Stream: Internet Engineering Task Force (IETF)
RFC: 9946
Category: Standards Track
Published: April 2026
ISSN: 2070-1721
Authors: A. Morton, *AT&T Labs*; L. Ciavattone, *AT&T Labs*; R. Geib, Ed., *Deutsche Telekom*

RFC 9946
The UDP Speed Test Protocol (UDPSTP) for One-Way IP Capacity Metric Measurement

Abstract

This document addresses the problem of protocol support for measuring One-Way IP Capacity metrics specified by RFC 9097. The Method of Measurement discussed in that RFC requires 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 (UDPSTP) for conducting measurements as described in RFC 9097 and other related measurements.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9946.

Copyright Notice

Table of Contents

1. Introduction

The performance community has seen the development of informative Bulk Transport Capacity (BTC) definitions in "A Framework for Defining Empirical Bulk Transfer Capacity Metrics" [RFC3148] and "Defining Network Capacity" [RFC5136] as well as experimental metric definitions and methods in "Model-Based Metrics for Bulk Transport Capacity" [RFC8337].

This document specifies the UDP Speed Test Protocol (UDPSTP) to enable the measurement of One-Way IP Capacity metrics as defined by [RFC9097]. The Method of Measurement discussed in that RFC deploys a feedback channel from the receiver to control the sender's transmission rate in near real-time. Section 8.1 of [RFC9097] specifies requirements for this method.

UDPSTP supports measurement features that weren't available via TCP-based speed tests and standard measurement protocols, such as the One-Way Active Measurement Protocol (OWAMP) [RFC4656], Two-Way Active Measurement Protocol (TWAMP) [RFC5357], and Simple Two-Way Active Measurement Protocol (STAMP) [RFC8762], prior to this work. The controlled BTC measurement or Speed Test, respectively, is based on UDP rather than TCP. The bulk measurement load is unidirectional. These specifications did support the creation of asymmetric traffic in combination with some two-way communication, as supported by TWAMP and STAMP, when work on UDPSTP started. Further, two-way communications of TWAMP and STAMP are limited to reflection or unidirectional load properties, but they lack support for closed loop feedback operation. The latter enables limiting congestion of a bottleneck, whose capacity is measured, to a short time range. Support of such a control loop is the main purpose of UDPSTP.

Apart from measurement functionality, a Key Derivation Function (KDF) has been added to provide cryptographic separation of key material for authentication of protocol messages in a standardized and cryptographically secure manner. This is a secondary improvement reached by UDPSTP and may simplify its reuse for other measurement purposes. Additionally, because the protocol uses synthetic payload data and contains no direct user information, a decision was made to forgo encryption support. This is also expected to increase the number of low-end devices that can support the test methodology.

2. Conventions and Terminology

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.

Downstream UDP Speed Test: A client-initiated Network Capacity measurement between a server acting as sender and a client acting as receiver.

Upstream UDP Speed Test: A client-initiated Network Capacity measurement between a client acting as sender and a server acting as receiver.

In this document, the term "message" and the term "Protocol Data Unit", or "PDU" [RFC5044], are used interchangeably.

3. Scope, Goals, and Applicability

The scope of this document is to define a protocol to measure the Maximum IP-Layer Capacity Metric according to the Method of Measurement standardized by Section 8 of [RFC9097]. As such, this document adheres to the applicability scope defined in Section 2 of [RFC9097].

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. Per [RFC9097], that algorithm must not be used as a general Congestion Control Algorithm (CCA). Instead, the load rate adjustment algorithm's goal is to help determine the Maximum IP-Layer Capacity in the context of an infrequent, diagnostic, short-term measurement.

The goal is to harmonize the specified IP-Layer Capacity Metric and Method across the industry, and this protocol supports the specifications of IETF (see [RFC9097]) and other Standards Development Organizations (SDOs) (see, e.g., [TR-471]).

The primary application of UDPSTP 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.

UDPSTP is a client-based protocol. It may be applied by consumers to measure their own access bandwidth. Consumers may prefer an independent third-party domain hosting the measurement server for this purpose. UDPSTP may be deployed in Large-scale Measurement of Broadband Performance (LMAP) environments (see [RFC7497]) and other independent third-party domain measurement server deployments. A network operator may support operation and maintenance by UDPSTP, a typical intra-domain deployment. All these deployments require or benefit from trusting the results, which are ensured by authenticated communication.

4. Protocol Overview

All messages defined by this document **SHALL** use UDP transport [RFC0768].

The remainder of this section gives an informative overview of the communication protocol between two test endpoints (without expressing requirements or elaborating on the authentication aspects).

One endpoint takes the role of server, listening for connection requests on a standard UDP Speed Test Protocol port number from the other 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 hostname or IP address(es) of the server(s) in order to begin the setup and configuration exchanges with the server(s). By default, the client uses the single, standard UDPSTP port number per connection (see Section 6). If the default port number is not used, the client may require configuration of the control port number used by each server. This would be the case if multiple server instances (processes) operate on one or more machines.

Additionally, multi-connection (multi-flow) testing is supported by the protocol. Each connection is 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 server instance(s) would process each connection independently, as if they were coming from separate clients. It is the responsibility of the client process to manage the inter-related connections such as handling the individual connection setup successes and failures, cleaning up connections during a test (should some fail), and aggregating the individual test results into an overall set of performance statistics. Fields in the Setup Request (i.e., mcIndex, mcCount, and mcIdent; see Section 6.1) are used to both differentiate and associate the multiple connections that comprise a single test.

The protocol uses UDP transport with two connection phases (Control and Data). As shown below, exchanges 1 and 2 constitute the Control phase, while exchanges 3 and 4 constitute the Data phase. In this document, the term "message" and the term "Protocol Data Unit", or "PDU" [RFC5044], are used interchangeably.

1. Test Setup Request and Response: If a server instance is identified with a host name that resolves to both IPv4/IPv6 addresses, it is recommended to use the first address returned in the name resolution response, regardless of whether it is IPv4 or IPv6. Thus, the decision on the preferred IP address family is left to the name resolver's default behavior. Support for separate IPv4 and IPv6 measurements or an IPv4 and IPv6 multi-connection setup are left for future improvement. The client then requests to begin a test by communicating its UDPSTP protocol version, intended security mode, and datagram size support. The server either confirms matching a configuration or rejects the connection request. If the request is accepted, the server provides a unique ephemeral port number for each test connection, allowing further communication. In a multi-connection setup, distinct UDP port numbers may be assigned with each Setup Response from a server instance. Distinct UDP port numbers will be assigned if all Setup Response messages originate from the same server in that case.
2. Test Activation Request and Response: After having received a confirmation of the configuration by a server, the client composes a request conveying parameters such as the testing direction, the duration of the test interval and test Sub-Intervals, and various thresholds (for a detailed discussion, see [RFC9097] and [TR-471]). 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. See [RFC9097] for a more detailed discussion of firewall

and NAT-related features. If the Test Activation Request is rejected or fails, the client assumes that the firewall will close the address/port number pinhole entry after the firewall's configured idle traffic timeout.

3. Test Stream Transmission and Measurement Feedback Messages: Testing proceeds with one endpoint sending the Load PDUs and the other endpoint receiving the Load PDUs and sending frequent status messages to communicate the status and reception conditions. 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. Note that the status messages also help keep the pinhole (or mapping, respectively) active at on-path stateful devices. UDPSTP is at least partially compliant to Section 3.1 of [RFC8085] if the bottleneck is congested, but pending congestion is avoided by limiting the duration of that congestion to the minimum required to determine the bottleneck capacity.
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 above. 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 or Data phase, respectively. Watchdog timers limit the time a server or client will wait before stopping all traffic and terminating a test.

Figure 1 provides an example exchange of control and measurement PDUs for both downstream and upstream UDP Speed Tests (always client initiated):

