Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 9 August 2025

L. Ciavattone
AT&T Labs
R. Geib
Deutsche Telekom
5 February 2025

Test Protocol for One-way IP Capacity Measurement
draft-ietf-ippm-capacity-protocol-12

Abstract

This document addresses the problem of protocol support for measuring Network Capacity metrics discussed 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 document defines the UDP Speed Test Protocol, a ~~simple~~ protocol to perform the RFC 9097 (and other) measurements.

Commenté [MB1]: Which method?

Commenté [MB2]: May be deleted

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.~~

~~In that time, t~~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 document 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.

This protocol supports measurement features which weren't available by TCP based speed tests and standard measurement protocols like OWAMP [RFC4656], TWAMP [RFC5357], and STAMP [RFC8762] prior to this work. The controlled Bulk Capacity measurement or Speed Test, respectively, is based on UDP rather than TCP. The bulk measurement load is unidirectional.

UDPSTP is at least partially compliant to section 3.1 of [RFC8085]: a bottleneck is congested, but pending congestion is avoided by limiting the duration of that congestion to the minimum required to determine the bottleneck capacity. This is ~~done~~ achieved by specifying periodic Status Feedback messages to steer a measurement load control algorithm (starting by sending conservatively chosen loads at the beginning of a measurement). In addition to this functional enhancement of IP performance measurement control protocols, the UDPSTP security features have been developed following guidance given by the IETF Security Area and the resulting specification is compliant with state of the art security implementations.

This document was edited by Al Morton, who passed away before being able to finalize this work. Ruediger Geib only joined later to help finalizing this draft. [EDITOR: if preferred, sentence may be moved to Contributors/Acknowledgements, but should stay in doc]

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 document is to define a protocol to measure the Maximum IP-Layer Capacity metric and according to the standardized method. ~~We note that a~~Some aspects of this protocol and end-host configuration can lead to support of additional forms of measurement, such as application emulation enabled by creative use of the load rate adjustment algorithm.

The ~~continued~~ goal is to harmonize the specified IP-Layer Capacity

Commenté [MB3]: Please expand

a mis en forme : Surlignage

Commenté [MB4]: It is too early to dive into this in an introduction. Please move this to another section of the document

Please focus here in introducing the protocol and the features it provided.

Commenté [MB5]: Please moved to the ACK section.

Commenté [MB6]: I don't parse this.

metric and method across the industry.~~, and t~~ This protocol supports the specifications of IETF and other Standards Development Organizations (SDOs).

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-point~~endpoints (without expressing requirements or ~~describing~~ elaborating on the authentication and encryption aspects~~; later sections provide these details and requirements~~).

One ~~end-point~~endpoint takes the role of server, listening for connection requests on a well-known ~~destination~~ port number from the other ~~end-point~~endpoint, 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~~ a hostname or IP address(es) of the server in order to begin the setup and configuration exchanges with the server.

Additionally, multi-connection (multi-flow) testing is ~~inherently~~ supported by the protocol. Each connection is ~~essentially~~ independent and attempts to maximize its own individual traffic rate. For multi-connection tests, a single client process would replicate the connection setup and test procedure multiple times (once for each flow) to one or more server instances. ~~The~~ A server instance(s) would process each connection independently, as if they were coming from separate clients. It ~~SHALL~~ shall be the responsibility of the client process to manage the inter-related connections logistically as well as aggregate the individual test results into an overall set of performance statistics. Fields in the Setup Request (mcIndex, mcCount, and mcIdent) ~~exist to both~~are used to differentiate and associate the multiple connections that comprise a single test.

The protocol uses UDP transport [RFC0768] with two connection phases (Control and Data). The ~~and has four types of exchanges in two phases. Exchanges~~ exchanges 1 and 2 (see below) constitute the Control phase, while exchanges 3 and 4 constitute the Data phase.

1. Setup Request and Response Exchange: The client requests to begin a test by communicating its protocol version, intended security mode, and datagram size support. The server either confirms

Commenté [MB7]: May cite BBF?

a mis en forme : Surlignage

Commenté [MB8]: The port number can be configured as well. No?

Commenté [MB9]: Assuming multiple address/port number pairs are configured?

Commenté [MB10]: What does that mean in this context?

Commenté [MB11]: Add a pointer to the section where this is defined.

Commenté [MB12]: If a server instance is identified with a host name that resolves to both IPv4/IPv6 addresses, any connecting pacing should be in place? Should this be following HE/RFC 8305?

matching a configuration or rejects the connection request. If the request is accepted, ~~T~~the server
also communicates ~~the~~ an ephemeral port number for use for ~~further~~ subsequent communication
~~when accepting the client's request~~.

2. Test Activation Request and Response: the client composes a request conveying parameters such as ~~the~~ a testing direction, ~~the~~a 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 number) 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 number ~~combination~~ pinhole entry after ~~the~~a firewall's configured idle traffic timeout.

3. Test Stream Transmission and Measurement Feedback Messages: Testing proceeds with one ~~end-point~~endpoint sending Load PDUs and the
other ~~end-point~~endpoint receiving the Load PDUs and sending frequent
status messages to communicate status and transmission conditions there. The data in the feedback messages, whether received from the client or when being sent to the client, is input to a load rate adjustment algorithm at the server which controls future sending rates at either end. The choice to locate the load rate adjustment 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 exchange to stop the test by setting a STOP indication in its outgoing Load PDUs or Status Feedback messages. After being received, the client acknowledges it by also setting a STOP indication in its outgoing Load PDUs or Status Feedback messages. A graceful connection termination at each end then follows. Since the Load PDUs and Status Feedback messages are used, this exchange is considered a sub-exchange of 3. If the Test traffic stops or the communication path fails, the client assumes that the firewall will close the address/port number
combination after the firewall's configured idle traffic timeout.

5. Both the client and server react to unexpected interruptions in the Control and Data ~~phase~~phases. Watchdog timers limit the time a
server or client will wait before stopping all traffic and terminating a test.

The following is an example exchange of control and measurement PDUs for both a downstream and upstream UDP Speed Test (always client initiated):

Commenté [MB13]: Is there a need to provision an alternate/preferred address?

Commenté [MB14]: Is this always the case? Only one port number is communicated?

Commenté [MB15]: May be worth to say explicitly that this is if the request was accepted.

Commenté [MB16]: Should be elaborated.

Commenté [MB17]: May be cite RFC6887 to exemplify how these can be opened

Commenté [MB18]: I guess you meant in the same network, not same device. I would clarify

Commenté [MB19]: You may also remind that APP behavior is assumed to be followed by these NAT (REQ-2 from [RFC4787])

Commenté [MB20]: Why only firewalls here? What about NATs? Refer to RFC 4787 - Network Address Translation (NAT) Behavioral Requirements for Unicast UDP

Commenté [MB21]: Is this also used to keep the pinhole/mapping in on-path statefull devices?

a mis en forme : Surlignage

Commenté [MB22]: Idem as for previous comment about NATs above.

Commenté [MB23]: Add those to the terminology section

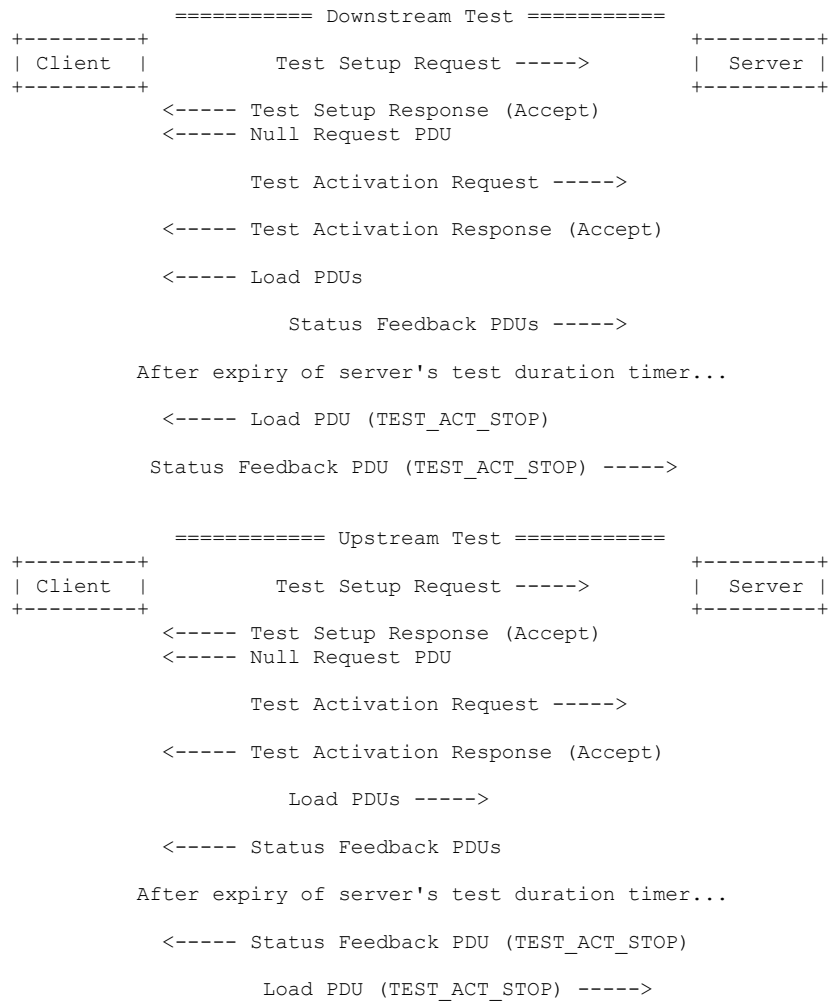


Figure 1: Successful UDPSTP ~~message~~ Message ~~exchanges~~ Exchanges ~~and processes~~

4. Parameters, Security Operations, and Optional Checksum

4.1. Parameters and Definitions

Refer to Section 4 of [RFC9097] ~~For~~ for an overview ~~Parameters~~ of parameters related to the Maximum IP-Layer Capacity Metric and Method~~, please see Section 4 of [RFC9097]~~.

4.2. Security Mode Operations

Commenté [MB24]: The flow is difficult to follow given that the message/attributes are not introduced yet.

Commenté [MB25]: Should there be parameters to instruct whether fragmentation is disabled, path MTU, number of connection retries?

There are four security modes of operation:

1. An OPTIONAL Unauthenticated mode for all messages. This mode, intended for a lab or secured environment, SHALL only be allowed when all other modes requiring authentication (or Partial Encryption) are blocked or unavailable for use.

2. A REQUIRED mode with authentication during the Control phase~~:~~ (Test Setup and Test Activation exchanges)~~:~~. When this mode is available, the ~~OPTIONAL~~ optional unauthenticated mode SHALL NOT be simultaneously available.

3. An OPTIONAL mode with the additional authentication of the Status Feedback messages during the Data phase. When this mode is available, the ~~OPTIONAL~~ optional unauthenticated mode SHALL NOT be simultaneously available.

4. An OPTIONAL mode that adds encryption, prior to authentication, of the Control phase exchanges and the Status Feedback messages. When this mode is available, the ~~OPTIONAL~~ optional unauthenticated mode SHALL NOT be simultaneously available.

The requirements ~~below~~ discussed hereafter refer to the PDUs in the sections that follow, primarily the authMode, keyId, authUnixTime, initVector, and authDigest fields.

The roles in this section have been generalized so that the requirements for the PDU sender and receiver can be re-used and referred to elsewhere.

4.2.1. Mode 0: ~~OPTIONAL~~ Optional Unauthenticated ~~mode~~Mode

~~In the OPTIONAL Unauthenticated mode~~In this mode, all PDU senders SHALL set the keyId ~~field~~, authUnixTime ~~field~~, the initVector ~~field~~, and the authDigest fields of all related packets to zero.

