenance
VI Violated Interval
VIR Violated Interval Ratio
SVI Severely Violated Interval
SVIR Severely Violated Interval Ratio

VFI Violation-Free Interval

3. Performance Availability Metrics

3.1. Introducing Violated Intervals

When analyzing the availability metrics of a service flow between two nodes, we need to select a time interval as the unit of PAM. In [ITU.G.826], a time interval of one second is used. That is reasonable, but some services may require different granularity. For that reason, the time interval in PAM is viewed as a variable parameter though constant for a particular measurement session. Further, for the purpose of PAM, each time interval, e.g., second or decamillisecond, is classified either as Violated Interval (VI), Severely Violated Interval (SVI), or Violation-Free Interval (VFI). These are defined as follows:

* VI is a time interval during which at least one of the performance parameters degraded compared to its pre-defined optimal level threshold.

* SVI is a time interval during which at least one the performance parameters degraded compared to its pre-defined critical threshold.

* Consequently, VFI is a time interval during which all performance objectives are at or better than their respective pre-defined optimal levels. In such a case, the service is in compliance with its specification.

Mechanisms of setting levels of threshold of an SLO are outside the scope for this document.

From these definitions, a set of basic metrics can be defined that count the numbers of time intervals that fall into each category:

* VI count.

* SVI count.

* VFI count.

These count metrics are essential in calculating respective ratios that can be used to assess the instability of the service.
3.2. Derived Precision Availability Metrics

A set of metrics can be created based on PAM introduced in Section 3. In this document, these metrics are referred to as derived PAM. Some of these metrics are modeled after Mean Time Between Failure (MTBF) metrics - a "failure" in this context referring to a failure to deliver a packet according to its SLO.

* Time since the last violated interval (e.g., since last violated ms, since last violated second). (This parameter is suitable for monitoring the current compliance status of the service, e.g., for trending analysis.)

* Packets since the last violated packet. (This parameter is suitable for the monitoring of the current compliance status of the service.)

* Mean time between EIs (e.g., between violated milliseconds, violated seconds) is the arithmetic mean of time between consecutive EIs.

* Mean packets between EIs is the arithmetic mean of the number of SLO-compliant packets between consecutive EIs. (Another variation of "MTBF" in a service setting.)

An analogous set of metrics can be produced for SVI:

* Time since the last SVI (e.g., since last violated ms, since last violated second). (This parameter is suitable for the monitoring of the current compliance status of the service.)

* Packets since the last severely violated packet. (This parameter is suitable for the monitoring of the current compliance status of the service.)

* Mean time between SVIs (e.g., between severely violated milliseconds, severely violated seconds) is the arithmetic mean of time between consecutive SVIs.

* Mean packets between SVIs is the arithmetic mean of the number of SLO-compliant packets between consecutive SVIs. (Another variation of "MTBF" in a service setting.)

Determining the condition in which the path is currently with respect to availability/unavailability is helpful. But because switching between periods requires ten consecutive intervals, shorter conditions may not be adequately reflected. Two additional PAMs can be used, and they are defined as follows:
* violated interval ratio (VIR) is the ratio of VI to the total number of time unit intervals in a time of the availability periods during a fixed measurement interval.

* severely violated interval ratio (SVIR) - is the ratio of SVIs to the total number of time unit intervals in a time of the availability periods during a fixed measurement interval.

3.3. Service Availability in PAMs

VI, SVI, and VFI characterize the communication between two nodes relative to the level of required and acceptable performance and when the performance level degrades below an acceptable level. The former condition in this document defined to as service availability. The latter is defined as service unavailability. Based on the definitions in Section 3.1, SVI is the one time interval of service unavailability while VI and VFI present an interval of service availability. Since the conditions of the service are are continually changing, periods of availability and unavailability need to be defined with duration larger than one time interval to reduce the number of state changes while correctly reflecting the service condition. The method to determine the state of the service in terms of PAM is described below:

* If ten consecutive SVIs been detected, then the PAM state of the service is defined as unavailability, and the beginning of that period of unavailability state is at the start of the first SVI in the sequence of the consecutive SVIs.

* Similarly, for ten consecutive non-SVIs (i.e., either VIs or VFI s), the service is defined to be available. The start of that period is at the beginning of the first non-SVI.

* Resulting from these two definitions, a sequence of less than ten consecutive SVIs or non-SVIs does not change the PAM state of the service. For example, if the PAM state is determined as unavailable, a sequence of seven VFI s is not viewed as an availability period.