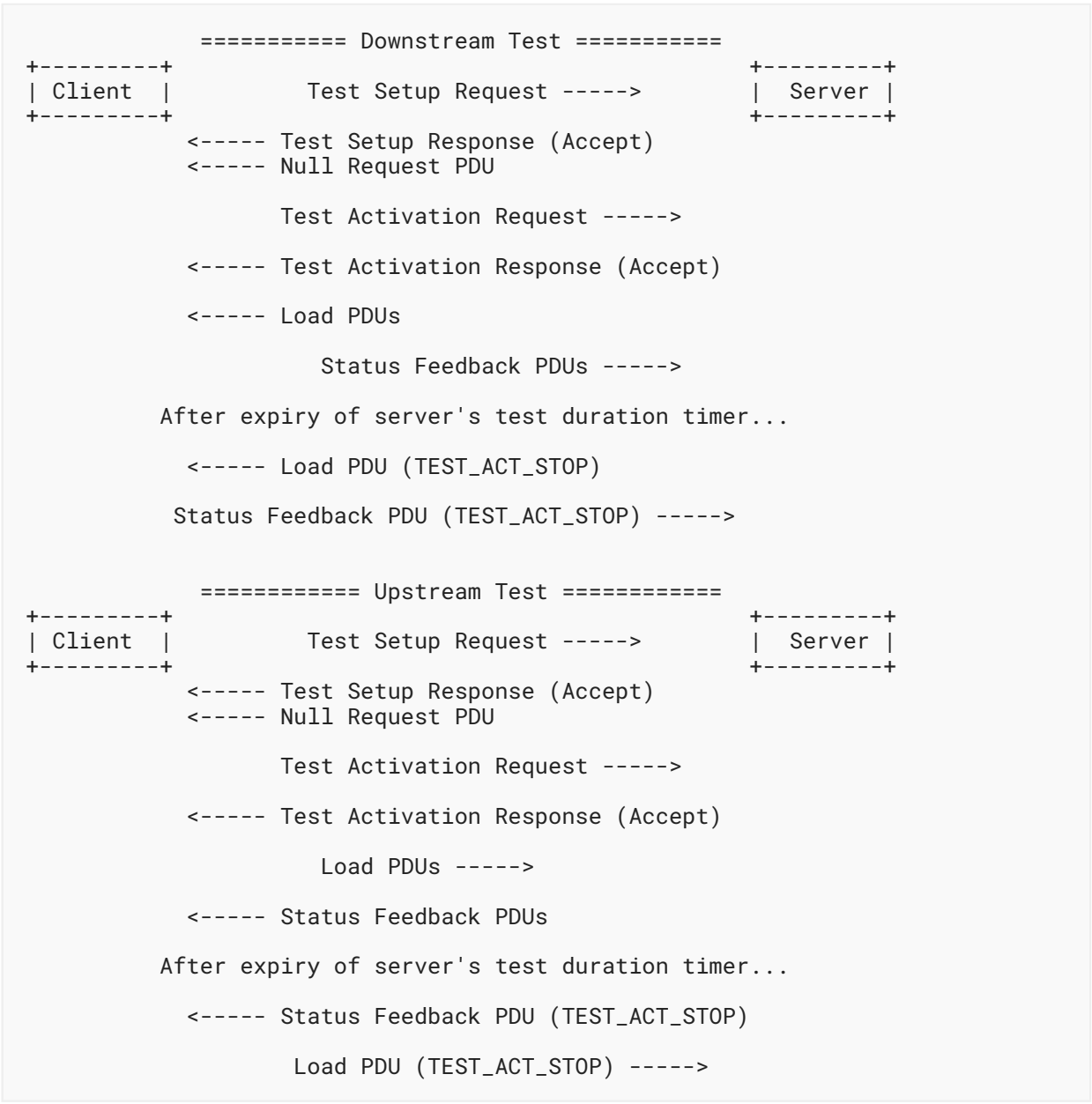


Figure 1: Successful UDPSTP Message Exchanges

4.1. Fixed-Rate Testing

A network operator who is certain of the IP-Layer Capacity to be validated can execute a fixed-rate test of the IP-Layer Capacity and avoid activating the measurement load rate adjustment algorithm (see Section 8.1 of [RFC9097]). Fixed-rate testing **SHOULD** only be activated for operation and maintenance purposes by operators within their local network domain.

If a subscriber requests a diagnostic test from the network operator, it strongly implies that there is no certainty on the bottleneck capacity and initiating a UDP Speed Test based on the load adjustment algorithm is **RECOMMENDED**. To protect against misuse, a client (and in general, a consumer) **MUST NOT** be able to initiate a fixed-rate test. A network operator may conduct a fixed-rate test for a stable measurement at or near the maximum determined by the load rate adjustment algorithm for debugging purposes. This may be valuable for post-installation or post-repair verification.

4.2. Handling of and Safeguards Required by Self-Induced Congestion

Active capacity measurement requires inducing intentional congestion. On paths where the capacity bottleneck is not shared with other flows, this self-congestion will be observed as loss and/or delay. However, when a path is shared by other flows, the measurement traffic can congest the bottleneck on the path and therefore degrade the performance of other flows. Unrestricted use of UDPSTP could lead to traffic starvation and significant issues.

Measurements that generate traffic on shared paths (including Wi-Fi and Internet paths) need to consider the impact on other traffic. Fixed-rate testing operates without congestion control and therefore must not be executed over other operators' network segments. Fixed-rate testing, therefore, is limited to paths within a domain entirely managed and operated section-wise and end-to-end by the network operator performing the measurement. When the risks of disruption to other flows has been considered, testing could be extended to include adjacent operational domains for which there is also a testing agreement.

Concurrent tests that congest a common bottleneck will impair the measurement and result in additional congestion. Concurrent measurements to measure the maximum capacity on a single path are counterproductive. The number of concurrent independent tests of a path **SHALL** be limited to one, regardless of the number of flows.

A load rate adjustment algorithm (see Section 5.1) is required to mitigate the impact of this congestion and to limit the duration of any congestion by terminating the test when sudden impairments or a loss of connectivity is detected.

5. Requirements, Security Operations, and Optional Checksum

Security and checksum operations aren't covered by [RFC9097], which only defines the Method of Measurement. This section adds the operational specification related to security and the optional checksum. Due to the additional complexities, and loss of the direct mapping of packets to datagrams between Layer 3 and Layer 4, it is recommended that Layer 3 fragmentation be avoided. A simplified approach is to choose the default datagram size that is small enough to prevent fragmentation. This version of the specification does not support Datagram Packetization Layer Path MTU Discovery (DPLPMTUD) [RFC8899]. A future version could specify how to support this. DPLPMTUD support will require a carefully adapted protocol design to ensure interoperability. Unless IP fragmentation is expected, and is one of the attributes being measured, the IPv4 Don't Fragment (DF) bit **SHOULD** be set for all tests.

Note: When this specification is used for network debugging, it may be useful for fragmentation to be under the control of the test administrator.

This section specifies generic requirements, which a measurement load rate adjustment algorithm conforming to this specification **MUST** fulfill.

5.1. Load Rate Adjustment Algorithm Requirements

This document specifies an active capacity measurement method using a load rate adjustment algorithm. The requirements listed in this section and the currently standardized load rate adjustment algorithms B [Y.1540Amd2] and C [TR-471] result from years of experiments and testing by the original authors. These tests were performed in labs, and also in the Internet, and covered a set of different fixed, broadband, mobile, and wireless access types and technologies in different countries and continents. Further, the load rate adjustment algorithm requirements listed below reflect feedback from performance measurement experts, as well as changes resulting from the standardization of [RFC9097] (reflected also in algorithm B [Y.1540Amd2], which updates a prior version of this algorithm).

Load rate adjustment algorithms for capacity measurement **MUST** comply with the requirements specified by this section. New standard load rate adjustment algorithms for capacity measurement **MUST** be reviewed by IETF designated experts prior to assignment of a code point in the "Test Activation PDU Rate Adjustment Algo" registry.

The load rate adjustment algorithm for capacity measurement requirements are as follows:

1. The measurement load rate adjustment algorithm described in this section **MUST NOT** be used as a general CCA.
2. This specification **MUST** only be used in the application of diagnostic and operations measurements.
3. Both Load PDU messages and Status Feedback PDU messages **MUST** contain sequence numbers.
4. The nominal duration of a measurement interval at the Destination, parameter testIntTime ("I" in [RFC9097]), **MUST** default to a value of no more than 10 seconds.
5. A high-speed mode to achieve high sending rates quickly **MUST** reduce the measurement load below a level for which the first feedback interval inferred "congestion" 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 indicate "congestion" within a test. The threshold of consecutive feedback intervals **SHALL** be configurable with a default of 3 intervals and a maximum duration to infer congestion of 500 ms (milliseconds).
6. Congestion **MUST** be indicated if the Status Feedback PDUs indicate that either sequence number anomalies were detected OR the delay range was above the upper delay variation threshold. The **RECOMMENDED** threshold values are 10 for sequence number gaps, 30 ms for lower delay variation thresholds, and 90 ms for upper delay variation thresholds.
7. The load rate adjustment algorithm **MUST** include a Load PDU timeout and a Status PDU timeout, which both stop the test when received PDU streams cease unexpectedly.

8. The Load PDU timeout **SHALL** be reset to the configured value each time a Load PDU is received. If the Load PDU timeout expires, the receiver **SHALL** be closed and no further Status PDU feedback sent. The default Load PDU timeout **MUST** be no more than 1 second.
9. The Status PDU timeout **SHALL** be reset to the configured value each time a feedback message is received. If the Status PDU timeout expires, the sender **SHALL** be closed and no further load packets sent. The default Status PDU timeout **MUST** be no more than 1 second.
10. A network operator who is certain of the IP-Layer Capacity to be validated **MAY** start with a fixed-rate test at the IP-Layer Capacity and avoid activating the measurement load rate adjustment algorithm (see Section 8.1 of [RFC9097]). However, the stimulus for a diagnostic test (such as a subscriber request) strongly implies that there is no certainty, and the load adjustment algorithm is **RECOMMENDED**.
11. This specification **MUST** only be used in circumstances consistent with Section 10 (Security Considerations) of [RFC9097].
12. Further measurement load rate adjustment algorithm requirements are specified by [RFC9097].

The following measurement load rate adjustment algorithms are subject to these requirements:

- Measurement load rate adjustment algorithm B [Y.1540Amd2].
- Measurement load rate adjustment algorithm C [TR-471].

5.2. Parameters and Definitions

Please refer to Section 4 of [RFC9097] for an overview of parameters related to the Maximum IP-Layer Capacity Metric and Method. A set of error codes to support debugging are provided in Section 12.3.5.

5.3. Security Mode Operations

There are two security modes of operation that perform authentication of the client/server messaging. The two modes are:

1. A **REQUIRED** mode with authentication during the Control phase (Test Setup and Test Activation exchanges). This mode may be preferred for large-scale servers or low-end client devices where processing power is a consideration (see Section 3).
2. An **OPTIONAL** mode with the additional authentication of the Status Feedback messages during the Data phase. This mode may be preferred for environments that desire an additional level of message integrity verification throughout the test (see Section 3).

The requirements discussed hereafter refer to the PDUs in Sections 6 and 7 below, primarily the authMode, keyId, authUnixTime, 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 by other sections within this document. Each successive mode increases security but comes with additional performance impacts and complexity. The protocol is used with unsubstantial payload, and it may operate on very low-end devices. Offering the flexibility of

various security operation modes allows for accommodation of available end-device resources. In general, an active measurement technique as the one defined by this document is better suited to protect the privacy of those involved in measurements [RFC7594].

A load rate adjustment method needs to satisfy the requirements listed in Section 5.1. This is necessary also to avoid potentially inducing congestion after there is an overload or loss (including loss on the control path).

5.3.1. Mode 1: Required Authenticated Mode

In this mode, the client and the server **SHALL** be configured to use one of a number of shared secret keys, designated via the numeric keyId field (see Section 5.4). This key **SHALL** be used as input to the KDF, as specified in Section 5.4.1, to obtain the actual keys used by the client and server for authentication.

During the Control phase, the sender **SHALL** read the current system (wall-clock) time and populate the authUnixTime field and next calculate the 32-octet HMAC-SHA-256 hash of the entire PDU according to Section 6 of [RFC6234] (with the authDigest and checkSum preset to all zeroes). The authDigest field is filled by the result, then the packet is sent to the receiver. The value in the authUnixTime field is a 32-bit timestamp, and a 10-second tolerance window (+/- 5 seconds) **SHALL** be used by the receiver to distinguish a subsequent replay of a PDU. See Table 2 of [TR-471] for a recommended timestamp resolution.