When the ~~OPTIONAL~~ Unauthenticated mode of operation is available, all other modes requiring authentication (or Partial Encryption) SHALL be blocked or unavailable for use. The availability of unauthenticated and authenticated modes SHALL be considered mutually exclusive.

Any errors (configuration mis~~s-~~match between clients and servers) found in ~~the~~ a Test Setup exchange or ~~the~~ a Test Activation exchange SHOULD result in silent rejection (no further packets sent on the address/port pairs). The exception is when the testing hosts ~~have been~~is configured for troubleshooting control phase failures and rejection messages will aid in the process.

4.2.2. Mode 1: ~~REQUIRED~~ Required Authenticated ~~mode~~Mode

In ~~the REQUIRED~~this ~~Authenticated~~ mode, the client and the server SHALL

Commenté [MB26]: If this is with the scope set in the previous sentence, why should we have this guard?

Commenté [MB27]: Isn't this conflicting with REQUIRED mention?

Commenté [MB28]: Inappropriate use of normative language

Commenté [MB29]: I'm afraid we are mixing support vs actual use.

Commenté [MB30]: Inappropriate use of normative language

Commenté [MB31]: Idem as above

Commenté [MB32]: Why is this set as optional?

Commenté [MB33]: Inappropriate use of normative language

Commenté [MB34]: Idem as similar comment above

Commenté [MB35]: Add explicit refs to the sections where these are defined.

Commenté [MB36]: ?

Commenté [MB37]: This is redundant with what was mentioned above.

Commenté [MB38]: Is any reason code enclosed there?

be configured to use one of a number of shared secret keys, designated via the keyId field.

During the Control phase, the sender SHALL read the current time and populate the authUnixTime field, then calculate the authDigest field of the entire PDU (with the initVector and authDigest fields set to all zeroes) according to [RFC6234] and send the packet to the receiver. The value in the authUnixTime field is a 32-bit ~~time stamp~~timestamp and a ~~10 second~~10-second tolerance window (+/- 5 seconds) SHALL be used by the receiver to distinguish a subsequent replay of a PDU.

Upon reception, the receiver SHALL validate the message PDU for correct length, validity of authDigest, immediacy of authUnixTime, and expected formatting (PDU-specific fields are also checked, such as protocol version). Validation of the authDigest ~~will~~ requires that it ~~be~~ will be extracted from the PDU and the field zeroed prior to the HMAC calculation used for comparison.

If the validation fails, the receiver SHOULD NOT continue with the Control phase and implement silent rejection (no further packets sent on the address/port pairs). The exception is when the testing hosts have been configured for troubleshooting Control phase failures and rejection messages will aid in the process.

If the validation succeeds, the receiver SHALL continue with the Control phase and compose a successful response or a response indicating the error conditions identified (if any).

This process SHALL be executed for the request and response in the Test Setup exchange, including the Null Request~~,~~ (Section 5) and the Test Activation exchange (Section 6).

4.2.3. Mode 2: ~~OPTIONAL~~ Optional Authenticated M~~m~~ode for Data ~~phase~~Phase

This mode incorporates Authenticated mode 1. When using the ~~OPTIONAL~~optional authentication during the Data Phase, authentication SHALL also be applied to the Status Feedback PDU (Section xX). The client sends the Status PDU in a downstream test, and the server sends it in an upstream test.

The Status PDU sender SHALL read the current time and populate the authUnixTime field, then calculate the authDigest field of the entire Status PDU (with the initVector and authDigest field set to all zeroes) and send the packet to the receiver.

Upon reception, the receiver SHALL validate the message PDU for correct length, validity of authDigest, immediacy of authUnixTime, and expected formatting (PDU-specific fields are also checked, such as protocol version). Validation of the authDigest will require that it be extracted from the PDU and the field zeroed prior to the HMAC calculation used for comparison.

If the authentication validation fails, the receiver SHALL ignore the message. If the watchdog timer expires (due to successive failed

Commenté [MB39]: How this is done?

Commenté [MB40]: From where?

Commenté [MB41]: Which section in that RFC?

Commenté [MB42]: Any requirement on the time sync?

Commenté [MB43]: Add authoritative ref

Commenté [MB44]: Please be consistent as both Data Phase and Data phase are used in the doc

Commenté [MB45]: To be completed

Commenté [MB46]: From where?

validations), the test session will prematurely terminate (no further load traffic SHALL be transmitted).

If this optional mode has not been selected, then the keyId ~~field~~, authUnixTime ~~field~~, initVector ~~field~~, and authDigest fields of the Status PDU (~~see~~ Section 7.2) SHALL be ~~populated with all~~set to zeroes.

4.2.4. Mode 3: ~~OPTIONAL~~ Optional Partial Encryption of Control and Status

This mode incorporates ~~authentication~~ Authentication mode 2 after encryption of all fields prior to the authMode field. This is done for all Control and Status Feedback messages. The encryption algorithm only provides encryption and relies on the existing authentication mechanism to provide integrity protection. This mode re-uses several protocol fields, which is reasonable since both authentication and encryption are provided (the field format is presented in later sections).

When using the ~~OPTIONAL~~ optional Partial Encryption, the process SHALL be applied to the Test Setup Request, the Test Setup Response, the Null Request (if applicable), the Test Activation Request, the Test Activation Response, and the Status PDU. The client sends the Status PDU in a downstream test, and the server sends it in an upstream test.

In the ~~OPTIONAL~~ optional Partial Encryption mode, the client and the server SHALL be configured to use one of a number of shared secret keys (see keyId).

The following encryption specifications SHALL be used for this mode:

1. Advanced Encryption Standard, AES, according to Federal Information Processing Standards Publication 197 [FIPS-197]

2. Cipher Block Chaining (CBC) [CBC]

3. Key size of 128 bits (fixed block size of 128 bits)

The encrypted portion of each PDU SHALL contain the padding required to maintain a multiple of the AES CBC block size of 16 octets. As such, any library functions used for encryption and decryption SHALL have padding disabled (to maintain an equal encrypted and unencrypted length). In OpenSSL, for example, this can be accomplished via "EVP_CIPHER_CTX_set_padding(ctx,0)".

The sender SHALL populate the initVector field with a random ~~16~~ 16-octet~~octet~~ Initialization Vector (IV). The IV, in conjunction with the shared secret key, SHALL be used to encrypt the header up to, but not including, the authMode field. The shared secret key is designated via the keyId field. The sender SHALL then read the current time and populate the authUnixTime field and then calculate (and populate) the

Commenté [MB47]: May explain why Authenticated encryption schemes are not used instead.

Commenté [MB48]: I don't parse this.

Commenté [MB49]: Better provide a section pointer

a mis en forme : Surlignage

Commenté [MB50]: Any guidance/examples about how such provisioning can be done?

Commenté [MB51]: Is there any key rotation expected to be in place?

authDigest, for the entire PDU, in the same manner as specified with Authentication modesmode 1 and 2. Finally, the sender SHALL send the packet with partially encrypted PDU to the receiver.

Upon reception, the receiver SHALL validate the message PDU for correct length, validity of authDigest, and immediacy of authUnixTime. Validation of the authDigest will require that it be extracted from the PDU and the field zeroed prior to the HMAC calculation used for comparison. If the PDU validation succeeds, the receiver SHALL then decrypt the initial portion of the PDU using the included IV and shared key designated by the keyId. Finally, the PDU-specific fields that control the test are validated and processed.

If the PDU validation fails in the Control phase, the receiver SHOULD NOT continue with current exchange and implement silent rejection (no further packets sent on the address/port pairs). The exception is when the testing hosts have been configured for troubleshooting Control phase failures and rejection messages will aid in the process.

If the validation succeeds in the Control phase, the receiver SHALL continue with the current exchange and compose a successful response or a response indicating the error conditions identified. The response PDU SHALL be encrypted and authenticated as described above using a new random 16 octet16-octet Initialization Vector (IV). The packet
with partially encrypted PDU SHALL be sent back to the originator.

This process SHALL be executed for the request and response in the Test Setup exchange, including the Null Request, (Section 5) and the Test Activation exchange (Section 6).

If the PDU validation fails for a Status PDU, the receiver SHALL ignore the message. If the watchdog timer expires (due to successive failed validations), the test session will prematurely terminate (no further load traffic SHALL be transmitted).

Commenté [MB52]: MUST?

4.3. Key Management

Section 2 of [RFC7210] specifies a conceptual database for long-lived cryptographic keys. The database is implemented as a plaintext table, to allow text editor maintenance of the key table. The key table SHALL be used with the REQUIRED required and optional authentication modes and the OPTIONAL authentication mode (using the same key). The same key table and key SHALL also be used with the OPTIONAL optional Partial Encryption
mode, when utilized.

The Key table SHALL have (at least) the following fields, referring to Section 2 of [RFC7210]:

Commenté [MB53]: Use consistent wording: Key table va key table

* AdminKeyName

* LocalKeyName

* AlgID

* Key

* SendLifetimeStart

* SendLifetimeEnd

* AcceptLifetimeStart

* AcceptLifetimeEnd

The LocalKeyName SHALL be determined from the corresponding keyId field in the PDUs that follow in a message.

4.4. Firewall Configuration

Normal firewall configuration allows a host to open a bidirectional connection using unique source and destination addresses and port numbers by sending a packet using that set of 4-tuple values. The client's interaction with its firewall depends on this configuration.

The firewall at the server side MUST be configured with an open pinhole for the server IP address and ~~well-known~~ the UDP port number on which the ~~of the~~ server listens.

Assuming that the firewall administration at the server does not allow an open UDP ephemeral port range, then the server MUST send a Null Request to the client from the ephemeral port number communicated to the client in ~~the~~ a Test Setup Response. The Null Request may not reach the client as ~~:~~ it may be discarded by the client's firewall or any other on-path stateful rewriting device.

If the server firewall administration allows an open UDP ephemeral Port range, then the Null Request is not strictly necessary. However, the availability of an open port range policy cannot be assumed.

4.5. Optional Checksum

All of the PDUs exchanged between the client and server support a header checksum that covers the various fields in the PDU (excluding the Payload Content of the Load PDU). This checksum is intended for environments where UDP data integrity may be uncertain. This includes situations where the standard UDP checksum is disabled or a nonstandard network API is in use (things typically done to improve performance on low-end devices). The calculation is the same as the 16-bit one's complement Internet checksum used in the IPv4 packet header.

If a PDU sender is populating the checkSum field, it SHALL do so after the PDU is built but prior to any authentication or encryption processing (i.e., all fields starting with authMode are zeroed). The PDU receiver SHALL subsequently verify the PDU checksum whenever checksum processing has been configured. Verification requires that all fields starting with authMode are zeroed prior to the checksum calculation used to verify the PDU.

Commenté [MB54]: Why only firewall, not NAT/CGN?

Commenté [MB55]: Don't know what this is about

a mis en forme : Surlignage

a mis en forme : Surlignage

Commenté [MB56]: What is the scope of the checksum? I guess this does not cover the header as this may change on path.

Commenté [MB57]: Cite rfc791#section-3.1

Because of its redundancy when authentication is being used, it is OPTIONAL for a PDU sender to utilize the checkSum field whenever the authDigest field ~~will also be~~is also utilized. However, because authentication is not applicable to the Load PDU, the checkSum field SHALL be utilized by the sender whenever UDP data integrity may be uncertain (as outlined above).

5. Test Setup Request and Response

All messages defined in this section SHALL use UDP transport.

5.1. Client Generates Test Setup Request

The client SHALL begin the Control phase exchanges by sending a Test Setup Request message to the server's ~~(well-known)~~configured control port number.

The client SHALL simultaneously start a test initiation timer so that if the Control phase fails to complete Test Setup and Test Activation exchanges in the allocated time, the client ~~software~~ SHALL exit (close the UDP socket and indicate an error message to the user). Lost messages SHALL NOT be retransmitted. The test initiation timer MAY reuse the test termination timeout value.

