Workgroup: Network Working Group
Internet-Draft:
draft-ietf-ippm-capacity-protocol-02
Published: 9 July 2022
Intended Status: Standards Track
Expires: 10 January 2023
Authors: L. Ciavattone A. Morton
AT&T Labs AT&T Labs

Test Protocol for One-way IP Capacity Measurement

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

Copyright Notice

Table of Contents

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 messsages 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, regardlesss of transmiision 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 neccessary.

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:

```
     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
     uint8_t authMode;     // Authentication mode
     uint32_t authUnixTime;// Authentication time stamp
     unsigned char authDigest[AUTH_DIGEST_LENGTH] // SHA256_DIGEST_LE
```

The UDP PDU format layout SHALL be 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 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          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 preeding 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 implemenation 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 udpst.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
        uint8_t modifierBitmap;// Modifier bitmap (replaced jumboStatus,
        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_AUTHNC   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_BADTMTU  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
     uint8_t modifierBitmap;// Modifier bitmap (replaced jumboStatus
     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:

*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 neccessary.

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:

```
        uint16_t controlId;          // Control ID
        uint16_t protocolVer;        // Protocol version
        uint8_t cmdRequest;          // Command request, 1 = upstream, 2
        uint8_t cmdResponse;         // Command response (set to 0)
        uint16_t lowThresh;          // Low delay variation threshold
        uint16_t upperThresh;        // Upper delay variation threshold
        uint16_t trialInt;           // Status feedback/trial interval (
        uint16_t testIntTime;        // Test interval time (sec)
        uint8_t subIntPeriod;        // Sub-interval period (sec)
        uint8_t ipTosByte;           // IP ToS byte for testing
        uint16_t srIndexConf;        // Configured sending rate index (s
        uint8_t useOwDelVar;         // Use one-way delay instead of RTT
        uint8_t highSpeedDelta;      // High-speed row adjustment delta
        uint16_t slowAdjThresh;      // Slow rate adjustment threshold
        uint16_t seqErrThresh;       // Sequence error threshold
        uint8_t ignoreOooDup;        // Ignore Out-of-Order/Duplicate da
*       uint8_t modifierBitmap;      // Modifier bitmap (replaced reserv
*       uint8_t rateAdjAlgo;         // Rate adjust. algo. (replaced res
*       uint8_t reserved1;           // (Alignment) (replaced reserved2
```

```
Control Header Test Activation Command Request Values:
CHTA_CREQ_NONE      0 = No Request
CHTA_CREQ_TESTACTUS 1 = Request test in Upstream direction (client to se
CHTA_CREQ_TESTACTDS 2 = Request test in Downstream direction (server to

modifierBitmap Code Field: Test Activation
CHTA_SRIDX_ISSTART 0x01 = Set when srIndexConf IS START rate for search
CHTA_RAND_PAYLOAD  0x02 = Set for RANDOMIZED UDP payload

rateAdjAlgo Values:
CHTA_RA_ALGO_B   = 0              // 0 = Algo. B, allows Algo. expansion
CHTA_RA_ALGO_MIN = CHTA_RA_ALGO_B // Limit check (with Algo B only)
CHTA_RA_ALGO_MAX = CHTA_RA_ALGO_B // Limit check (with Algo B only)

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 paramete
```

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 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:

```
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:

```
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

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-|-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          txInterval1                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          udpPayload1                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          burstSize1                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          txInterval2                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          udpPayload2                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          burstSize2                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          udpAdddon2                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Note that the server additionally has the option of completely rejecting the request and sending back an appropriate command response value:

```
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 neccessary.

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:

*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.

*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 (=0xBEEF for the LOad PDU)
      uint8_t testAction;  // Test action (= 0x00 normally, until test
      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
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):

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |           loadId              |   testAction  | rxStopped     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           lpduSeqNo                           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |           udpPayload          |           spduSeqErr          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                          spduTime_sec                         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                         spduTime_nsec                         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-|-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                          lpduTime_sec                         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                         lpduTime_nsec                         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   .                    MBZ = udpPayload - 28 octets               .
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   .                                                               .
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   .                                                               .
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   .                                                               .
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   .                                                               .
```

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 end-point.