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 requires that it be extracted from the PDU and the field, along with the checkSum field, zeroed prior to the Hashed Message Authentication Code (HMAC) calculation used for comparison (see Section 7.2 of [RFC9145]).

If the validation fails, the receiver **SHOULD NOT** continue with the Control phase and **SHOULD** 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 6) and the Test Activation exchange (Section 7).

5.3.2. Mode 2: Optional Authenticated Mode for Data Phase

This mode incorporates Authenticated mode 1. When using the optional authentication during the Data phase, authentication **SHALL** also be applied to the Status Feedback PDU (see Section 8.2). The client sends the Status PDU in a downstream test, and the server sends it in an upstream test.

The Status PDU sender **SHALL** 1) read the current system (wall-clock) time and populate the authUnixTime field, 2) calculate the authDigest field of the entire Status PDU (with the authDigest and checkSum preset to all zeroes), and 3) send the packet to the receiver. The values of authUnixTime field and authDigest field are determined as defined by Section 5.3.1.

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, along with the checkSum 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 validations), the test session will prematurely terminate (and no further load traffic **SHALL** be transmitted). This is necessary also to avoid potentially inducing congestion after there is an overload or loss on the control path.

If this optional mode has not been selected, then the keyId, authUnixTime, and authDigest fields of the Status PDU (see Section 8.2) **SHALL** be set to all zeroes.

5.4. Key Management

Section 2 of [RFC7210] specifies a conceptual database for long-lived cryptographic keys. The key table **SHALL** be used with the **REQUIRED** authentication mode and the **OPTIONAL** authentication mode (using the same key). For authentication, this key **SHALL** only be used as input to the KDF, as specified in Section 5.4.1, to derive the actual keys used for authentication processing. Key rotation and related management specifics are beyond the scope of this document.

The key table **SHALL** have (at least) the following fields per Section 2 of [RFC7210]:

- AdminKeyName
- LocalKeyName
- KDF
- AlgID
- Key
- SendLifetimeStart
- SendLifeTimeEnd
- AcceptLifeTimeStart
- AcceptLifeTimeEnd

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

5.4.1. Key Derivation Function (KDF)

A KDF is a one-way function that provides cryptographic separation of key material. The protocol requires a KDF to securely derive cryptographic keys used for authentication of protocol messages. The inclusion of a KDF ensures that keys are generated in a standardized, cryptographically secure manner, reducing the risk of key compromise and enabling interoperability across implementations. The benefits of using a KDF include:

- Security: A KDF produces keys with high entropy, resistant to brute-force and related-key attacks, ensuring robust protection for protocol communications.
- Flexibility: The KDF allows derivation of multiple keys from a single shared secret, supporting distinct keys for client and server authentication.
- Standardization: By adhering to established cryptographic standards, the KDF ensures compatibility with existing security frameworks and facilitates implementation audits.
- Efficiency: The KDF enables efficient key generation without requiring additional key exchange mechanisms, minimizing protocol overhead.

The KDF algorithm **SHALL** be a Key Derivation Function in Counter Mode, as specified in Section 4.1 of [NIST800-108]. This algorithm uses a counter-based mechanism to generate key material from a shared secret, ensuring deterministic and secure key derivation. The Pseudorandom Function (PRF) used in the KDF **SHALL** be HMAC-SHA-256, as defined in Section 6 of [RFC6234]. IANA has assigned "HMAC-SHA-256" as a new KeyTable KDF (Section 12.2).

The KDF **SHALL** use the following parameters:

- Kin (key-derivation key): The shared key as identified by the keyId field in the PDU.
- Label: The fixed string "UDPSTP" (without quotes), encoded as a UTF-8 string, used to bind the derived keys to this specific protocol.
- Context: The UTF-8 string representation of the authUnixTime field received in the very first Setup Request PDU sent from the client to the server. This ensures that the derived keys are unique to the session and tied to the temporal context of the initial setup exchange. The authUnixTime field serves as a nonce and is protected from modification by the HMAC-SHA-256 hash present in the authDigest field.
- r: The length of the binary encoding of the counter **SHALL** be 32 (bits).

The total derived key material **SHALL** be 512 bits (64 octets) in length. The key material **SHALL** be structured as follows, from most significant bit (MSB) to least significant bit (LSB):

- Client Authentication Key: 256 bits (32 octets); used for authenticating messages sent by the client.
- Server Authentication Key: 256 bits (32 octets); used for authenticating messages sent by the server.

This structure ensures that the derived keys are sufficient for securing authentication operations within the protocol, while maintaining clear separation of function and directionality.

If authentication of the initial Setup Request PDU received by the server fails, due to an invalid authDigest field, any and all derived keying material and keys **SHALL** be considered invalid.

The key material derived from the initial Setup Request PDU, either at the client prior to transmission or at the server upon reception, **SHALL** be used for all subsequent PDUs sent between them for that test connection. As such, the KDF is only required to be executed once by the client and server for each test connection.

Appendix A, Figure 12 provides a code snippet demonstrating derivation of the specified keys from key material using the OpenSSL cryptographic library, specifically the high-level Key-Based EVP_KDF implementation (Key-Based Envelope Key Derivation Function); see [EVP_KDF-KB] for details.

5.5. Configuration of Network Functions with Stateful Filtering

Successful interaction with a local firewall assumes the firewall is configured to allow a host to open a bidirectional connection using unique source and destination addresses as well as port numbers (i.e., a 4-tuple) by sending a packet using that 4-tuple for a given transport protocol. The client's interaction with its firewall depends on this configuration.

The firewall at the server **MUST** be configured with an open pinhole for the server IP address and standard UDP port number of the server. All messages sent by the client to the server use this standard UDP port number.

The server uses one ephemeral UDP port number per test connection. 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 Test Setup Response. The Null Request may not reach the client: it may be discarded by the client's firewall.

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.

Network Address Translators (NATs) are expected to offer support of a wider set of operational configurations as compared to firewalls. Specifications covering NAT behavior, apart from the above, are out of the scope of this document, as are combined implementations of NAT and firewalls too.

5.6. Optional Checksum

The protocol **MUST** utilize the standard UDP checksum for all IPv4 and IPv6 datagrams it sends. The purpose of this checksum is to protect the intended recipient as well as other recipients to whom a corrupted packet may be delivered. This provides:

- Protection of the endpoint transport state from unnecessary extra state (e.g., Invalid state from rogue packets).
- Protection of the endpoint transport state from corruption of internal state.

- Pre-filtering by the endpoint of erroneous data, to protect the transport from unnecessary processing and from corruption that it cannot itself reject.
- Pre-filtering of incorrectly addressed destination packets, before responding to a source address.

All of the PDUs exchanged between the client and server support an optional header checksum that covers the various fields in the UDPSTP PDU (excluding the payload content of the Load PDU and, to be clear, also the IP and UDP headers). The calculation is the same as the 16-bit one's complement Internet checksum used in the IPv4 packet Header Checksum specification (see Section 3.1 of [RFC0791]). This checksum is intended for environments where UDP data integrity may be uncertain. This includes situations where the standard UDP checksum is not verified upon reception or a nonstandard network API is in use (things typically done to improve performance on low-end devices). However, all UDPSTP datagrams transmitted via IPv4 or IPv6 **SHALL** include a standard UDP checksum to protect other potential recipients to whom a corrupted packet may be delivered. In the case of a nonstandard network API, one option to reduce processing overhead may be to restrict testing to only utilize a payload content of all zeros so that the UDP checksum calculation need not include it for Load PDUs.

If a PDU sender is populating the checkSum field, it **SHALL** do so as the last step after the PDU is built in all other respects (with the checkSum field set to zero prior to the calculation). The PDU receiver **SHALL** subsequently verify the PDU checksum whenever checksum processing has been configured and the field is populated. If PDU checksum validation fails, the PDU **SHALL** be discarded.

Because of the redundancy when used in conjunction with authentication, it is **OPTIONAL** for a PDU sender to utilize the UDPSTP checkSum field. 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).

6. Test Setup Request and Response

The client source IP address and the server destination IP address **MUST NOT** be a broadcast or multicast address. Any Test Setup Request or Test Setup Response packet containing a multicast or broadcast source or destination IP address **MUST** be silently dropped and ignored.

The measurement method and the protocol specified by this document are expected to function with unicast and anycast IP addresses.

6.1. Client Generates Test Setup Request

The client **SHALL** begin the Control phase exchange by sending a Test Setup Request message to the server's (standard) control port. This standard UDPSTP port number is utilized for each connection of a multi-connection test.

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 (i.e., the UDP socket will be closed and an error message will be displayed to the user). Lost messages result in a Test Setup and Test Activation failure. The test initiation timer **MAY** reuse the test termination timeout value.

The watchdog timeout is configured as a 1-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.

Note: Any field labeled as 'reserved for alignment', in any PDU, **MUST** be set to 0 and **MUST** be ignored upon receipt.