As of version 9, the watchdog timeout is configured as a ~~1~~ 1-~~second~~second interval to trigger a warning message that the received traffic has stopped. The test termination timeout is based on the watchdog interval, and implements a wait time of 2 additional seconds before triggering a non-graceful termination.

The Setup Request/Response message PDU SHALL be ~~organized~~ structured as follows:

<CODE BEGINS>

```
//
// Control header for UDP payload of Setup Request/Response PDUs
//

struct controlHdrSR {
#define CHSR_ID 0xACE1
        uint16_t controlId;   // Control ID
#define PROTOCOL_VER 20
        uint16_t protocolVer; // Protocol version
        uint8_t mcIndex;    // Multi-connection index
        uint8_t mcCount;    // Multi-connection count
        uint16_t mcIdent;   // Multi-connection identifier
#define CHSR_CREQ_NONE     0
#define CHSR_CREQ_SETUPREQ 1  // Setup request
#define CHSR_CREQ_SETUPRSP 2  // Setup response
        uint8_t cmdRequest;   // Command request
#define CHSR_CRSP_NONE     0  // (used with request)
#define CHSR_CRSP_ACKOK    1  // Acknowledgment
#define CHSR_CRSP_BADVER   2  // Bad version
#define CHSR_CRSP_BADJS    3  // Jumbo setting mismatch
#define CHSR_CRSP_AUTHNC   4  // Auth. not configured
#define CHSR_CRSP_AUTHREQ  5  // Auth. required
```

Commenté [MB58]: This is already mention in previous section. I would delete the sentence.

Commenté [MB59]: What if multiple IP addresses are available? What if the client is multihomed? Should this be done for each network attachement? Should HE be in place?

Commenté [MB60]: How these are detected?

Commenté [MB61]: But version 20 is also requested to be registered.

Commenté [MB62]: Use the same for other code snippets. Thanks

```
#define CHSR_CRSP_AUTHINV  6  // Auth. (mode) invalid
#define CHSR_CRSP_AUTHFAIL 7  // Auth. failure
#define CHSR_CRSP_AUTHTIME 8  // Auth. time invalid
#define CHSR_CRSP_NOMAXBW  9  // Max bandwidth required
#define CHSR_CRSP_CAPEXC   10 // Capacity exceeded
#define CHSR_CRSP_BADTMTU  11 // Trad. MTU mismatch
#define CHSR_CRSP_MCINVPAR 12 // Multi-conn. invalid params
#define CHSR_CRSP_CONNFAIL 13 // Conn. allocation failure
        uint8_t cmdResponse;  // Command response
#define CHSR_USDIR_BIT 0x8000  // Upstream direction bit
        uint16_t maxBandwidth; // Required bandwidth
        uint16_t testPort;     // Test port on server
#define CHSR_JUMBO_STATUS    0x01
#define CHSR_TRADITIONAL_MTU 0x02
        uint8_t modifierBitmap; // Modifier bitmap
        uint8_t reserved1;      // (reserved for alignment)
        uint16_t checkSum;      // Header checksum
        uint16_t reserved2;     // (reserved for alignment)
        //
        uint8_t padding[12]; // Padding for encryption
        // ========== Encryption ends here ==========
#define AUTHMODE_0 0 // Mode 0: Unauthenticated
#define AUTHMODE_1 1 // Mode 1: Authenticated Control
#define AUTHMODE_2 2 // Mode 2: Authenticated Control+Status
#define AUTHMODE_3 3 // Mode 3: Encrypted+Auth Control+Status
        uint8_t authMode;      // Authentication mode
        uint8_t keyId;         // Key ID in shared table
        uint16_t reserved1a;   // (reserved for alignment)
        uint32_t authUnixTime; // Authentication time stamp
#define AUTH_IV_LENGTH 16 // Initialization Vector length
        uint8_t initVector[AUTH_IV_LENGTH]     // IV
#define AUTH_DIGEST_LENGTH 32 // SHA-256 digest length
        uint8_t authDigest[AUTH_DIGEST_LENGTH] // HMAC
};
```

<CODE ENDS>

Figure 2: Test Setup PDU

Commenté [MB63]: I would position this after Figure 3 is explained

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

0 1 2 3

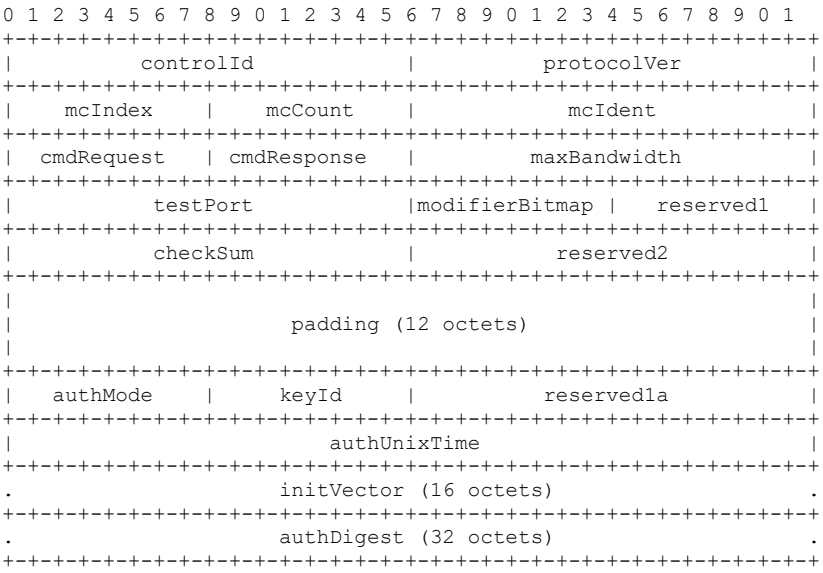


Figure 3: Test Setup PDU Layout

Additional details regarding the Setup Request and Response fields are as follows:

mcIndex: The index ~~(0,1,2,...)~~ of a connection relative to all connections that make up a single test. It is used to differentiate separate connections within a multi-connection test.

mcCount: The total count of attempted connections.

mcIdent: A pseudorandom non-zero identifier (via RNG, source port number, ...) that is common to all connections of a single test. It is used by clients/servers to associate separate connections within a multi-connection test.

maxBandwith: When this field is non-zero, it is a specification of the maximum ~~bit rate~~bitrate that the client expects to send or receive during the requested test. The server compares this value to its currently available configured limit for test admission control. This field MAY ~~optionally~~ be used for rate-limiting the maximum rate that the server should attempt. The CHSR_USDIR_BIT bit is set to 0 by default to indicate "downstream" and has to be set to 1 to indicate "upstream".

testPort: The UDP ephemeral port number on the server that the client ~~SHALL~~ has to use for the Test Activation Request and subsequent Load or Status PDUs.

xx

modifierBitmap: There are two bits currently assigned in this bitmap:

Commenté [MB64]: I guess this is monotonically set (by +1) with wrap around when max is reached?

Commenté [MB65]: I would naively increase the size of the mcIndex and decrease this one

a mis en forme : Français (France)

Commenté [MB66]: Redundant with MAY

Commenté [MB67]: This is present in Figure 2, but not figure 3. Is that intended?

Commenté [MB68]: Inappropriate use of normative language

Commenté [MB69]: Should be described in the order they appear in the figure: cmdRequest | cmdResponse

CHSR_JUMBO_STATUS Above a sending rate of 1Gbps, allow datagram sizes that result in Jumbo Frames (with a max IP packet size of 9000 bytes). Up to a sending rate of 1Gbps, or for all sending rates if CHSR_JUMBO_STATUS is not set, datagram sizes SHALL NOT produce an IP packet size greater than 1250 bytes (unless CHSR_TRADITIONAL_MTU is also set).

CHSR_TRADITIONAL_MTU Allow datagram sizes, at any sending rate, that can result in a Traditional IP packet size of 1500 bytes. Effectively increasing the default non-Jumbo maximum from 1250 bytes to 1500 bytes.

Other bit positions ~~are currently undefined~~are unassigned. A ~~new~~ registry ~~will be~~is used ~~needed~~ for modifierBitmap assignments; see the IANA Considerations ~~section~~Section.

checkSum: An ~~OPTIONAL~~ optional checksum of the entire PDU. The calculation is done with the checkSum field, and all fields starting with authMode, set to zero.

Commenté [MB70]: Double check

authMode: The authMode field currently has four values assigned:

AUTHMODE_0: ~~OPTIONAL~~ Optional Unauthenticated mode

AUTHMODE_1: ~~REQUIRED~~ Required Authentication for Control phase

AUTHMODE_2: ~~OPTIONAL~~ Optional Authentication for Control and Data phase
(Status Feedback PDU only)

AUTHMODE_3: ~~OPTIONAL~~ Optional Partial Encrypted mode

60 through 63 range is reserved ~~plus, a range of values~~ for experimentation: ~~60 through 63~~.

A ~~new~~ registry ~~will be~~is defined to record assigned ~~needed for~~ mode values; see the IANA Considerations ~~S~~section.

keyId: This is a localKeyName, the numeric key identifier for a key in the shared key table.

authUnixTime: A 32-bit ~~time stamp~~timestamp of the current time since the Unix Epoch on January 1st, 1970 at UTC.

initVector: This field contains the ~~16 octet~~16-octet random ~~Initialization Vector (~~IV~~)~~ used for Partial Encryption mode. A new random ~~16~~ 16-octe~~octet~~ IV SHALL be used each time encryption is performed.

authDigest: This field contains the ~~32~~ 32-octet~~octet~~ HMAC-SHA256 hash that

covers the entire PDU.

5.2. ~~Server Processes~~ Processing Test Setup Request and ~~Generates~~ Generating Response by Servers

This section describes the processes at the server to ~~evaluate~~ handle the Test Setup Request and determine the next steps.

5.2.1. Test Setup Request Processing - Rejection

When the server receives the Setup Request, it SHALL:

* verify the size of the Setup Request message and if correct interrogate the authMode field

* if operating in one of the Authenticated modes, validate the Setup Request message by checking the authDigest as prescribed in Section 4.2.2

* if operating in the Partial Encryption mode, use the available keyId and IV to decrypt the Setup Request message up to the authMode field using the method prescribed in Section 4.2.4

~~and t~~Then, the server ~~proceed to~~ evaluates the other fields in the protocol header, such as the protocol version, the Control ID (to validate the type of message), the maximum Bandwidth requested for the test, and the modifierBitmap for use of options such as Jumbo datagram status and ~~Traditional~~ conventional MTU (1500 bytes).

If the client has selected options for:

* Jumbo datagram support (modifierBitmap),

* ~~Traditional~~ Conventional MTU (modifierBitmap),

* Authentication mode (authMode)

that do not match the server configuration, the server MUST reject the Setup Request.

If the Setup Request must be rejected, the conditions below determine whether the server sends a response:

* In Authenticated modes, if the authDigest is valid, a Test Setup Response SHALL be sent back to the client with a corresponding command response value indicating the reason for the rejection. If operating in the Partial Encryption mode, the server SHALL ~~follow the requirements of~~proceed per Section 4.2.4, else the server SHALL ~~follow the requirements of~~proceed per Section 4.2.2.

* In Authenticated modes, if the authDigest is invalid, then the Test 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

Commenté [MB71]: An obvious check is to also to check whether this is a request

Commenté [MB72]: I don't parse this

Commenté [MB73]: Distinct defaults may be assumed for IPv4/IPv6 PMTU? 576 bytes ipv4/1280 bytes ipv6.

Commenté [MB74]: Where is this set?

troubleshooting.

* If Unauthenticated mode is selected, the Test Setup Request SHALL fail silently.