4. Statistical SLO

It should be noted that certain Service Level Agreements (SLA) may be statistical, requiring the service levels of packets in a flow to adhere to specific distributions. For example, an SLA might state that any given SLO applies only to a certain percentage of packets, allowing for a certain level of, for example, packet loss and/or exceeding packet delay threshold to take place. Each such event, in that case, does not necessarily constitute an SLO violation.
However, it is still useful to maintain those statistics, as the number of out-of-SLO packets still matters when looked at in proportion to the total number of packets.

Along that vein, an SLA might establish an SLO of, say, end-to-end latency to not exceed 20 ms for 99% of packets, to not exceed 25 ms for 99.999% of packets, and to never exceed 30 ms for any packet. In that case, any individual packet with latency larger than 20 ms latency and lower than 30 ms cannot be considered an SLO violation in itself, but compliance with the SLO may need to be assessed after the fact.

To support statistical services more directly requires additional metrics, such as metrics that represent histograms for service level parameters with buckets corresponding to individual service level objectives. For the example just given, a histogram for a given flow could be maintained with three buckets: one containing the count of packets within 20 ms, a second with a count of packets between 20 and 25 ms (or simply all within 25 ms), a third with a count of packets between 25 and 30 ms (or merely all packets within 30 ms), and a fourth with a count of anything beyond (or simply a total count). Of course, the number of buckets and the boundaries between those buckets should correspond to the needs of the SLA associated with the application, i.e., to the specific guarantees and SLOs that were provided. The definition of histogram metrics is for further study.

5. Other PAM Benefits

PAM provides a number of benefits with other, more conventional performance metrics. Without PAM, it would be possible to conduct ongoing measurements of service levels and maintain a time-series of service level records, then assess compliance with specific SLOs after the fact. However, doing so would require the collection of vast amounts of data that would need to be generated, exported, transmitted, collected, and stored. In addition, extensive postprocessing would be required to compare that data against SLOs and analyze its compliance. Being able to perform these tasks at scale and in real-time would present significant additional challenges.

Adding PAM allows for a more compact expression of service level compliance. In that sense, PAM does not simply represent raw data but expresses actionable information. In conjunction with proper instrumentation, PAM can thus help avoid expensive postprocessing.
6. Discussion Items

The following items require further discussion:

* Metrics. The foundational metrics defined in this draft refer to violated intervals. In addition, counts of violations related to individual packets may also need to be maintained. Metrics referring to violated packets (i.e., packets that on an individual basis miss a performance objective) may be added in a later revision of this document.

The following is a list of items for which further discussion is needed as to whether they should be included in the scope of this specification:

* A YANG data model.

* A set of IPFIX Information Elements.

* Statistical metrics: e.g., histograms/buckets.

* Policies regarding the definition of "violated" and "severely violated" time interval.

* Additional second-order metrics, such as "longest disruption of service time" (measuring consecutive time units with SVIs).

7. IANA Considerations

This document has no IANA actions.

8. Security Considerations

Instrumentation for metrics that are used to assess compliance with SLOs constitute an attractive target for an attacker. By interfering with the maintaining of such metrics, services could be falsely identified as complying (when they are not) or vice-versa (i.e., flagged as being non-compliant when indeed they are). While this document does not specify how networks should be instrumented to maintain the identified metrics, such instrumentation needs to be adequately secured to ensure accurate measurements and prohibit tampering with metrics being kept.

Where metrics are being defined relative to an SLO, the configuration of those SLOs needs to be adequately secured. Likewise, where SLOs can be adjusted, the correlation between any metrics instance and a particular SLO must be clear. The same service levels that constitute SLO violations for one flow that should be maintained as
part of the "violated time units" and related metrics, may be perfectly compliant for another flow. In cases when it is impossible to tie together SLOs and PAM properly, it will be preferable to merely maintain statistics about service levels delivered (for example, overall histograms of end-to-end latency) without assessing which constitutes violations.

By the same token, where the definition of what constitutes a "severe" or a "significant" violation depends on policy or context. The configuration of such policy or context needs to be specially secured. Also, the configuration of this policy must be bound to the metrics being maintained. This way, it will be clear which policy was in effect when those metrics were being assessed. An attacker that can tamper with such policies will render the corresponding metrics useless (in the best case) or misleading (in the worst case).

9. Acknowledgments

TBA

10. References

10.1. Informative References

[I-D.ietf-teas-ietf-network-slices]

[ITU.G.826]


Contributors’ Addresses

Liuyan Han
China Mobile
32 XuanWuMenXi Street
Beijing
100053
China
Email: hanliuyan@chinamobile.com

Mohamed Boucadair
Orange
35000 Rennes
France
Email: mohamed.boucadair@orange.com

Authors’ Addresses

Greg Mirsky
Ericsson
Email: gregimirsky@gmail.com

Joel Halpern
Ericsson
Email: joel.halpern@ericsson.com

Xiao Min
ZTE Corp.
Email: xiao.min2@zte.com.cn