@@@@ To Do: What protections from bit errors (checksum) or on-path attacks (something stronger) are warrented 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 octet
        //
        uint32_t subIntSeqNo;      // Sub-interval sequence number
        struct subIntStats sisSav; // Sub-interval Saved Stats Structure
        //
        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-
        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, co
        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):

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          statusId             |   testAction  | rxStopped     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           spduSeqNo                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
.               Sending Rate Structure (28 octets)              .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          subIntSeqNo                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
.        Sub-interval Saved Stats Structure  (52 octets)        .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           seqErrLoss                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           seqErrOoo                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           seqErrDup                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         clockDeltaMin                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          delayVarMin                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          delayVarMax                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          delayVarSum                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          delayVarCnt                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          rttMinimum                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           rttSample                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  delayMinUpd  |   reserved2   |           reserved3           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          tiDeltaTime                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         tiRxDatagrams                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           tiRxBytes                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         spduTime_sec                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         spduTime_nsec                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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:

```
        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
------------------------------------------------------------------------
```

```
   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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                          rxDatagrams                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                            rxBytes                            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           deltaTime                           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           seqErrLoss                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           seqErrOoo                           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           seqErrDup                           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                          delayVarMin                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                          delayVarMax                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                          delayVarSum                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                          delayVarCnt                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                          rttMinimum                           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                          rttMaximum                           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           accumTime                           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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 acces 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.