The additional, non-authentication circumstances when a server SHALL NOT communicate the appropriate Command Response code for an error condition (fail silently) are when:

1. the Setup Request PDU size is not correct,

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 the server replies to ~~the~~ a Test Setup Request message, the Test Setup Response PDU is structured identically to the Request PDU and SHALL retain the original values received in it, with the following exceptions:

* The cmdRequest field is set to CHSR_CREQ_SETUPRSP, indicating a response.

* The cmdResponse field is set to an error code (starting at CHSR_CRSP_BADVER), indicating the reason for rejection. If cmdResponse indicates a bad protocol version (CHSR_CRSP_BADVER), the protocolVer field is also updated to indicate the current expected version.

* The PDU is encrypted up to the authMode field using a new random IV, if doing encryption.

* The authUnixTime field is updated to the current time and the authDigest is recalculated, if doing authentication.

5.2.2. Test Setup Request Processing - Acceptance

If the server finds that the Setup Request matches its configuration and is otherwise acceptable, the server SHALL initiate a new connection to receive the Test Activation Request from the client, using a new UDP socket allocated from the UDP ephemeral port range. This new socket will also be used for the subsequent Load and Status PDUs that are part of testing (with the port number communicated back to the client in the Test Setup Response). Then, the server SHALL start a watchdog timer (to terminate the new connection if the client goes silent) and SHALL send the Test Setup Response back to the client.

When the server replies to the Test Setup Request message, the Test Setup Response PDU is structured identically to the Request PDU and SHALL retain the original values received in it, with the following exceptions:

* The cmdRequest field is set to CHSR_CREQ_SETUPRSP, indicating a

Commenté [MB75]: Is there any provision for diagnostic payload?

Commenté [MB76]: That is concretely?

Commenté [MB77]: How the check is done?

Commenté [MB78]: Is there an IANA registry to track those?

Commenté [MB79]: We need a section with a summary of errors and the conditions under which they are sent

Commenté [MB80]: Why not simply replying using the existing flow? Or send a message to the client to issue a new connection?

In doing so, we soften much issues related to NAT/FW

Commenté [MB81]: Is there any default or recommended value?

response.

* The cmdResponse field is set to CHSR_CRSP_ACKOK, indicating an acknowledgment.

* The testPort field is set to the ephemeral port number to be used for the client's Test Activation Request and all subsequent communication.

* The PDU is encrypted up to the authMode field using a new random IV, if doing encryption.

* The authUnixTime field is updated to the current time and the authDigest is recalculated, if doing authentication.

Finally, the new UDP connection associated with the new socket and port are made ready, and the server awaits further communication there.

To ensure that a server's local firewall will successfully allow packets received for the new ephemeral port, the server SHALL immediately send a Null Request with the corresponding values including the source and destination IP addresses and port numbers. The source port SHALL be the new ephemeral port. This operation allows communication to the server even when the server's local firewall prohibits open ranges of ephemeral ports. The packet is not expected to arrive successfully at the client if the client-side firewall blocks unexpected traffic. If the Null Request arrives at the client, it is a confirmation that further exchanges are possible on the new port-pair (but this is not strictly necessary). Note that there is no response to a Null Request.

a mis en forme : Surlignage

The Null Request message PDU SHALL be organized as follows:

```
//
// Control header for UDP payload of Null Request PDU
//
struct controlHdrNR {
#define CHNR_ID 0xDEAD
        uint16_t controlId;   // Control ID
        uint16_t protocolVer; // Protocol version
#define CHNR_CREQ_NONE    0
#define CHNR_CREQ_NULLREQ 1  // Null request
        uint8_t cmdRequest;  // Command request
#define CHNR_CRSP_NONE 0     // (used with request)
        uint8_t cmdResponse; // Command response
        uint16_t checkSum;   // Header checksum
        //
        uint8_t padding[8]; // Padding for encryption
        // ========== Encryption ends here ==========
        uint8_t authMode;      // Authentication mode
        uint8_t keyId;         // Key ID in shared table
        uint16_t reserved1a;   // (reserved for alignment)
        uint32_t authUnixTime; // Authentication time stamp
        uint8_t initVector[AUTH_IV_LENGTH]     // IV
        uint8_t authDigest[AUTH_DIGEST_LENGTH] // HMAC
};
```

Figure 4: Null Request PDU

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

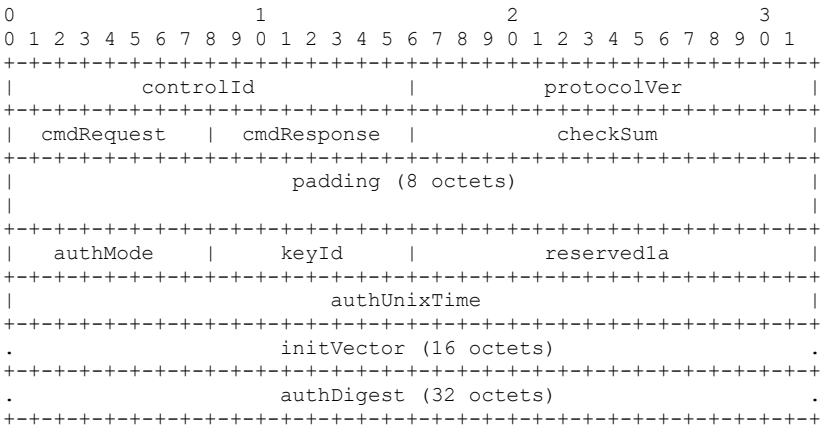


Figure 5: Null Request PDU Layout

Additional details regarding the Null Request fields are as follows:

checkSum: An OPTIONAL checksum of the entire PDU. The calculation is done with the checkSum field, and all fields starting with authMode, set to zero.

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

5.3. Setup Response Processing at the Client

When the client receives the Test Setup Response message, it SHALL:

* verify the size of the Setup Response message and if correct interrogate the authMode field,
* if operating in one of the Authenticated modes, validate the Setup Response message by checking the authDigest as prescribed in Section 4.2.2,
* if operating in the Partial Encryption mode, use the available keyId and IV to decrypt the Setup Response message up to the authMode using the method prescribed in Section 4.2.4,

and then proceed to evaluate the other fields in the protocol, beginning with the protocol version, Control ID (to validate the type of message), and cmdRequest for the role of the message (SHOULD be Test Setup Response).

Commenté [MB82]: ?

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

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.~~

6.1. Client Generates Test Activation Request

Upon a successful setup exchange, the client SHALL compose and send the Test Activation Request to the UDP port number the server communicated in the Test Setup Response (the new ephemeral port, and not the well-known port).

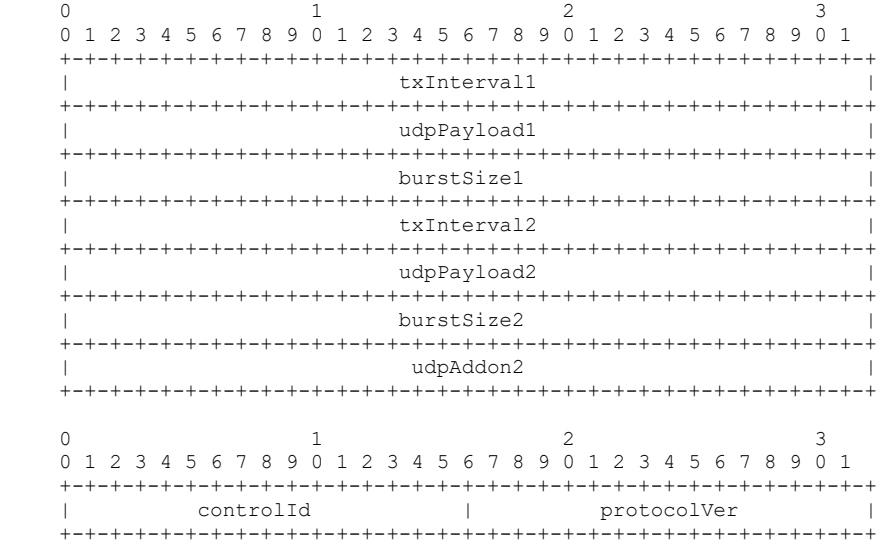
The Test Activation Request/Response message PDU (as well as the included Sending Rate structure) SHALL be organized as follows:

```
//
// Sending rate structure for a single row of transmission parameters
//
struct sendingRate {
        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)
};
//
// Control header for UDP payload of Test Act. Request/Response PDUs
//
struct controlHdrTA {
#define CHTA_ID 0xACE2
        uint16_t controlId;   // Control ID
        uint16_t protocolVer; // Protocol version
#define CHTA_CREQ_NONE      0
#define CHTA_CREQ_TESTACTUS 1 // Test activation upstream
#define CHTA_CREQ_TESTACTDS 2 // Test activation downstream
        uint8_t cmdRequest;   // Command request
#define CHTA_CRSP_NONE     0  // (used with request)
#define CHTA_CRSP_ACKOK    1  // Acknowledgment
#define CHTA_CRSP_BADPARAM 2  // Bad/invalid test params
        uint8_t cmdResponse;  // Command response
        uint16_t lowThresh;   // Low delay variation threshold (ms)
        uint16_t upperThresh; // Upper delay variation threshold (ms)
        uint16_t trialInt;    // Status Feedback/trial interval (ms)
        uint16_t testIntTime; // Test interval time (sec)
```

```
        uint8_t subIntPeriod; // Sub-interval period (sec)
        uint8_t ipTosByte;    // IP ToS byte for testing
#define CHTA_SRIDX_DEF UINT16_MAX // Request default server rate search
        uint16_t srIndexConf;   // Configured Sending Rate Table index
        uint8_t useOwDelVar;    // Use one-way delay, not RTT (BOOL)
        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/Duplicates (BOOL)
#define CHTA_SRIDX_ISSTART 0x01 // Use srIndexConf as starting index
#define CHTA_RAND_PAYLOAD  0x02 // Randomize payload
        uint8_t modifierBitmap; // Modifier bitmap
#define CHTA_RA_ALGO_B   0   // Algorithm B
#define CHTA_RA_ALGO_C   1   // Algorithm C
        uint8_t rateAdjAlgo; // Rate adjust. algorithm
        uint8_t reserved1;   // (reserved for alignment)
        struct sendingRate srStruct; // Sending rate structure
        uint16_t reserved2; // (reserved for alignment)
        uint16_t checkSum;  // Header checksum
        //
        uint8_t padding[4]; // Padding for encryption
        // ========== Encryption ends here ==========
        uint8_t authMode;      // Authentication mode
        uint8_t keyId;         // Key ID in shared table
        uint16_t reserved1a;   // (reserved for alignment)
        uint32_t authUnixTime; // Authentication time stamp
        uint8_t initVector[AUTH_IV_LENGTH]     // IV
        uint8_t authDigest[AUTH_DIGEST_LENGTH] // HMAC
};
```

Figure 6: Test Activation PDU

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



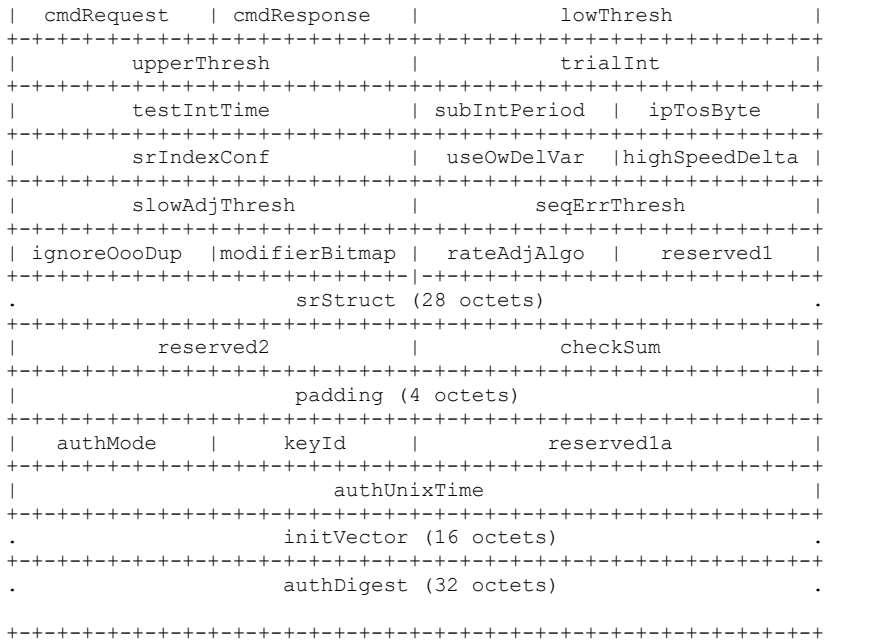


Figure 7: Test Activation PDU Layout

Fields are populated based on default values or command-line options. Authentication and encryption modes follow the same methodology as with the Setup Request and Response.