The UDP PDU format layout **SHALL** be as follows (big-endian AB, starting with the most significant byte and ending with the least significant byte):

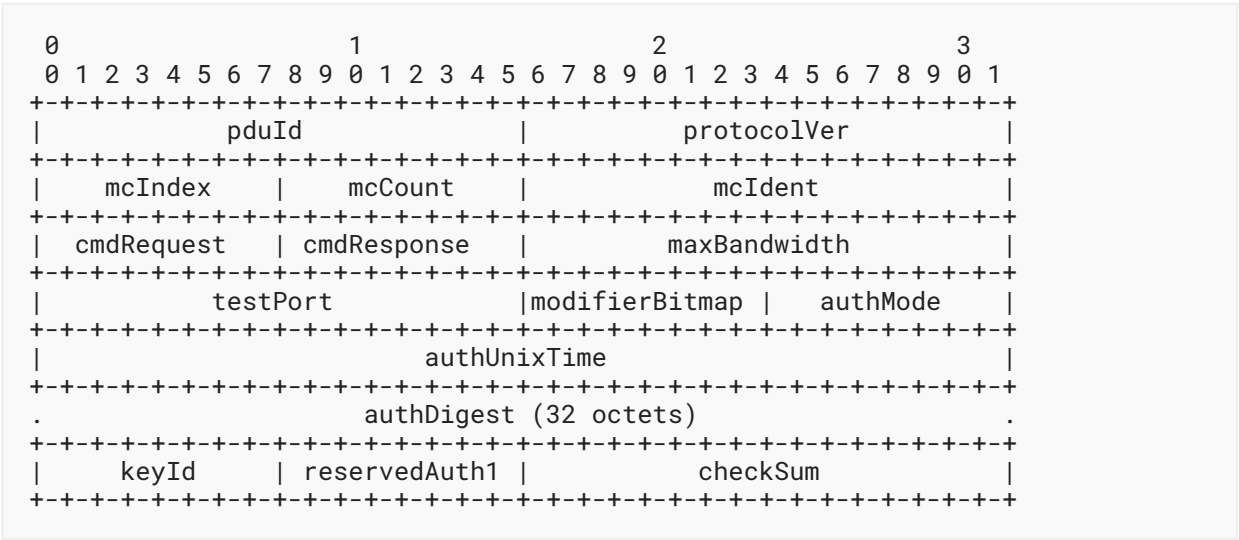


Figure 2: Test Setup PDU Layout

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

pduId: A two-octet field. IANA has assigned the hex value 0xACE1 (Section 12.3.1).

protocolVer: A two-octet field identifying the actual protocol version. IANA has assigned 20 as the value (Section 12.3.2).

mcIndex: A one-octet field indicating the index of a connection relative to all connections that make up a single test (starting at 0, incremented by 1 per connection). It is used to differentiate separate connections within a multi-connection test. An implementation may restrict the number of connections supported for a single test to a value less than or equal to 255.

mcCount: A one-octet field indicating the total count of connections that the client is attempting to set up.

mcIdent: A two-octet field containing a pseudorandom non-zero identifier (via a Random Number Generator, source port number, ...) that is common to all connections of a single test. It is used by clients/servers to associate separate connections with a single multi-connection test.

cmdRequest: A one-octet field set to CHSR_CREQ_SETUPREQ to indicate a Setup Request message. Note that CHSR_CREQ_NONE remains unused.

cmdResponse: A one-octet field. All Request PDUs always have a Command Response of XXXX_CRSP_NONE (i.e., CHSR_CRSP_NONE, CHNR_CRSP_NONE, or CHTA_CRSP_NONE).

maxBandwidth: A two-octet field. A non-zero value of this field specifies the maximum bit rate the client expects to send or receive during the requested test in Mbps. The server compares this value to its currently available configured limit for test admission control. This field **MAY** be used to rate-limit the maximum rate the server should attempt. The maxBandwidth field's most significant bit, the CHSR_USDIR_BIT, is set to 0 by default to indicate "downstream" and has to be set to 1 to indicate "upstream".

testPort: A two-octet field set to zero in the Test Setup Request and populated by the server in the Test Setup Response. It contains the UDP ephemeral port number on the server that the client has to use for the Test Activation Request and subsequent Load or Status PDUs.

modifierBitmap: A one-octet field. This document only assigns two bits in this bitmap; see Section 12.3.3:

CHSR_JUMBO_STATUS (0x01): This bit **SHALL** be set by default. By default, sending rates up to 1 Gbps **SHALL NOT** produce IP packet sizes greater than 1250 bytes (unless CHSR_TRADITIONAL_MTU is set), while rates above 1 Gbps **MAY** produce IP packet sizes up to 9000 bytes. When CHSR_JUMBO_STATUS is not set, all sending rates **SHALL NOT** produce IP packet sizes greater than 1250 bytes (unless CHSR_TRADITIONAL_MTU is set).

CHSR_TRADITIONAL_MTU (0x02): This bit **SHALL NOT** be set by default. If set, sending rates up to 1 Gbps **MAY** produce IP packets up to the traditional size of 1500 bytes. If CHSR_JUMBO_STATUS is simultaneously not set, all sending rates **SHALL NOT** produce IP packets greater than the traditional size of 1500 bytes.

Other bit positions are left unassigned per this document.

authMode: A one-octet field. The authMode field currently has two values assigned (see Section 12.3.4). One of the following has to be set (see Section 5.3 for requirements and details of operation):

AUTHMODE_1: Required authentication for the Control phase.

AUTHMODE_2: Required authentication for the Control and Data phases (Status Feedback PDU only).

A range of 60 through 63 is reserved for experimentation. IANA has created the "Test Setup PDU Authentication Mode" registry for the assigned values; see Section 12.3.4.

authUnixTime: A 32-bit timestamp of the current system (wall-clock) time since the Unix Epoch on January 1, 1970 at 00:00:00 UTC.

authDigest: This field contains the 32-octet HMAC-SHA-256 hash that covers the entire PDU. Normally, the calculation is done as the last step of building the PDU. However, if the optional checkSum field is being utilized, it becomes the penultimate step and is done just prior to the checksum calculation (with the checkSum field set to zero).

keyId: A one-octet field carrying localKeyName, the numeric key identifier for a key in the shared key table.

reservedAuth1: A one-octet field. This field **MUST** be set to 0 and **MUST** be ignored upon receipt. Consistent naming and placement of the reservedAuth1 field across all PDUs is done to minimize authentication-related changes in future UDPSTP versions.

checkSum: A two-octet field containing an optional checksum of the entire PDU (see Section 5.6 for guidance). The calculation is done as the very last step of building the PDU, with the checkSum field set to zero.

6.2. Server Test Setup Request Processing and Response Generation

This section describes the processes at the server that are used to evaluate the Test Setup Request and determine the next steps. When the server receives the Setup Request, it **SHALL** first perform the following:

Message Verification Procedure:

1. Verify that the size of the message is correct.
2. If the optional checkSum field is being utilized, validate the checksum as described in Section 5.6 and (if valid) zero the checkSum field prior to authentication verification.
3. Verify that the authMode value is valid and appropriate (per Section 5.3) for the message type.
4. If the authMode is valid and appropriate, authenticate the message by checking the authDigest as prescribed in Section 5.3.
5. If the message is authentic, check the authUnixTime field for acceptable immediacy.

Note: If any of the above checks fail, the message **SHALL** be considered invalid.

6.2.1. Test Setup Request Processing -- Rejection

The server **SHALL** then evaluate the other fields in the protocol header, such as the protocol version, the PDU 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 MTU (1500 bytes).

If the client has selected options for

- Jumbo datagram support (modifierBitmap),
- Traditional MTU (modifierBitmap), and
- 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:

- 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 the authDigest is invalid, then the Test Setup Request **SHOULD** fail silently. The exception is for operations support: server administrators are permitted to send a Setup Response to support operations and troubleshooting.

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

- the Setup Request PDU size is not equal to the 'struct controlHdrSR' size shown in Figure 3,
- the PDU ID is not 0xACE1 (Test Setup PDU), or
- a directed attack has been detected.

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

When the server replies to a Test Setup Request message, the Test Setup Response PDU is structured identically to the Test Setup 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 cmdResponse 2, Bad Protocol Version; see Section 12.3.5), 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 authUnixTime field is updated to the current system (wall-clock) time and, after the authDigest and checkSum fields are zeroed, the authDigest is recalculated and inserted. If the optional checkSum field is being utilized, it is then also calculated and inserted.

The Setup Request/Response message PDU **SHALL** be organized as follows (here and in all following code figures coded by programming language C [C-Prog]):

```
<CODE BEGINS>
//
// Control header for UDP payload of Setup Request/Response PDUs
//
struct controlHdrSR {
#define CHSR_ID 0xACE1
        uint16_t pduId; // PDU 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
#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 in Mbps
        uint16_t testPort;     // Test port on server
#define CHSR_JUMBO_STATUS    0x01
#define CHSR_TRADITIONAL_MTU 0x02
        uint8_t modifierBitmap; // Modifier bitmap
        // ========== Integrity Verification ==========
#define AUTHMODE_1 1 // Mode 1: Authenticated Control
#define AUTHMODE_2 2 // Mode 2: Authenticated Control+Status
        uint8_t authMode;      // Authentication mode
        uint32_t authUnixTime; // Authentication timestamp
#define AUTH_DIGEST_LENGTH 32  // SHA-256 digest length
        uint8_t authDigest[AUTH_DIGEST_LENGTH];
        uint8_t keyId;         // Key ID in shared table
        uint8_t reservedAuth1; // (reserved for alignment)
        uint16_t checkSum;     // Header checksum
};
#define SHA256_KEY_LEN 32 // Authentication key length

<CODE ENDS>
```

Figure 3: Test Setup PDU

6.2.2. 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 testPort field of 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. The watchdog timer is set to the same value as on the client side (see Section 6)

When the server replies to the Test Setup Request message, the Test Setup Response PDU is structured identically to the Test Setup 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 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 authUnixTime field is updated to the current system (wall-clock) time and, after the authDigest and checkSum fields are zeroed, the authDigest is recalculated and inserted. If the optional checkSum field is being utilized, it is then also calculated and inserted.

Finally, the new UDP connection associated with the new socket and port number 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 number, 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). If received, the client **SHALL** follow the Message Verification Procedure listed in Section 6.2, Paragraph 2. Note that there is no response to a Null Request.

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

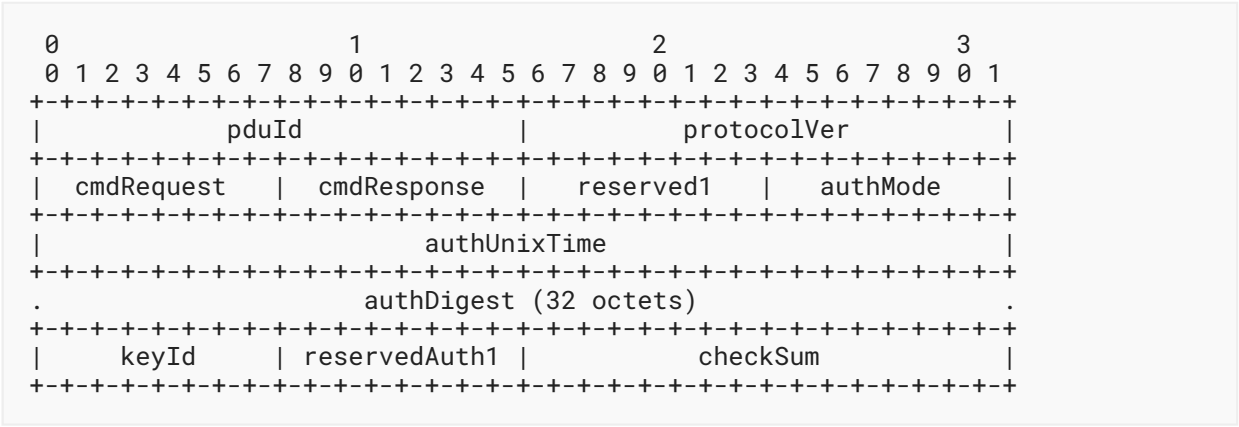


Figure 4: Null Request PDU Layout

The authentication and checkSum fields follow the same methodology as with the Setup Request and Response.