```
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

```
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 formateed 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 Activtion (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 accomodate 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 suggest

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

*** We would like a SEC-DIR recommendation to accomplish C and/or D bel

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-know
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 m
+ synergy: we are already using the OpenSSL library in the running code

New: we think this mode D might not be used very often, the demands on h
measure at Gbps rates usually require all the cycles they can allocate t
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

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] Morton, A., Geib, R., and L. Ciavattone, "Metrics and Methods for One-Way IP Capacity", Work in Progress, Internet-Draft, draft-ietf-ippm-capacity-metric-method-12, 9 June 2021, <https://www.ietf.org/archive/id/draft-ietf-ippm-capacity-metric-method-12.txt>.

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

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

[RFC2681] Almes, G., Kalidindi, S., and M. Zekauskas, "A Round-trip Delay Metric for IPPM", RFC 2681, DOI 10.17487/RFC2681, September 1999, <https://www.rfc-editor.org/info/rfc2681>.

[RFC6438] Carpenter, B. and S. Amante, "Using the IPv6 Flow Label for Equal Cost Multipath Routing and Link Aggregation in Tunnels", RFC 6438, DOI 10.17487/RFC6438, November 2011, <https://www.rfc-editor.org/info/rfc6438>.

[RFC7497] Morton, A., "Rate Measurement Test Protocol Problem Statement and Requirements", RFC 7497, DOI 10.17487/RFC7497, April 2015, <https://www.rfc-editor.org/info/rfc7497>.

[RFC7680] Almes, G., Kalidindi, S., Zekauskas, M., and A. Morton, Ed., "A One-Way Loss Metric for IP Performance Metrics (IPPM)", STD 82, RFC 7680, DOI 10.17487/RFC7680, January 2016, <https://www.rfc-editor.org/info/rfc7680>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8468] Morton, A., Fabini, J., Elkins, N., Ackermann, M., and V. Hegde, "IPv4, IPv6, and IPv4-IPv6 Coexistence: Updates for the IP Performance Metrics (IPPM) Framework", RFC 8468, DOI 10.17487/RFC8468, November 2018, <https://www.rfc-editor.org/info/rfc8468>.

[RFC9097] Morton, A., Geib, R., and L. Ciavattone, "Metrics and Methods for One-Way IP Capacity", RFC 9097, DOI 10.17487/RFC9097, November 2021, <https://www.rfc-editor.org/info/rfc9097>.

13.2. Informative References

[copycat] Edleine, K., Kuhlewind, K., Trammell, B., and B. Donnet, "copycat: Testing Differential Treatment of New Transport Protocols in the Wild (ANRW '17)", 15 July 2017, <https://irtf.org/anrw/2017/anrw17-final5.pdf>.

[LS-SG12-A] 12, I. S., "LS - Harmonization of IP Capacity and Latency Parameters: Revision of Draft Rec. Y.1540 on IP packet transfer performance parameters and New Annex A with Lab Evaluation Plan", May 2019, <https://datatracker.ietf.org/liaison/1632/>.

[LS-SG12-B] 12, I. S., "LS on harmonization of IP Capacity and Latency Parameters: Consent of Draft Rec. Y.1540 on IP packet transfer performance parameters and New Annex A with Lab & Field Evaluation Plans", March 2019, <https://datatracker.ietf.org/liaison/1645/>.

[RFC2544] Bradner, S. and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices", RFC 2544, DOI 10.17487/RFC2544, March 1999, <https://www.rfc-editor.org/info/rfc2544>.

[RFC3148] Mathis, M. and M. Allman, "A Framework for Defining Empirical Bulk Transfer Capacity Metrics", RFC 3148, DOI 10.17487/RFC3148, July 2001, <https://www.rfc-editor.org/info/rfc3148>.

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

[RFC5136] Chimento, P. and J. Ishac, "Defining Network Capacity", RFC 5136, DOI 10.17487/RFC5136, February 2008, <https://www.rfc-editor.org/info/rfc5136>.

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

[RFC6815] Bradner, S., Dubray, K., McQuaid, J., and A. Morton, "Applicability Statement for RFC 2544: Use on Production Networks Considered Harmful", RFC 6815, DOI 10.17487/RFC6815, November 2012, <https://www.rfc-editor.org/info/rfc6815>.

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

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

[RFC7799] Morton, A., "Active and Passive Metrics and Methods (with Hybrid Types In-Between)", RFC 7799, DOI 10.17487/RFC7799, May 2016, <https://www.rfc-editor.org/info/rfc7799>.

[RFC8337] Mathis, M. and A. Morton, "Model-Based Metrics for Bulk Transport Capacity", RFC 8337, DOI 10.17487/RFC8337, March 2018, <https://www.rfc-editor.org/info/rfc8337>.

[RFC8762] Mirsky, G., Jun, G., Nydell, H., and R. Foote, "Simple Two-Way Active Measurement Protocol", RFC 8762, DOI 10.17487/RFC8762, March 2020, <https://www.rfc-editor.org/info/rfc8762>.

[RFC8972] Mirsky, G., Min, X., Nydell, H., Foote, R., Masputra, A., and E. Ruffini, "Simple Two-Way Active Measurement Protocol Optional Extensions", RFC 8972, DOI 10.17487/RFC8972, January 2021, <https://www.rfc-editor.org/info/rfc8972>.

[TR-471] Morton, A., "Broadband Forum TR-471: IP Layer Capacity Metrics and Measurement", 10 July 2020, <https://www.broadband-forum.org/technical/download/TR-471.pdf>.

[udpst] udpst Project Collaborators, "UDP Speed Test Open Broadband project", December 2020, <https://github.com/BroadbandForum/obudpst>.

[Y.1540] Y.1540, I. R., "Internet protocol data communication service - IP packet transfer and availability performance parameters", December 2019, <https://www.itu.int/rec/T-REC-Y.1540-201912-I/en>.

[Y.Sup60] Morton, A., Rapporteur., "Recommendation Y.Sup60, (09/20) Interpreting ITU-T Y.1540 maximum IP-layer capacity measurements", 11 September 2020, <https://www.itu.int/rec/T-REC-Y.Sup60/en>.

Authors' Addresses

Len Ciavattone
AT&T Labs
200 Laurel Avenue South
Middletown,, NJ 07748
United States of America

Email: lencia@att.com

Al Morton

AT&T Labs
Chicago,, IL 60660
United States of America

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