The content of many of the unique fields in Figures 6 and 7 are defined in Section 4 of [RFC9097] and Appendix A of [RFC9097]. Additional details are given below.

srIndexConf: The requested Configured Sending Rate Table index, used in a Test Activation Request, of the desired fixed or starting sending rate (depending on whether CHTA_SRIDX_ISSTART is cleared or set respectively). Because a value of zero is a valid fixed or starting sending rate index, the field SHALL be set to its maximum (CHTA_SRIDX_DEF) when requesting the default behavior of the server (starting the selected load rate adjustment algorithm at its minimum/ zero index). This SHALL be equivalent to setting srIndexConf to zero and setting the CHTA_SRIDX_ISSTART bit.

modifierBitmap: There are two bits currently assigned in this bitmap:

CHTA_SRIDX_ISSTART Treat srIndexConf as the starting sending rate to be used by the load rate adjustment algorithm

CHTA_RAND_PAYLOAD Randomize the Payload Content beyond the Load PDU header

Other bit positions are currently undefined. A new registry will be needed for modifierBitmap assignments; see the IANA Considerations

section.

srStruct: Sending Rate structure, used by the server in a Test Activation Response for an upstream test, to communicate the (initial) Load PDU ~~transmisstion~~transmission parameters the client SHALL use.
For a Test Activation Request or a downstream test, this structure SHALL be zeroed. Two sets of periodic transmission parameters are available, allowing for dual independent transmitters (to support a high degree of rate granularity). The udpAddon2 field specifies the size of a single Load PDU to be sent at the end of the txInterval2 send sequence, even when udpPayload2 or burstSize2 are zero and result in no transmission of their own.

checkSum: An ~~OPTIONAL~~ optional checksum of the entire PDU. The calculation is
done with the checkSum field, and all fields starting with authMode, set to zero.

6.2. Server Processes Test Activation Request and Generates 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 replies to the Test Activation Request message, the Test Activation Response PDU is structured identically to the Request PDU and SHALL retain the original values received in it unless they are explicitly coerced to a server acceptable value.

6.2.1. Server Rejects or Modifies Request

When evaluating the Test Activation Request, the server MAY allow the client to specify its own fixed or starting send rate via srIndexConf.

Alternatively, the server MAY enforce a maximum limit 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 bit rate, and communicate that maximum to the client in the Test Activation Response. The client can of course choose to end the test, as appropriate.

Other parameters where the server has the OPTION to coerce the client to use values other than those in the Test Activation Request are (grouped by role):

* Load rate adjustment algorithm: lowThresh, upperThresh, useOwDelayVar, highSpeedDelta, slowAdjThresh, seqErrThresh, highSpeedDelta, ignoreOooDup, rateAdjAlgo.

* Test duration/intervals: trialInt, testIntTime, subIntPeriod

* Packet marking: ipTosByte

Coercion is a step toward performing a test with the server-configured values; even though the client might prefer certain values that the server gives the client an opportunity to run a test with

different values than the preferred set. In these cases, the Command Response value SHALL be CHTA_CRSP_ACKOK.

Note that the server also has the option of completely rejecting the request and sending back an appropriate Command Response value (only CHTA_CRSP_BADPARAM currently).

Whether this error response is sent or not depends on the Security mode of operation and the outcome of authDigest validation.

If the Test Activation Request must be rejected (due to the Command Response value being CHTA_CRSP_BADPARAM), and

* In Authenticated modes, if the authDigest is valid, a Test Activation Response SHALL be sent back to the client with a corresponding command response value indicating the reason for the rejection. If operating in the Partial Encryption mode, the server SHALL follow the requirements of Section 4.2.4, else the server SHALL follow the requirements of Section 4.2.2.

* In Authenticated modes, if the authDigest is invalid, then the Test Activation 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.

* If Unauthenticated mode is selected, the Test Activation Request SHALL fail silently.

The additional, non-authentication circumstances when a server SHALL NOT communicate the appropriate Command Response code for an error condition (fail silently) are when:

1. the Test Activation Request PDU size is not correct,

2. the Control ID is invalid, or

3. a directed attack has been detected,

in which case the server will allow Test Activation Requests to terminate silently. Attack detection is beyond the scope of this specification.

6.2.2. Server Accepts Request and Generates Response

When the server sends the Test Activation Response, it SHALL set the Command Response field to CHTA_CRSP_ACKOK

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 (if requested by srIndexConf having been set to CHTA_SRIDX_DEF), OR

* include the transmission parameters from the designated Configured Sending Rate Table index (srIndexConf) of the Sending Rate Table where, if CHTA_SRIDX_ISSTART is set in modifierBitmap, this will be used as the starting rate for the load rate adjustment

algorithm, else it will be considered a fixed rate test.

When generating the Test Activation Response (acceptance) for a downstream test, the server SHALL set all octets of the Sending Rate structure to zero.

If activation continues, the server prepares the new connection for an upstream OR downstream test.

In the case of an upstream test, the server SHALL prepare to use a single timer to send Status PDUs at the specified interval. For a downstream test, the server SHALL prepare to utilize dual timers to send Load PDUs based on

* the transmission parameters directly from the first row of the Sending Rate Table (if requested by srIndexConf having been set to CHTA_SRIDX_DEF), OR

* the transmission parameters from the designated Configured Sending Rate Table index (srIndexConf) of the Sending Rate Table where, if CHTA_SRIDX_ISSTART is set in modifierBitmap, this will be used as the starting rate for the load rate adjustment algorithm, else it will be considered a fixed rate test.

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

6.3. Client Processes Test Activation Response

When the client receives the Test Activation Response, it SHALL:

* If operating in an Authenticated mode, check the message PDU for validity via the authDigest field and acceptable immediacy via the authUnixTime field. If validated, and if operating in Partial Encryption mode, decrypt the PDU using the included IV and shared key designated via keyId. Finally, check the PDU for general formatting, such as protocol version, and any PDU-specific fields that control the test.

When the client receives the (vetted) 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, the client MUST terminate the connection and terminate operation of the current setup procedure.

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 and start a watchdog timer to detect if the server goes quiet.

The connection is now ready for testing.

7. Test Stream Transmission and Measurement Feedback Messages

This section describes the data 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.

7.1. Test Packet PDU and Roles

Testing proceeds with one ~~end-point~~endpoint sending Load PDUs, based on transmission parameters from the Sending Rate Table, and the other ~~end-point~~endpoint sending Status Feedback messages to communicate the traffic conditions at the receiver. When the server is sending Status Feedback messages, they will also contain the latest transmission parameters from the Sending Rate Table that the client SHALL use.

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

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 indirectly based on the feedback in its received Status Feedback 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 Feedback messages (and not referencing a Sending Rate Table directly). 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 rate adjustment algorithm which evaluates measurements taken locally at the Load PDU receiver. When the client is the receiver, the information is communicated to the server via the periodic Status Feedback messages. When the server is the receiver, the information is used directly (although it is also communicated to the client via its periodic Status Feedback messages). This approach is unique to this protocol; it provides the ability to search for the Maximum IP Capacity and specify specific sender behaviors that is absent from other testing tools. Although the algorithm depends on the protocol, it is not part of the protocol per se.

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

1. When there are no impairments present (no sequence errors and low

delay variation), resulting in a sending rate increase.

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

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

Algorithm B also has two modes for increasing/decreasing the sending rate:

* A high-speed mode (fast) to achieve high sending rates quickly, but also back-off quickly when "congestion" is inferred from the measurements. Consecutive feedback intervals that have a supra-threshold count of sequence number anomalies and/or contain an upper delay variation threshold exception in all of the consecutive intervals are sufficient to declare "congestion" within a test. The threshold of consecutive feedback intervals SHALL be configurable with a default of 3 intervals.

* A single-step (slow) 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.

An OPTIONAL load rate adjustment algorithm (designated C) has been defined in [TR-471]. Algorithm C operation and modes are similar to B, but C uses multiplicative increases in the fast mode to reach the Gigabit range quickly and adds the possibility to re-try the fast mode during a test (which improves the measurement accuracy in dynamic or error-prone access, such as radio access).

On the other hand, the test configuration MAY use a fixed sending rate requested by the client, using the field srIndexConf.

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

The Load PDU SHALL be organized as follows (followed by any payload content):

```
//
// Load header for UDP payload of Load PDUs
//
struct loadHdr {
#define LOAD_ID 0xBEEF
        uint16_t loadId; // Load ID
#define TEST_ACT_TEST  0 // Test active
#define TEST_ACT_STOP1 1 // Stop indication used locally by server
#define TEST_ACT_STOP2 2 // Stop indication exchanged with client
        uint8_t testAction;  // Test action
        uint8_t rxStopped;   // Receive traffic stopped indicator (BOOL)
```

```
        uint32_t lpduSeqNo;  // Load PDU sequence number
        uint16_t udpPayload; // UDP payload (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
        uint16_t rttRespDelay;  // Response delay for RTT (ms)
        uint16_t checkSum; // Header checksum
};
```

Figure 8: Load PDU

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

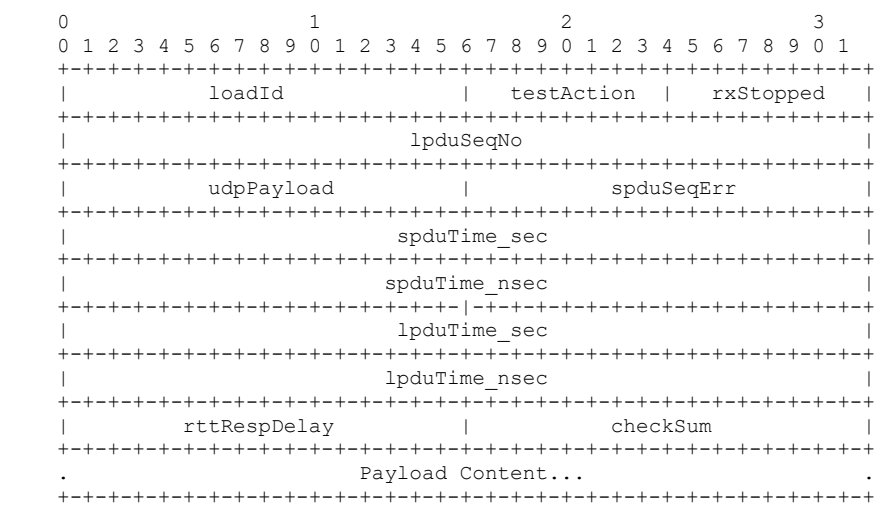


Figure 9: Load PDU Layout

Specific details regarding Load PDU fields are as follows:

testAction: Designates the current test action as either TEST_ACT_TEST (testing in progress), TEST_ACT_STOP1 (first phase of graceful termination, used locally by server), or TEST_ACT_STOP2 (second phase of graceful termination, sent by server and reciprocated by client). See Section 8 for additional information on test termination.