Additional details regarding the Null Request fields are as follows:

pduId: IANA has assigned the hex value 0xDEAD (Section 12.3.1).

cmdRequest: Set to CHNR_CREQ_NULLREQ to indicate a Null Request message.

cmdResponse: Set to CHNR_CRSP_NONE.

authMode: Same as in Section 6.1.

authUnixTime: Same as in Section 6.1.

authDigest: Same as in Section 6.1.

keyId: Same as in Section 6.1.

reservedAuth1: Same as in Section 6.1.

checkSum: Same as in Section 6.1.

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.

The Null Request message PDU **SHALL** be organized as follows:

```
<CODE BEGINS>
//
// Control header for UDP payload of Null Request PDU
//
struct controlHdrNR {
#define CHNR_ID 0xDEAD
        uint16_t pduId;       // PDU 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
        uint8_t reserved1;   // (reserved for alignment)
        // ========== Integrity Verification ==========
        uint8_t authMode;      // Authentication mode
        uint32_t authUnixTime; // Authentication timestamp
        uint8_t authDigest[AUTH_DIGEST_LENGTH];
        uint8_t keyId;         // Key ID in shared table
        uint8_t reservedAuth1; // (reserved for alignment)
        uint16_t checkSum;     // Header checksum
};

<CODE ENDS>
```

Figure 5: Null Request PDU

6.3. Setup Response Processing at the Client

When the client receives the Test Setup Response message, it **SHALL** first follow the Message Verification Procedure listed in Section 6.2, Paragraph 2.

The client **SHALL** then proceed to evaluate the other fields in the protocol, beginning with the protocol version, PDU ID (to validate the type of message), and cmdRequest for the role of the message, which **MUST** be Test Setup Response, CHSR_CREQ_SETUPRSP, as indicated by Figure 3.

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 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.

7. Test Activation Request and Response

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

7.1. 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 standard UDPSTP port).

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

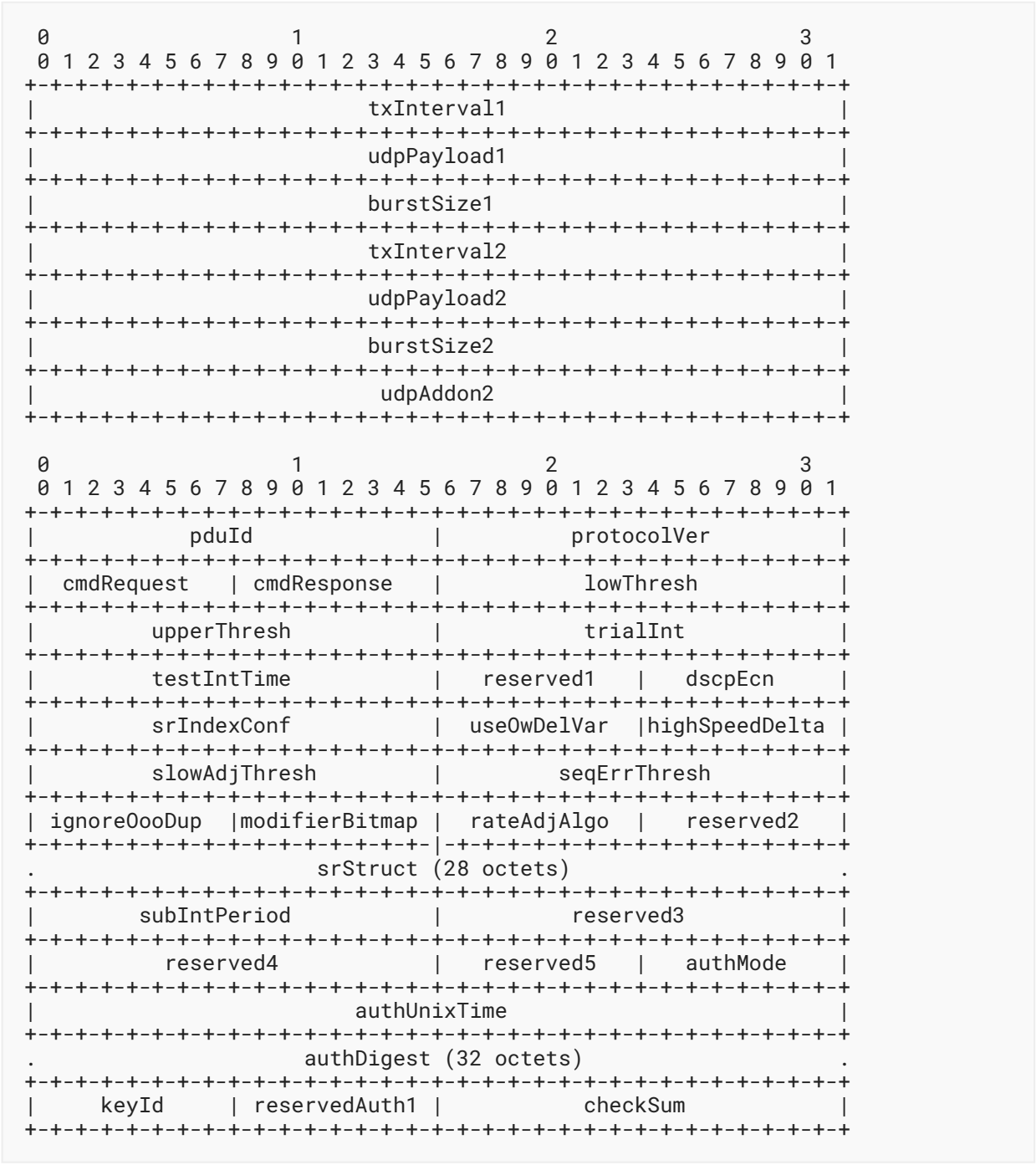


Figure 6: Test Activation PDU Layout

Fields are populated based on default values or command-line options. The authentication and checkSum fields follow the same methodology as with the Setup Request and Response.

pduId: IANA has assigned the hex value 0xACE2 (Section 12.3.1).

cmdRequest: Set to CHTA_CREQ_TESTACTUS to indicate an upstream test activation or alternatively to CHTA_CREQ_TESTACTDS to indicate a downstream test activation. Note that CHTA_CREQ_NONE remains unused. See Section 12.3.6.

cmdResponse: Three CHTA_CRSP_<Indication> values are defined; see Figure 7.

lowThresh: A two-octet field. The lower threshold on the range of Round-Trip Time (RTT) variation (the range is composed of values above the minimum RTT); see also Table 3 [TR-471].

upperThresh: A two-octet field. The upper threshold on the range of RTT variation (the range is composed of values above the minimum RTT); see also Table 3 of [TR-471].

trialInt: A two-octet field indicating the Status Feedback / trial interval in ms. The test interval Delta_t is subdivided into a number of Sub-Intervals dt, and each Sub-Interval is further divided into a number of trial intervals (see [TR-471]). Starts by 1 and is continuously incremented during a single test interval (testIntTime).

testIntTime: A two-octet field. Duration of the test (either downlink or uplink) with search algorithm in use, which serves as the maximum duration of the search process in seconds (see also TestInterval in Table 3 of [TR-471]).

dscpEcn: Diffserv code point and ECN field; see also the DSCP field specified by [TR-471]. This specification does not provide an ECN-capable transport; therefore, the sender **SHALL** set the ECN field to not_ECT.

srIndexConf: A two-octet field. 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 with 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.

useOwDelVar: A one-octet field. Boolean, default True (if False, use RTT=round-trip delay variation in the load rate adjustment algorithm; if True, use EnableIPDV, which uses one-way delay variation for the load rate adjustment algorithm). See EnableIPDV in Table 1 of [TR-471].

highSpeedDelta: A one-octet field; see Appendix A of [RFC9097].

slowAdjThresh and seqErrThresh: Two two-octet fields; see Appendix A of [RFC9097].

ignoreOooDup: A one-octet field. Ignore out-of-order duplicates, Boolean. When True, only loss counts toward received packet sequence number errors. When False, loss, out-of-order, and duplicate totals are all counted as sequence number errors. Default is True (see also Table 3 of [TR-471]).

modifierBitmap: A one-octet field. This document only assigns two bits in this bitmap; see Section 12.3.7:

CHTA_SRIDX_ISSTART (0x01):

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

CHTA_RAND_PAYLOAD (0x02): Randomize the payload content beyond the Load PDU header.

Other bit positions are left unassigned per this document.

rateAdjAlgo: A one-octet field. The applied load rate adjustment algorithm; see Section 12.3.8.

srStruct: Sending Rate structure. Used by the server in a Test Activation Response for an upstream test to communicate the (initial) Load PDU 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 fields are defined as follows:

txInterval1 and txInterval2: Two four-octet fields indicating the load rate transmit interval in us (microseconds). A 100 us granularity is recommended for optimal rate selection.

udpPayload1 and udpPayload2: Two four-octet fields indicating the UDP payload at load rate in bytes.

burstSize1 and burstSize2: Two four-octet fields indicating the burst size at load rate by a dimensionless number (of datagrams).

udpAddon2: A four-octet field indicating 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.

subIntPeriod: A two-octet field. Test Sub-Interval period in ms (see also Table 3 of [TR-471]). Trials with subIntPeriod in a range of 100 to 10000 ms resulted in a default value of 1000 ms.

authMode: Same as in Section 6.1.

authUnixTime: Same as in Section 6.1.

authDigest: Same as in Section 6.1.

keyId: Same as in Section 6.1.

reservedAuth1: Same as in Section 6.1.

checkSum: Same as in Section 6.1.