rxStopped: A boolean (0 or 1) used to indicate to the remote end-point that local receive traffic (either Load or Status PDUs) has stopped. All outgoing Load or Status PDUs SHALL continue to assert this indication until traffic is received again, or the test is terminated. The time threshold to trigger this condition is expected to be a reasonable fraction of the watchdog timeout (a default of one second is recommended).

lpduSeqNo: Load PDU sequence number (starting at 1). Used to determine loss, out-of-order, and duplicates.

udpPayload: The total payload size of the UDP datagram including the Load PDU message header and Payload Content (i.e., what the UDP socket read function would return). This field allows the Load PDU receiver to maintain accurate receive statistics if utilizing receive truncation (only requesting the Load PDU message header octets from the protocol stack).

spduSeqErr: Status PDU loss count, as seen by the Load PDU sender. This is determined by the Status PDU sequence number (spduSeqNo) in the most recently received Status PDU. Used to communicate to the Load PDU receiver that return traffic (in the unloaded direction) is being lost.

spduTime_sec/spduTime_nsec: A copy of the most recent spduTime_sec/ spduTime_nsec from the last Status PDU received. Used for RTT measurements made by the Load PDU receiver.

lpduTime_sec/lpduTime_nsec: The local send time of the Load PDU. Used for one-way delay variation measurements made by the Load PDU receiver.

rttRespDelay: RTT response delay, used to "adjust" raw RTT. On the Load PDU sender, it is the number of milliseconds from reception of the most recent Status PDU (when the latest spduTime_sec/ spduTime_nsec was obtained) to the transmission of the Load PDU (where the previously obtained spduTime_sec/spduTime_nsec is returned). When the Load PDU receiver is calculating RTT, by subtracting the copied Status PDU send time (in the Load PDU) from the local Load PDU receive time, this value is subtracted from the raw RTT to correct for any response delay due to Load PDU scheduling.

checkSum: An OPTIONAL checksum of only the Load PDU header. The checksum does not cover the Payload Content. The calculation is done with the checkSum field set to zero.

Payload Content: All zeroes, all ones, or a pseudorandom binary sequence.

7.2. Status PDU

The Load PDU receiver SHALL send a Status PDU to the sender during a test at the configured feedback interval, after at least one Load PDU has been received (when there is something to provide status on). In test scenarios with long delays between client and server, it is possible for the Status PDU send timer to fire before the first Load PDU arrives. In these cases, the Status PDU SHALL NOT be sent.

The watchdog timer at the Load PDU sender SHALL be reset each time a Status PDU is received. See non-graceful test stop in Section 8 for handling the watchdog timeout expiration at each ~~end-point~~endpoint.

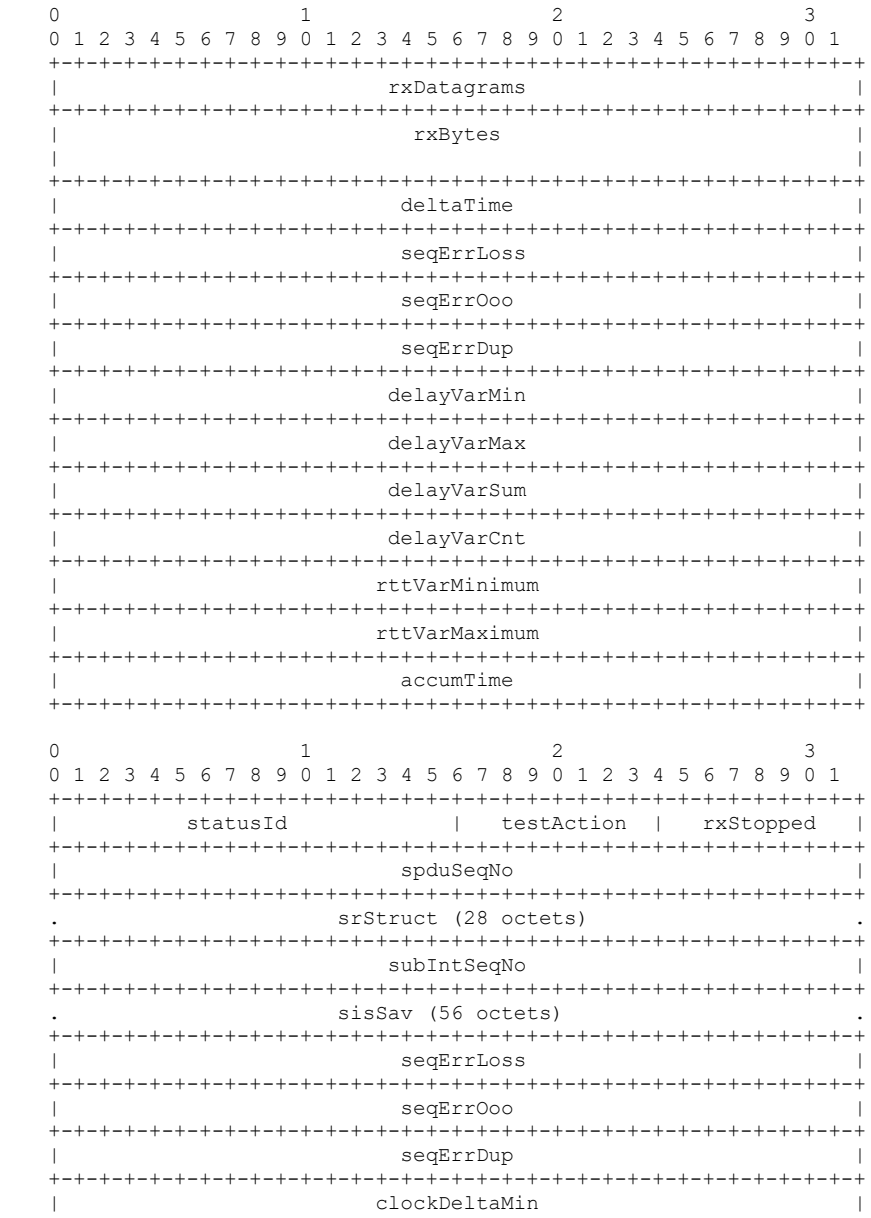
The Status Feedback message PDU (as well as the included Sub-Interval Statistics structure) SHALL be organized as follows:

```
//
// Sub-interval statistics structure for received traffic information
//
struct subIntStats {
        uint32_t rxDatagrams; // Received datagrams
        uint64_t rxBytes;     // Received bytes (64 bits)
        uint32_t deltaTime;   // Time delta (us)
        uint32_t seqErrLoss;  // Loss sum
        uint32_t seqErrOoo;   // Out-of-Order sum
        uint32_t seqErrDup;   // Duplicate sum
        uint32_t delayVarMin; // Delay variation minimum (ms)
        uint32_t delayVarMax; // Delay variation maximum (ms)
        uint32_t delayVarSum; // Delay variation sum (ms)
        uint32_t delayVarCnt; // Delay variation count
        uint32_t rttMinimum;  // Minimum round-trip time (ms)
        uint32_t rttMaximum;  // Maximum round-trip time (ms)
        uint32_t accumTime;   // Accumulated time (ms)
};
//
// Status feedback header for UDP payload of status PDUs
//
struct statusHdr {
#define STATUS_ID 0xFEED
        uint16_t statusId;  // Status ID
        uint8_t testAction; // Test action
        uint8_t rxStopped;  // Receive traffic stopped indicator (BOOL)
        uint32_t spduSeqNo; // Status PDU sequence number
        struct sendingRate srStruct; // Sending rate structure
        uint32_t subIntSeqNo;        // Sub-interval sequence number
        struct subIntStats sisSav;   // Sub-interval saved stats
        uint32_t seqErrLoss;    // Loss sum
        uint32_t seqErrOoo;     // Out-of-Order sum
        uint32_t seqErrDup;     // Duplicate sum
        uint32_t clockDeltaMin; // Clock delta minimum (ms)
        uint32_t delayVarMin;   // Delay variation minimum (ms)
        uint32_t delayVarMax;   // Delay variation maximum (ms)
        uint32_t delayVarSum;   // Delay variation sum (ms)
        uint32_t delayVarCnt;   // Delay variation count
#define STATUS_NORTT UINT32_MAX // No RTT data/value
        uint32_t rttMinimum;    // Minimum round-trip time sampled (ms)
        uint32_t rttVarSample;  // Last round-trip time sample (ms)
        uint8_t delayMinUpd;    // Delay minimum(s) updated (BOOL)
        uint8_t reserved1;      // (reserved for alignment)
        uint16_t checkSum;      // Header checksum
        uint32_t tiDeltaTime;   // Trial interval delta time (us)
        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
        //
        // ========== Encryption ends here ==========
        uint8_t authMode;      // Authentication mode
        uint8_t keyId;         // Key ID in shared table
        uint16_t reserved1a;   // (reserved for alignment)
        uint32_t authUnixTime; // Authentication time stamp
        uint8_t initVector[AUTH_IV_LENGTH]     // IV
        uint8_t authDigest[AUTH_DIGEST_LENGTH] // HMAC
```

};

Figure 10: Status PDU

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



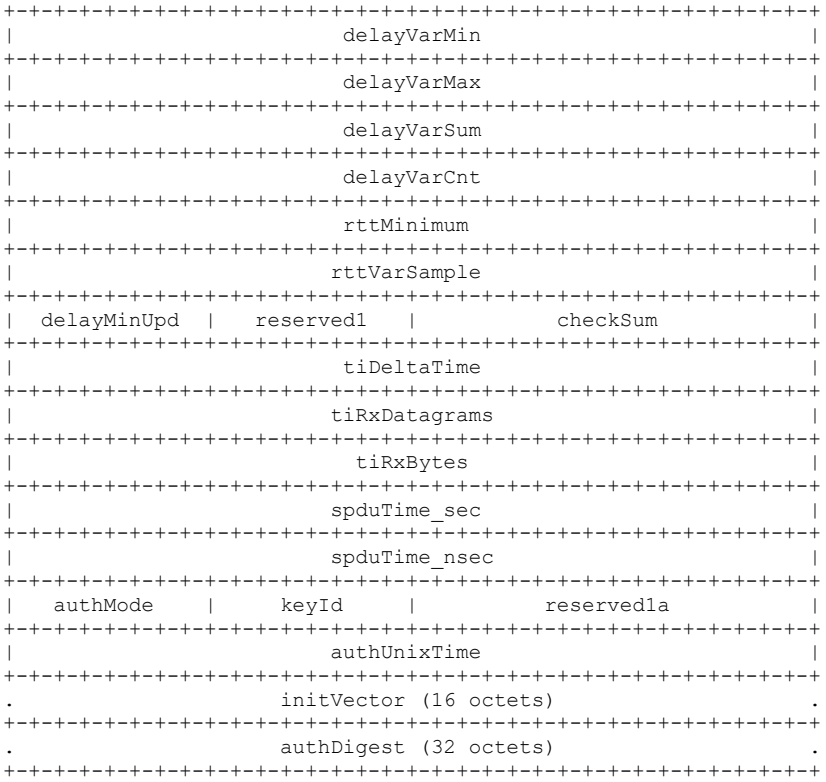


Figure 11: Status PDU Layout

Note that the Sending Rate structure is defined in Section 6.

The primary role of the Status Feedback message is to communicate to the Load PDU sender the traffic conditions at the Load PDU receiver. While the Sub-Interval Statistics structure (sisSav) covers the most recently saved (completed) sub-interval, similar fields directly in the Status PDU itself cover the most recent trial interval (the time period between Status Feedback messages, completed by this Status PDU). Both sets of statistics SHALL always be populated by the Load PDU receiver, regardless of role (client or server).

Details on the Status PDU measurement fields are provided in [RFC9097]. Additional information regarding fields not defined previously are as follows:

rxDatagrams/rxBytes/deltaTime: Sub-interval received datagram and byte counts as well as the exact duration of the sub-interval in microseconds. Used to calculate the received traffic rate for the sub-interval. The rxBytes field is a 64-bit value to prevent overflow at high speeds.

seqErrLoss/seqErrOoo/seqErrDup: Loss, out-of-order, and duplicate

totals. Available for both the sub-interval and trial interval. seqErrOoo and seqErrDup are realized by comparing sequence numbers. A lookback list of the last n sequence numbers received is used as the basis. Each Load PDU sequence number is checked against this lookback. The number n may depend on the implementation and on typical characteristics of environments, where UDPST is deployed (like mobile networks or Wi-Fi). Currently, a default sequence number interval of n=32 has been chosen. Specifically for seqErrOoo, each successively received higher seqno sets the next-expected-seqno to seqno+1 and anything below that is considered out-of-order (i.e., delayed). For example, given the sequence 93, 94, 95, 100, 96, 97,

101, 98, 99, 102, 103, ... reception of 96, 97, 98, and 99 would not increment the next-expected-seqno and would all be considered out-of-order.

delayVarMin/delayVarMax/delayVarSum/delayVarCnt: The one-way delay variation measurements of all received Load PDUs (where avg = sum/cnt). For each Load PDU received, the send time (lpduTime_sec/lpduTime_nsec) is subtracted from the local receive time, which is then normalized by subtracting the current clockDeltaMin. Available for both the sub-interval and trial interval.

rttVarMinimum/rttVarMaximum (in sisSav): The minimum and maximum RTT delay variation (rttVarSample) in the sub-interval designated by the subIntSeqNo.

accumTime: The accumulated time of the test in milliseconds, based on the duration of each sub-interval. Equivalent to the sum of each deltaTime (although in ms) sent in each Status PDU during the test.

spduSeqNo: Status PDU sequence number (starting at 1). Used by the Load PDU sender to detect Status PDU loss (in the unloaded direction). The loss count is communicated back to the Load PDU receiver via spduSeqErr in subsequent Load PDUs.

subIntSeqNo: Sub-interval sequence number (starting at 1) that corresponds to the statistics provided in sisSav, for the last saved (completed) sub-interval.

sisSav: Sub-interval statistics saved (completed) for the most recent sub-interval (as designated by the subIntSeqNo).

clockDeltaMin: The minimum clock delta (difference) since the beginning of the test. Obtained by subtracting the send time of each Load PDU (lpduTime_sec/lpduTime_nsec) from the local time that it was received. This value is initialized with the first Load PDU received and is updated with each subsequent one to maintain a current (and continuously updated) minimum. If the ~~end-point~~endpoint clocks are sufficiently synchronized, this will be the minimum one-way delay in milliseconds. Otherwise, this value may be negative, but still valid for one-way delay variation measurements for the default test duration (default is 10 [s]). If the test duration is extended to a range of minutes, where significant clock drift can occur, synchronized (or at least well disciplined) clocks may be required.

rttMinimum (in Status PDU): The minimum "adjusted" RTT measured since the beginning of the test. See rttRespDelay regarding "adjusted"

measurements. RTT is obtained by subtracting the copied spduTime_sec/spduTime_nsec in the received Load PDU from the local time at which it was received. This minimum SHALL be kept current (and continuously updated) via each Load PDU received with an updated spduTime_sec/spduTime_nsec. This value MUST be positive. Before an initial value can be established, and because zero is itself valid, it SHALL be set to STATUS_NORTT when communicated in the Status PDU.

rttVarSample: The most recent "adjusted" RTT delay variation measurement. See rttRespDelay regarding "adjusted" measurements. RTT delay variation is obtained by subtracting the current (and continuously updated) "adjusted" RTT minimum, communicated as rttMinimum (in Status PDU), from each "adjusted" RTT measurement (which is itself obtained by subtracting the copied spduTime_sec/ spduTime_nsec in the received Load PDU from the local time at which it was received). Note that while one-way delay variation is measured for every Load PDU received, RTT delay variation is only sampled via the Status PDU sent and the very next Load PDU received with the corresponding updated spduTime_sec/spduTime_nsec. When a new value is unavailable (possibly due to packet loss), and because zero is itself valid, it SHALL be set to STATUS_NORTT when communicated in the Status PDU.

delayMinUpd: Boolean (0 or 1) indicating that the clockDeltaMin and/ or rttMinimum (in Status PDU), as measured by the Load PDU receiver, has been updated.

checkSum: An OPTIONAL checksum of the entire PDU. The calculation is done with the checkSum field, and all fields starting with authMode, set to zero.

tiDeltaTime/tiRxDatagrams/tiRxBytes: The trial interval time in microseconds, along with the received datagram and byte counts. Used to calculate the received traffic rate for the trial interval.

spduTime_sec/spduTime_nsec: The local transmit time of the Status PDU. Expected to be copied into spduTime_sec/spduTime_nsec in subsequent Load PDUs after being received by the Load PDU sender. Used for RTT measurements.

The authentication, encryption, and checksum fields and their operation are as defined previously in Section 4.

8. Stopping ~~the~~ a Test

When the test duration timer (testIntTime) on the server expires, it SHALL set the local connection test action to TEST_ACT_STOP1 (phase 1 of graceful termination). This is simply a non-reversible state awaiting the next message(s) to be sent from the server. During this time, any received Load or Status PDUs are processed normally.

Upon transmission of the next Load or Status PDUs, the server SHALL set the local connection test action to TEST_ACT_STOP2 (phase 2 of graceful termination) and mark any outgoing PDUs with a testAction value of TEST_ACT_STOP2. While in this state, the server MAY reduce any Load PDU bursts to a size of one.

When the client receives a Load or Status PDU with the TEST_ACT_STOP2

indication, it SHALL finalize testing, display the test results, and also mark its local connection with a test action of TEST_ACT_STOP2 (so that any PDUs subsequently received can be ignored).

With the test action of the client's connection set to TEST_ACT_STOP2, the very next expiry of a send timer, for either a Load or Status PDU, SHALL result in it and any subsequent PDUs to be sent with a testAction value of TEST_ACT_STOP2 (as confirmation to the server). While in this state, the client MAY reduce any Load PDU bursts to a size of one. The client SHALL then schedule an immediate end time for the connection.

When the server receives the TEST_ACT_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 the associated resources. The TEST_ACT_STOP2 exchange constitutes a graceful termination of the test.

In a non-graceful test stop due to path failure, the watchdog timeouts at each ~~end-point~~endpoint will expire (sometimes at one ~~end-point~~endpoint first), notifications in logs, STDOUT, and/or formatted output SHALL be made, and the ~~end-point~~endpoint SHALL schedule an immediate end time for the connection.

If an attacker clears the TEST_ACT_STOP2 indication, then the configured test duration timer (testIntTime) at the server and client SHALL take precedence and the ~~end-point~~endpoint SHALL schedule an immediate end time for the connection.

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, 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. Notes on Interface Measurements

Additional measurements may be useful in specific circumstances. For example, interface byte counters measured by a client at a residential gateway are possible when the client application has access to an interface that sees all traffic to/from a service subscriber's location. Adding a byte counter at the client for download or upload directions could be used to measure total traffic and possibly detect when non-test traffic is present (and using capacity). The client may not have the CPU cycles available to count both the interface traffic and IP-layer Capacity simultaneously, so this form of diagnostic measurement may not be possible.

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 document.

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 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 REQUIRED. To accommodate different host limitations and testing circumstances, different modes of operation are recommended, as described in Section 4 above.

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 sending rates that will be supported. The default and optional load rate adjustment algorithm results in "ramp up" from the lowest rate in the table. Optionally, the server could utilize the maxBandwidth field (and CHSR_USDIR_BIT bit) in the Setup Request from the client to limit the maximum that it will attempt to achieve.

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 test interval duration values SHOULD be limited).

One specific attack that has been recognized is an on-path attack on the testAction field where the attacker would set or clear the STOP indication. Setting the indication in successive packets terminates

the test prematurely, with no threat to the Internet but annoyance for the testers. If an attacker clears the STOP indication, the mitigation relies on knowledge of the test duration at the client and server, where these hosts cease all traffic when the specified test duration is complete.

11. IANA Considerations

This document requests IANA to assign a "well-known" UDP port for the Test Setup exchange in the Control phase of protocol operation, and to create a new registry group for the UDP Speed Test Protocol (UDPSTP).

11.1. New System Port Number Assignment

IANA ~~will~~ is requested to allocate the following service name to the "Service Name and Transport Protocol Port Number Registry" registry:

Service: udpst-control

Transport Protocol: UDP

Assignee: IESG <iesg@ietf.org>

Contact: IETF Chair <chair@ietf.org>

Description: UDP-based IP-Layer Capacity and performance measurement protocol

Reference: This RFC, RFCYYYY. The protocol uses IP-Layer Unicast.

Port Number: <left blank, as instructed>

11.2. New UDPSTP Registry Group

IANA is request to create a new registry group, entitled

~~This section describes the design of the~~ "UDP Speed Test Protocol (UDPSTP)" ~~registry group~~.

~~The new registry group SHALL be named, "UDP Speed Test Protocol (UDPSTP)".~~

The following applies to each registry in the sub-sections below:

Registration Procedure: see below

Reference: <This RFC>

Experts: Performance Metrics Experts

Note: TBD

11.2.1. PDU Identifier Registry

Commenté [MB83]: I guess that you checked rfc6335 for the guidance for when it makes sense to request one.

Commenté [MB84]: Do you mean Performance Metrics Directorate (perfmetrdir) or something else?

IANA ~~will~~ is requested to create the "PDU Identifier" ~~Registry~~registry under the "UDP Speed Test Protocol (UDPSTP)"" registry group.

The first two octets of the PDUs used in the UDPST Protocol identify the role and format of the PDU that follows. The code points in this registry are allocated according to the registration procedures [RFC8126] described in Table 1.

Range (Hex)	Registration Procedures
0xFFFF and 0x0000	Reserved
0x8000-0xFFFE	IETF Review~~IETF~~
0x0001-0x7F00	First Ccome, ~~first~~ First Sserved
0x7F01-0x7FE0	Experimental
0x7FE1-0x7FFF	Private Use

Table 1: Registration procedures for the PDU Identifier Registry

~~By this document, IANA will assign the values listed in Table 2 in the "PDU Identifier Registry":~~The registry is initialized with the values in Table 2.

Identifier Name	Value	Reference	Change Controller	Description
controlId	0xACE1	<this RFC>	IETF	Test Setup PDU
controlId	0xACE2	<this RFC>	IETF	Test Activation PDU
controlId	0xDEAD	<this RFC>	IETF	Null PDU
loadId	0xBEEF	<this RFC>	IETF	Load PDU
statusId	0xFEED	<this RFC>	IETF	Status Feedback PDU

Table 2: Initial PDU Identifier ~~Registry values~~Values ~~to be assigned~~

11.2.2. Protocol Number Registry

IANA ~~will~~ is requested to create the "Protocol Number" ~~Registry"~~ under the "UDP Speed Test Protocol (UDPSTP)" registry group.

The second two octets of the PDUs used in the UDPST Protocol identify the version of the protocol in use. The code points in this registry are allocated according to the registration procedures [RFC8126] described in Table 3.

Range (Decimal)	Registration Procedures
0-7	Reserved
8-40960	IETF Review~~IETF~~

Commenté [MB85]: Is Expert Review needed here?