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

```
<CODE BEGINS>
//
// 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 pduId;       // PDU 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 reserved1;    // (reserved for alignment)
        uint8_t dscpEcn;      // Diffserv and ECN field for testing
#define CHTA_SRIDX_DEF UINT16_MAX // Request default server 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/Dup (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 reserved2;   // (reserved for alignment)
        struct sendingRate srStruct; // Sending Rate structure
        uint16_t subIntPeriod; // Sub-Interval period (ms)
        uint16_t reserved3;    // (reserved for alignment)
        uint16_t reserved4;    // (reserved for alignment)
        uint8_t reserved5;     // (reserved for alignment)
        // ========== Integrity Verification ==========
        uint8_t authMode;      // Authentication mode
        uint32_t authUnixTime; // Authentication timestamp
        uint8_t authDigest[AUTH_DIGEST_LENGTH];
```

```
        uint8_t keyId;         // Key ID in shared table
        uint8_t reservedAuth1; // (reserved for alignment)
        uint16_t checkSum;     // Header checksum
};

<CODE ENDS>
```

Figure 7: Test Activation PDU

7.2. Server Processes Test Activation Request and Generates Response

After the server receives the Test Activation Request on the new connection, it chooses 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.

When the server receives the Test Activation Request message, it **SHALL** first follow the Message Verification Procedure listed in Section 6.2, Paragraph 2.

7.2.1. 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, and rateAdjAlgo
- Test duration/intervals: trialInt, testIntTime, and subIntPeriod
- Packet marking: dscpEcn

Coercion is a step towards performing a test with the server-configured values; even though the client might prefer certain values, 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 cmdResponse field value (currently only CHTA_CRSP_BADPARAM; see Section 12.3.9).

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

- 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 the authDigest is invalid, the Test Activation Request **SHOULD** fail silently. The exception is for operations support: server administrators are permitted to send an Activation Response to support operations and troubleshooting.

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

- the Test Activation Request PDU size is not equal to the 'struct controlHdrTA' size shown in Figure 7,
- the PDU ID is not 0xACE2 (Test Activation PDU), or
- a directed attack has been detected.

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

7.2.2. Server Accepts Request and Generates Response

When the server sends the Test Activation Response, it **SHALL** set the cmdResponse field to CHTA_CRSP_ACKOK (see Section 12.3.9).

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 setting srIndexConf to a value of 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.

7.3. Client Processes Test Activation Response

When the client receives the Test Activation Response message, it **SHALL** first follow the Message Verification Procedure listed in Section 6.2, Paragraph 2.

After the client receives the (vetted) Test Activation Response, it first checks the Command Response value.

If the client receives a Test Activation cmdResponse field value that indicates an error, the client **SHALL** display/report a relevant message to the user or measurement system and exit.

If the client receives a Test Activation cmdResponse field value that is not equal to one of the codes defined in Section 12.3.9, 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 (e.g., CHTA_CRSP_ACKOK; see Section 12.3.9), the client **SHALL** update its configuration to use any test parameters modified by the server. If the setup parameters coerced by the server are not acceptable to the client, the client ends the test.

To finalize an accepted test activation, 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.

8. Test Load Stream Transmission and Measurement Status Feedback Messages

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

8.1. Load PDU and Roles

Testing proceeds with one endpoint sending Load PDUs, based on transmission parameters from the Sending Rate Table, and the other 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.

When a Load PDU is received, the receiver **SHALL** do the following:

1. Verify that the size of the message is greater than or equal to the 'struct loadHdr' size shown in Figure 9.
2. Validate the checksum for the Load PDU header portion of the total message (as described in Section 5.6) if the optional checkSum field is being utilized.
3. Confirm that the PDU ID is 0xBEEF (Load PDU).

If any of the above checks fail, the message **SHALL** be considered invalid.

The watchdog timer at the receiver **SHALL** be reset each time a valid Load PDU is received (which includes verification of the checkSum, if in use). See non-graceful test stop in Section 9 for handling the watchdog timeout expiration at each endpoint. Note that the watchdog timer's purpose is to detect a connection failure or a massive congestion condition only.

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 discrete 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 that evaluates measurements taken locally at the Load PDU receiver. When the client is the receiver, the information is communicated to the server via 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 are 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; see [Y.1540]) 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 that also backs 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 provides the option to retry 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 UDP PDU format layout **SHALL** be as follows (big-endian AB):

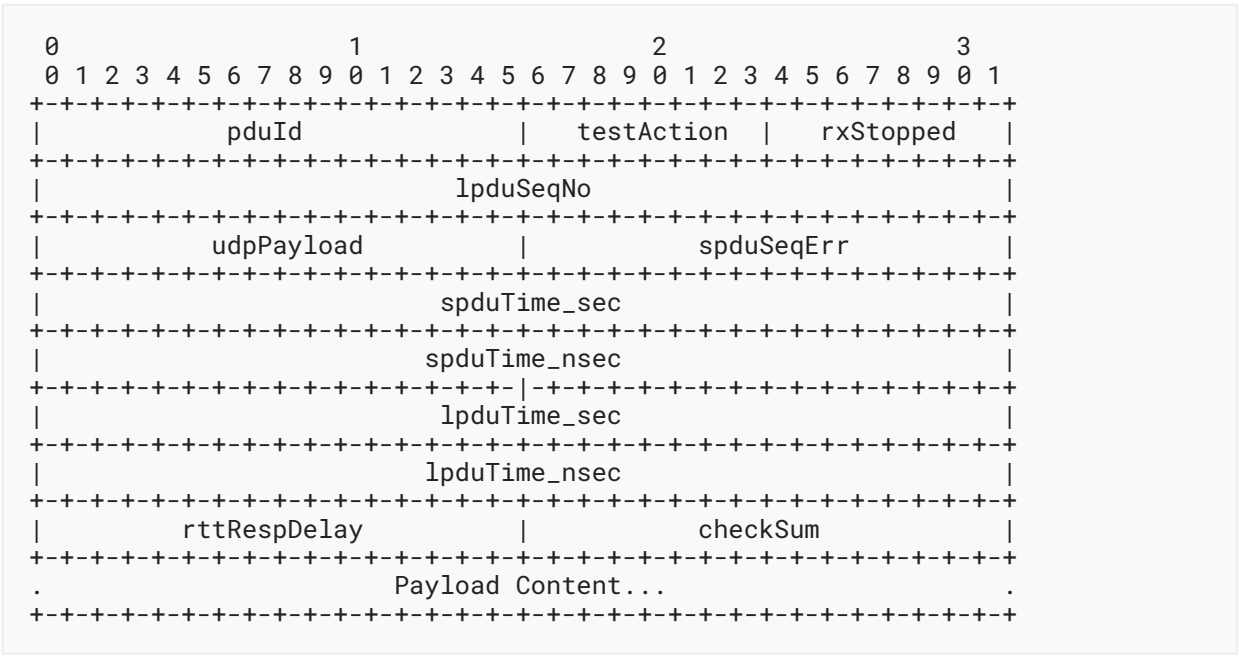


Figure 8: Load PDU Layout

Specific details regarding Load PDU fields are as follows:

pduId: IANA has assigned the hex value 0xBEEF (Section 12.3.1).

testAction: A one-octet field designating 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 9 for additional information on test termination.

rxStopped: A one-octet field. Boolean, 0 or 1, used to indicate to the remote endpoint 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: A four-octet field indicating the Load PDU sequence number (starting at 1). Used to determine loss, out-of-order, and duplicate totals.

udpPayload: A two-octet field indicating 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: A two-octet field indicating the 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: Two four-octet fields containing 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: Two four-octet fields containing the local send time of the Load PDU. Used for one-way delay variation measurements made by the Load PDU receiver.

rttRespDelay: A two-octet field indicating the RTT response delay, used to "adjust" raw RTT. On the Load PDU sender, it is the number of ms 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 (see Section 5.6 for guidance). The checksum does not cover the payload content. The calculation is done as the very last step of building the PDU header, with the checkSum field set to zero.

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

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

```
<CODE BEGINS>
//
// Load header for UDP payload of Load PDUs
//
struct loadHdr {
#define LOAD_ID 0xBEEF
        uint16_t pduId;  // PDU 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 (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 rx'd status PDU
        uint32_t spduTime_nsec; // Send time in last rx'd 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
};

<CODE ENDS>
```