Commenté [MB86]: The guidance should describe whether the identifier is unique or not

Commenté [MB87]: Not sure this is needed here as the range is sufficient

Commenté [MB88]: I would move the reference column to be the last one

40961-53248	First Ceome, ~~first~~ First ~~served~~Served
53249-65534	Experimental
65535	Reserved

Table 3: Registration procedures for the Protocol Number Registry

~~By this document,~~IANA is request to register the following initial values (Table 4) ~~IANA will assign the decimal values listed in Table 4 in the "Protocol Number Registry"~~:

Field Name	Value	Reference	Change Controller	Description
protocolVer	8	<this RFC>	IETF	Protocol version 8
protocolVer	9	<this RFC>	IETF	Protocol version 9
protocolVer	10	<this RFC>	IETF	Protocol version 10
protocolVer	20	<this RFC>	IETF	Protocol version 20

Table 4: Initial Protocol Number ~~Registry V~~values ~~to be assigned~~

11.2.3. Test Setup PDU Modifier Bitmap Registry

IANA ~~will~~ is requested to create the "Test Setup PDU Modifier Bitmap Registry" under the "UDP Speed Test Protocol (UDPSTP)" registry group.

The Test Setup PDU layout contains a modifierBitmap field. The bitmaps in this registry are allocated according to the registration procedures [RFC8126] described in Table 5.

Range(Bitmap)	Registration Procedures
00000000-01111111	IETF Review
10000000	Reserved

Table 5: Registration procedures for the Test Setup PDU Modifier Bitmap Registry

By this document, IANA will assign the bitmap values defined by Table 6 in the "Test Setup PDU Modifier Bitmap Registry".

Field Name	Value	Reference	Change Controller	Description
modifierBitmap	0x00	<this RFC>	IETF	No modifications
modifierBitmap	0x01	<this RFC>	IETF	Allow Jumbo datagram sizes above sending rates of 1Gbps

Commenté [BMI89]: I would position this one as the last column

Commenté [BMI90]: This one is not needed

Commenté [BMI91]: Are there references for these ones?

Commenté [BMI92]: List this one as the last column

Commenté [MB93]: I guess this should be the name of the bit, not the field

Commenté [MB94]: That is?

modifierBitmap	0x02	<this RFC>	IETF	Use ~~Traditional~~ Conventional MTU (1500 bytes with IP-header)

Table 6: Initial ~~-~~Test Setup PDU Modifier Bitmap ~~Registry values to be assigned~~Values

11.2.4. Test Setup PDU Authentication Mode Registry

IANA ~~will~~ is requested to create the "Test Setup PDU Authentication Mode" ~~Registry~~registry under the "UDP Speed Test Protocol (UDPSTP)" registry group".

The Test Setup PDU layout contains an authMode field. The code points in this registry are allocated according to the registration procedures [RFC8126] described in Table 7.

Range(Decimal)	Registration Procedures
0-59	IETF Review
60-63	Experimental
64-255	Reserved

Table 7: Registration procedures for the Test Setup PDU Authentication Mode Registry

By this document, IANA will assign the decimal values defined by Table 8 in the "Test Setup PDU Authentication Mode Registry".

Field Name	Value	Reference	Change Controller	Description
authMode	0	<this RFC>	IETF	~~OPTIONAL~~ Optional Unauthenticated mode
authMode	1	<this RFC>	IETF	~~REQUIRED~~ Required authentication for the Control phase
authMode	2	<this RFC>	IETF	~~OPTIONAL~~ Optional authentication for the Data phase, in addition to the Control phase
authMode	3	<this RFC>	IETF	~~OPTIONAL~~ Optional partial encrypted mode

Table 8: Initial Test Setup PDU Authentication Mode ~~Registry values to be Assigned~~Values

Commenté [MB95]: Idem as above

Commenté [MB96]: Idem as above

11.2.5. Test Setup PDU Command Response Field Registry

IANA will create the "Test Setup PDU Command Response Field" Registry". the "UDP Speed Test Protocol (UDPSTP)" registry group.

The Test Setup PDU layout contains a cmdResponse field. The code points in this registry are allocated according to the registration procedures [RFC8126] described in Table 9.

Range(Decimal)	Registration Procedures
0-127	IETF Review
128-239	First Ceome, Ffirst servedServed
240-249	Experimental
250-254	Private Use
255	Reserved

Table 9: Registration procedures for the Test Setup PDU Command Response Field Registry

By this document, IANA will assign the decimal values defined by Table 10 in the "Test Setup PDU Command Response Field Registry".

Commenté [MB97]: Idem as above

Field Name	Value	Reference	Change Controller	Description

Commenté [MB98]: Why the same name is used?

Guidance is needed to help IANA

Commenté [MB99]: Idem as above

cmdResponse	0	<this RFC>	IETF	None (used by client in Request)
cmdResponse	1	<this RFC>	IETF	Acknowledgment
cmdResponse	2	<this RFC>	IETF	Bad Protocol Version
cmdResponse	3	<this RFC>	IETF	Invalid Jumbo datagram option
cmdResponse	4	<this RFC>	IETF	Unexpected Authentication in Setup Request
cmdResponse	5	<this RFC>	IETF	Authentication missing in Setup Request
cmdResponse	6	<this RFC>	IETF	Invalid authentication method
cmdResponse	7	<this RFC>	IETF	Authentication failure
cmdResponse	8	<this RFC>	IETF	Authentication time is invalid in Setup Request
cmdResponse	9	<this RFC>	IETF	No Maximum test Bit rate specified
cmdResponse	10	<this RFC>	IETF	Server Maximum Bit rate exceeded
cmdResponse	11	<this RFC>	IETF	MTU option does not match server
cmdResponse	12	<this RFC>	IETF	Multi-connection parameters rejected by server
cmdResponse	13	<this RFC>	IETF	Connection allocation failure on server

Table 10: Test Setup PDU Command Response Field Registry values to be assigned

11.2.6. Test Activation PDU Modifier Bitmap Registry

IANA will create the "Test Activation PDU Modifier Bitmap Registry". The Test Activation PDU layout (also) contains a modifierBitmap field. The bitmaps in this registry are allocated according to the registration procedures [RFC8126] described in Table 11.

Range(Bitmap)	Registration Procedures
00000000-01111111	IETF
10000000	Reserved

Table 11: Registration procedures for the Test Activation PDU

Commenté [MB100]: Idem as above

Modifier Bitmap Registry

By this document, IANA will assign the bitmap values defined by Table 12 in the "Test Activation PDU Modifier Bitmap Registry".

Field Name	Value	Reference	Change Controller	Description
modifierBitmap	0x00	<this RFC>	IETF	No modifications
modifierBitmap	0x01	<this RFC>	IETF	Set when srIndexConf is start rate for search
modifierBitmap	0x02	<this RFC>	IETF	Set for randomized UDP payload

Table 12: Test Activation PDU Modifier Bitmap Registry values to be assigned

11.2.7. Test Activation PDU Command Request Registry

IANA will create the "Test Activation PDU Command Request Registry". The Test Setup PDU layout contains a cmdRequest field. The code points in this registry are allocated according to the registration procedures [RFC8126] described in Table 13.

Commenté [MB101]: Idem as above

Range(Decimal)	Registration Procedures
0-127	IETF
128-239	First come, first served
240-249	Experimental
250-254	Private
255	Reserved

Table 13: Registration procedures for the Test Activation PDU Command Request Registry

By this document, IANA will assign the decimal values defined by Table 14 in the "Test Activation PDU Command Request Registry".

Field Name	Value	Reference	Change Controller	Description
cmdRequest	0	<this RFC>	IETF	No Request
cmdRequest	1	<this RFC>	IETF	Request test in Upstream direction (client to server)
cmdRequest	2	<this RFC>	IETF	Request test in Downstream direction (server to client)

Table 14: Test Activation PDU Command Request Registry values to be assigned

11.2.8. Test Activation PDU Rate Adjustment Algo. Registry

The Test Activation PDU layout contains a rateAdjAlgo field. The table below defines the assigned Capitalized alphabetic UTF-8 values in the registry.

IANA will create the "Test Activation PDU Rate Adjustment Algo. Registry". The Test Activation PDU layout contains a rateAdjAlgo field. The code points in this registry are allocated according to the registration procedures [RFC8126] described in Table 15.

Commenté [MB102]: Idem as above

Range(Capital alphabet. UTF-8)	Registration Procedures
A-Q	IETF
R-V	Experimental
W-Y	Private
Z	Reserved

Table 15: Registration procedures for the Test Activation PDU Rate Adjustment Algo. Registry

By this document, IANA will assign the Capitalized alphabetic UTF-8 values, as well as the corresponding incremental numeric, defined by Table 16 in the "Test Activation PDU Rate Adjustment Algo. Registry".

Field Name	Value (Numeric)	Reference	Change Controller	Description
rateAdjAlgo	A(n/a)	<this RFC>	IETF	Not used
rateAdjAlgo	B(0)	<this RFC>	IETF	Rate algorithm Type B
rateAdjAlgo	C(1)	<this RFC>	IETF	Rate algorithm Type C

Table 16: Test Activation PDU Rate Adjustment Algo. Registry values to be assigned

11.2.9. Test Activation PDU Command Response Field Registry

IANA will create the "Test Activation PDU Command Response Field Registry". The Test Activation PDU layout (also) contains a cmdResponse field. The code points in this registry are allocated according to the registration procedures [RFC8126] described in Table 17.

Range(Decimal)	Registration Procedures
0-127	IETF
128-239	First come, first served
240-249	Experimental

250-254	Private
255	Reserved

Table 17: Registration procedures for the Test Activation PDU Command Response Field Registry

By this document, IANA will assign the decimal values defined by Table 18 in the "Test Activation PDU Command Response Field Registry".

Field Name	Value	Reference	Change Controller	Description
cmdResponse	0	<this RFC>	IETF	None (used by client in Request)
cmdResponse	1	<this RFC>	IETF	Server Acknowledgment
cmdResponse	2	<this RFC>	IETF	Server indicates an error

Table 18: Test Activation PDU Command Response Field Registry values to be assigned

12. Acknowledgments

This specification has been edited by Al Morton. Al Morton died before he was able to finalise this work. As Al can't complete author tasks during the IETF standardisation process anymore, it was decided not to keep him as an author in the proper section. Respecting, that almost all of the content here has been edited or created by Al, it seems fair not just to give him credit for his work, but also list him as editor, if this document reaches RFC status.

Thanks to 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. Amanda Baber provided early reviews of the IANA Considerations section.

Brian Weis provided an early SEC-DIR review; version 02 captures clarifications and version 03-07 took up on the protocol changes which Brian suggested.

13. References

13.1. Normative References

[FIPS-197] National Institute of Standards and Technology, NIST., "Federal Information Processing Standards Publication 197 (FIPS-197), ADVANCED ENCRYPTION STANDARD (AES)", 26 November 2001, <https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf>.

[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>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[RFC7210] Housley, R., Polk, T., Hartman, S., and D. Zhang, "Database of Long-Lived Symmetric Cryptographic Keys", RFC 7210, DOI 10.17487/RFC7210, April 2014, <https://www.rfc-editor.org/info/rfc7210>.

[RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <https://www.rfc-editor.org/info/rfc8085>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

[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

[CBC] Dworkin, M., "NIST Special Publication 800-38A: Recommendation for Block Cipher Modes of Operation: Methods and Techniques, U.S. National Institute of Standards and Technology", December 2001, <https://csrc.nist.gov/pubs/sp/800/38/a/final>.

[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>.

[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>.

[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>.

[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>.

a mis en forme : Anglais (États-Unis)

Authors' Addresses

Len Ciavattone
AT&T Labs
Middletown, NJ
United States of America
Email: lenciavattone@gmail.com

Ruediger Geib
Deutsche Telekom
Deutsche Telekom Allee 9
64295 Darmstadt
Germany
Phone: +49 6151 5812747
Email: Ruediger.Geib@telekom.de