Figure 9: Load PDU

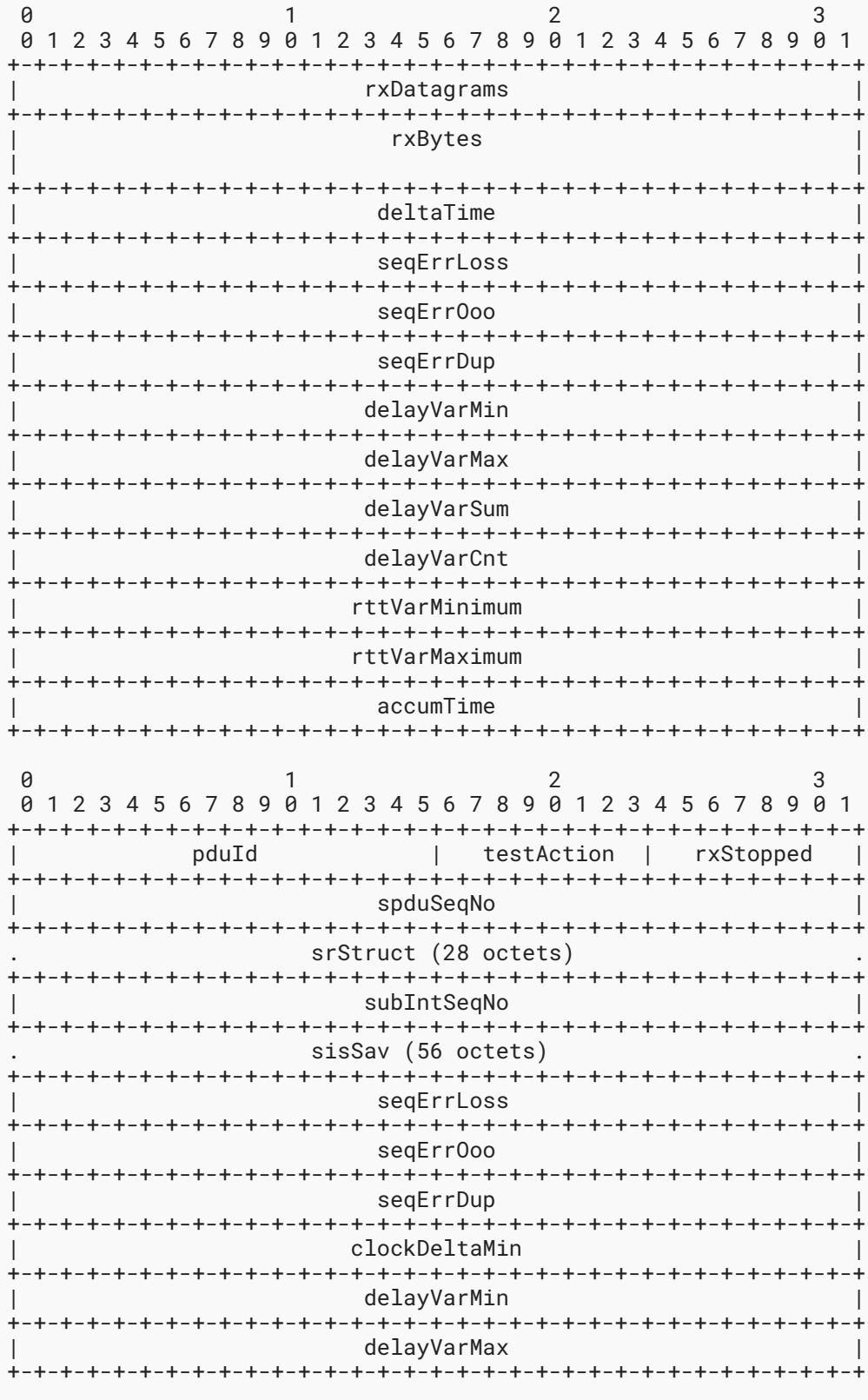
8.2. Status PDU

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.

When the Load PDU sender receives a Status PDU message, it **SHALL** first follow the Message Verification Procedure listed in Section 6.2, Paragraph 2.

The watchdog timer at the Load PDU sender **SHALL** be reset each time a valid Status PDU is received (which includes verification of the checkSum and/or authDigest, if in use). See non-graceful test stop in Section 9 for handling the watchdog timeout expiration at each endpoint.

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


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
rxDatagrams
rxBytes
deltaTime
seqErrLoss
seqErrOoo
seqErrDup
delayVarMin
delayVarMax
delayVarSum
delayVarCnt
rttVarMinimum
rttVarMaximum
accumTime
0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
pduId
testAction
rxStopped
spduSeqNo
srStruct (28 octets)
subIntSeqNo
sisSav (56 octets)
seqErrLoss
seqErrOoo
seqErrDup
clockDeltaMin
delayVarMin
delayVarMax

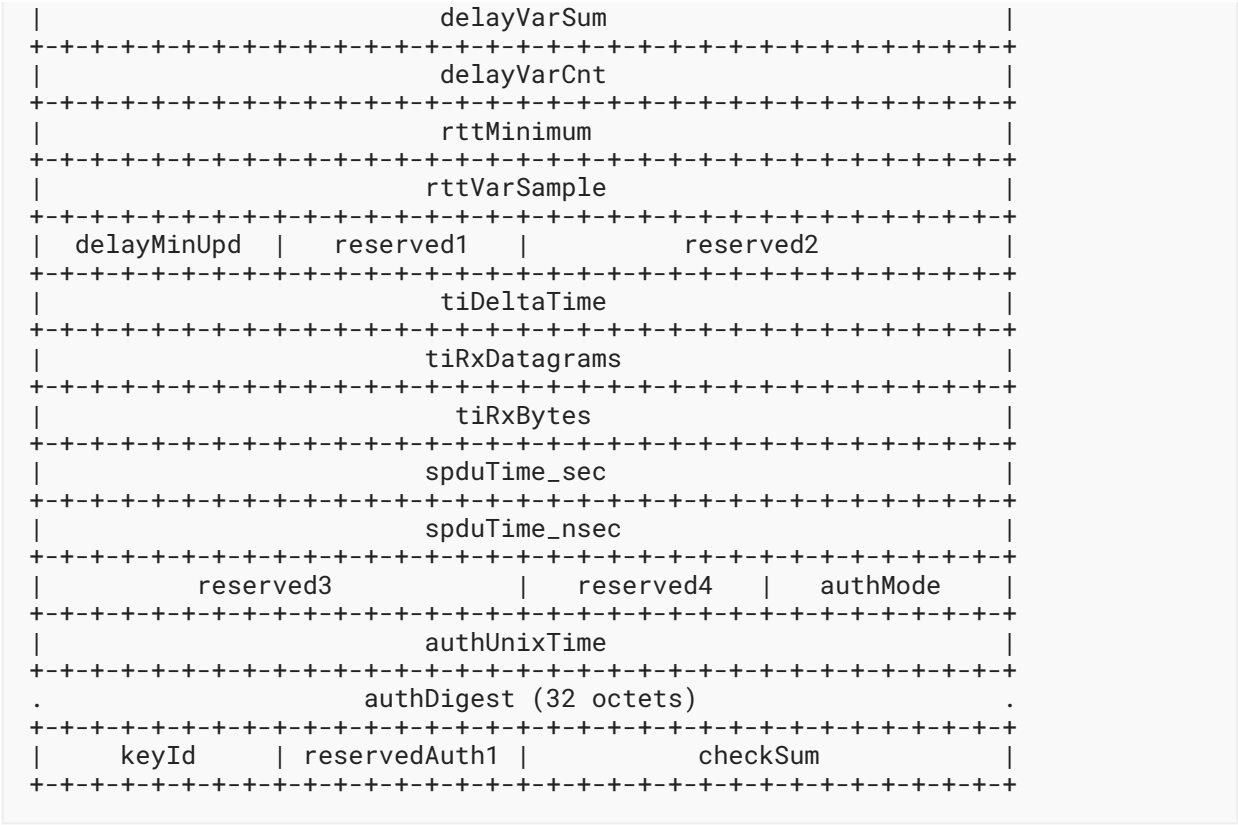


Figure 10: Status PDU Layout

Note that the Sending Rate structure is defined in Section 7.

The primary role of the Status Feedback message is to communicate the traffic conditions at the Load PDU receiver to the Load PDU sender. While the Sub-Interval statistics saved (sisSav) structure 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]. The authentication and checkSum fields follow the same methodology as with the Setup Request and Response. Additional information regarding fields not defined previously are as follows:

pduId: IANA has assigned the hex value 0xFEED (Section 12.3.1).

spduSeqNo: A four-octet field containing the 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: A four-octet field containing the 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). These consist of the following fields:

rxDatagrams: A four-octet field Sub-Interval indicating the number of received datagrams during the Sub-Interval.

rxBytes: An eight-octet field indicating the Sub-Interval byte count (eight octets chosen to prevent overflow at high speeds).

deltaTime: A four-octet field indicating the exact duration of the Sub-Interval in us. Used to calculate the received traffic rate together with rxBytes.

seqErrLoss/seqErrOoo/seqErrDup: Three four-octet fields populated by the loss, out-of-order, and duplicate totals. Available for both the Sub-Interval and trial interval; it is a breakout of the SeqErrors count in Table 3 of [TR-471]. 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 UDPSTP 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: Four four-octet fields populated by 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): Two four-octet fields populated by the minimum and maximum RTT delay variation (rttVarSample) in the Sub-Interval designated by the subIntSeqNo.

accumTime: The accumulated time of the test in ms, 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.

clockDeltaMin: A four-octet field indicating 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 endpoint clocks are sufficiently synchronized, this will be the minimum one-way delay in ms. Otherwise, this value may be negative but still valid for one-way delay variation measurements for the default test duration (default is 10 seconds). 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): A four-octet field indicating the minimum "adjusted" RTT measured since the beginning of the test. See rttRespDelay (Section 8.1) 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_NODEL when communicated in the Status PDU.

rttVarSample: A four-octet field indicating the most recent "adjusted" RTT delay variation measurement. See rttRespDelay (Section 8.1) 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_NODEL when communicated in the Status PDU.

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

tiDeltaTime/tiRxDatagrams/tiRxBytes: Three four-octet fields populated by the trial interval time in us, along with the received datagram and byte counts. Used to calculate the received traffic rate for the trial interval.

spduTime_sec/spduTime_nsec: Two four-octet fields containing 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.

authMode: Same as in Section 6.1.

authUnixTime: Same as in Section 6.1.

authDigest: Same as in Section 6.1.

keyId: Same as in Section 6.1.

reservedAuth1: Same as in Section 6.1.

checkSum: Same as in Section 6.1.

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

```
<CODE BEGINS>
//
// Sub-Interval statistics structure for received traffic information
//
#pragma pack(push, 1)
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 rttVarMinimum; // Minimum RTT variation (ms)
        uint32_t rttVarMaximum; // Maximum RTT variation (ms)
        uint32_t accumTime;     // Accumulated time (ms)
};
#pragma pack(pop)
//
// Status Feedback header for UDP payload of status PDUs
//
struct statusHdr {
#define STATUS_ID 0xFEED
        uint16_t pduId;     // PDU ID
        uint8_t testAction; // Test action
        uint8_t rxStopped;  // Receive traffic stopped (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 stats saved
        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_NODEL UINT32_MAX // No delay data/value
        uint32_t rttMinimum;    // Min 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 reserved2;     // (reserved for alignment)
        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
        uint16_t reserved3;     // (reserved for alignment)
        uint8_t reserved4;      // (reserved for alignment)
        // ========== Integrity Verification ==========
        uint8_t authMode;      // Authentication mode
        uint32_t authUnixTime; // Authentication timestamp
```

```
        uint8_t authDigest[AUTH_DIGEST_LENGTH];
        uint8_t keyId;         // Key ID in shared table
        uint8_t reservedAuth1; // (reserved for alignment)
        uint16_t checkSum;     // Header checksum
};

<CODE ENDS>
```

Figure 11: Status PDU

9. Stopping 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 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 a Status PDU, **SHALL** result in it and any subsequent PDUs being 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 that 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 endpoint will expire (sometimes at one endpoint first); notifications in logs, STDOUT, and/or formatted output **SHALL** be made; and the 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 endpoint **SHALL** schedule an immediate end time for the connection.

10. Operational Considerations for the Measurement Method

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 nominal duration of a measurement interval at the Destination, parameter testIntTime, **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.

It is **RECOMMENDED** to locate test endpoints as close to the intended measured link(s) as practical. The testing operator **MUST** set a value for the MaxHops parameter, based on the expected path length. This parameter can keep measurement traffic from straying too far beyond the intended path.

It is obviously counterproductive to run more than one independent and concurrent test (regardless of the number of flows in the test stream) when attempting to measure the maximum capacity on a single path. The number of concurrent, independent tests of a path **SHALL** be limited to one.

The load rate adjustment algorithm's scope is limited to helping determine the Maximum IP-Layer Capacity in the context of an infrequent, diagnostic, short-term measurement. It is **RECOMMENDED** to discontinue non-measurement traffic that shares a subscriber's dedicated resources while testing: measurements may not be accurate, and throughput of competing elastic traffic may be greatly reduced.

See Section 8 of [RFC9097] for a discussion of the Method of Measurement beyond purely operational aspects.

10.1. 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.

11. 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 users activating measurements as a service or users whose traffic is measured, the sensitive information available to potential observers is greatly reduced when using active techniques that are within this scope of work. Passive observations of user traffic for measurement purposes raise many privacy issues. See the privacy considerations described in the LMAP Framework [RFC7594], which covers active and passive techniques.

Below 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 the client roles. One way to assure a cooperative agreement employs the optional Authorization mode is through the use of the authDigest field and the known identity associated with the shared key used to create the authDigest field via the KDF. 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 that goes to either a control port or ephemeral ports that are only created as needed. Firewalls protecting each host can continue to do their jobs normally.
3. Client-server authentication and integrity protection for feedback messages conveying measurements is **RECOMMENDED**. To accommodate different host limitations and testing circumstances, different modes of operation are available, as described in Section 5.
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 the 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.

Authentication methods and requirements steadily evolve. Alternate authentication modes provide for algorithm agility by defining a new mode, whose support is indicated by assigning a suitable "Test Setup PDU Authentication Mode" registry value (see Section 12.3.4).

12. IANA Considerations

Per this document, IANA has assigned a UDP port number for the Test Setup exchange in the Control phase of protocol operation and has created a new registry group for the UDPSTP.

12.1. New User Port Number Assignment

IANA has registered the following service name in the "Service Name and Transport Protocol Port Number Registry":

Service Name: udpstp
Port Number: 24601
Transport Protocol: udp
Description: UDP-based IP-Layer Capacity and Performance Measurement protocol
Assignee: IESG <iesg@ietf.org>
Contact: IETF Chair <chair@ietf.org>
Reference: RFC 9946

The protocol uses IP-Layer Unicast. A single port number was assigned to help configure firewalls and other port-based systems for access control prior to negotiating dynamic ports between client and server.

12.2. New KeyTable KDF

IANA has added the following KDF entry to the "KeyTable KDFs" registry (see <https://www.iana.org/assignments/keytable>):

KDF	Description	Reference
HMAC-SHA-256	HMAC using the SHA-256 hash	[RFC6234]

Table 1: KeyTable KDFs Registry

12.3. New UDPSTP Registry Group

IANA has created the registries in the subsections that follow under a new registry group called "UDP Speed Test Protocol (UDPSTP)".

IANA has added the following note under the "UDP Speed Test Protocol (UDPSTP)" registry group:

> Note: Values reserved for Experimental Use are not expected to be used on the Internet but are expected to be used for experiments that are confined to closed environments.

12.3.1. PDU Identifier Registry

IANA has created the "PDU Identifier" registry under the "UDP Speed Test Protocol (UDPSTP)" registry group. Every UDPSTP PDU contains a two-octet pduId field identifying 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 2.

Range	Registration Procedures
0x0000	Reserved
0x0001-0x7F00	Specification Required
0x7F01-0x7FE0	Experimental Use
0x7FE1-0x7FFF	Private Use
0x8000-0xFFFE	IETF Review
0xFFFF	Reserved

Table 2: Registration Procedures for the PDU Identifier Registry

IANA has assigned values in the "PDU Identifier" registry as follows:

Value	Description	Reference
0x0000	Reserved	RFC 9946
0x7F01-0x7FE0	Reserved for Experimental Use	RFC 9946
0x7FE1-0x7FFF	Reserved for Private Use	RFC 9946
0xACE1	Test Setup PDU	RFC 9946
0xACE2	Test Activation PDU	RFC 9946
0xBEEF	Load PDU	RFC 9946
0xDEAD	Null PDU	RFC 9946
0xFEED	Status Feedback PDU	RFC 9946
0xFFFF	Reserved	RFC 9946

Table 3: Initial Values of the PDU Identifier Registry

12.3.2. Protocol Version Registry

IANA has created the "Protocol Version" registry under the "UDP Speed Test Protocol (UDPSTP)" registry group. UDPSTP Test Setup Request, Test Setup Response, and Test Activation Request PDUs contain a two-octet protocolVer field, identifying the version of the protocol in use. The code points in this registry are allocated according to the registration procedures [RFC8126] described in Table 4.

Range	Registration Procedures
0-19	Reserved
20-40960	IETF Review
40961-53248	Specification Required
53249-65534	Experimental Use
65535	Reserved

Table 4: Registration Procedures for the Protocol Version Registry

IANA has assigned decimal value 20 in the "Protocol Version" registry as follows:

Value	Reference
20	RFC 9946

Table 5: Initial Value of the Protocol Version Registry

12.3.3. Test Setup PDU Modifier Bitmap Registry

IANA has created 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 6.

Range	Registration Procedures
00000000-01111111	IETF Review
10000000	Reserved

Table 6: Registration Procedures for the Test Setup PDU Modifier Bitmap Registry

IANA has assigned bitmap values in the "Test Setup PDU Modifier Bitmap" registry as follows:

Value	Description	Reference
0x00	No modifications	RFC 9946
0x01	Allow Jumbo datagram sizes above sending rates of 1 Gbps	RFC 9946
0x02	Use Traditional MTU (1500 bytes with an IP header)	RFC 9946

Table 7: Initial Values of the Test Setup PDU Modifier Bitmap Registry

12.3.4. Test Setup PDU Authentication Mode Registry

IANA has created the "Test Setup PDU Authentication Mode" 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 8.

Range	Registration Procedures
0-59	IETF Review
60-63	Experimental Use
64-255	Reserved

Table 8: Registration Procedures for the Test Setup PDU Authentication Mode Registry

IANA has assigned decimal values in the "Test Setup PDU Authentication Mode" registry as follows:

Value	Description	Reference
0	Not used	RFC 9946
1	Required authentication for the Control phase	RFC 9946
2	Required authentication for the Control and Data phases	RFC 9946

Table 9: Initial Values of the Test Setup PDU Authentication Mode Registry

12.3.5. Test Setup PDU Command Response Field Registry

IANA has created the "Test Setup PDU Command Response Field" registry under 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 10.

Range	Registration Procedures
0-127	IETF Review
128-239	Specification Required
240-249	Experimental Use
250-254	Private Use
255	Reserved

Table 10: Registration Procedures for the Test Setup PDU Command Response Field Registry

IANA has assigned decimal values in the "Test Setup PDU Command Response Field" registry as follows:

Value	Description	Reference
0	None (used by client in Request)	RFC 9946
1	Acknowledgment	RFC 9946
2	Bad protocol version	RFC 9946
3	Invalid Jumbo datagram option	RFC 9946
4	Unexpected authentication in Setup Request	RFC 9946
5	Authentication missing in Setup Request	RFC 9946
6	Invalid authentication method	RFC 9946
7	Authentication failure	RFC 9946
8	Authentication time is invalid in Setup Request	RFC 9946
9	No maximum test bit rate specified	RFC 9946
10	Server maximum bit rate exceeded	RFC 9946
11	MTU option does not match server	RFC 9946
12	Multi-connection parameters rejected by server	RFC 9946
13	Connection allocation failure on server	RFC 9946
240-249	Reserved for Experimental Use	RFC 9946

Value	Description	Reference
250-254	Reserved for Private Use	RFC 9946
255	Reserved	RFC 9946

Table 11: Initial Values of the Test Setup PDU Command Response Field Registry

Note that value 4 is required for backward compatibility with previous experimental versions of software already in use. Further, value 6 signals that a client erroneously used an authMode that hasn't been standardized yet (i.e., authMode is greater than 1 or 2).

12.3.6. Test Activation PDU Command Request Registry

IANA has created the "Test Activation PDU Command Request" registry under the "UDP Speed Test Protocol (UDPSTP)" registry group. 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 12.

Range	Registration Procedures
0-127	IETF Review
128-239	Specification Required
240-249	Experimental Use
250-254	Private Use
255	Reserved

Table 12: Registration Procedures for the Test Activation PDU Command Request Registry

IANA has assigned decimal values in the "Test Activation PDU Command Request" registry as follows:

Value	Description	Reference
0	No Request	RFC 9946
1	Request test in upstream direction (client to server)	RFC 9946
2	Request test in downstream direction (server to client)	RFC 9946
240-249	Reserved for Experimental Use	RFC 9946

Value	Description	Reference
250-254	Reserved for Private Use	RFC 9946
255	Reserved	RFC 9946

Table 13: Initial Values of the Test Activation PDU Command Request Registry

12.3.7. Test Activation PDU Modifier Bitmap Registry

IANA has created the "Test Activation PDU Modifier Bitmap" registry under the "UDP Speed Test Protocol (UDPSTP)" registry group. 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 14.

Range	Registration Procedures
00000000-01111111	IETF Review
10000000	Reserved

Table 14: Registration Procedures for the Test Activation PDU Modifier Bitmap Registry

IANA has assigned bitmap values in the "Test Activation PDU Modifier Bitmap" registry as follows:

Value	Description	Reference
0x00	No modifications	RFC 9946
0x01	Set when srIndexConf is start rate for search	RFC 9946
0x02	Set for randomized UDP payload	RFC 9946

Table 15: Initial Values of the Test Activation PDU Modifier Bitmap Registry

12.3.8. Test Activation PDU Rate Adjustment Algo Registry

IANA has created the "Test Activation PDU Rate Adjustment Algo" registry under the "UDP Speed Test Protocol (UDPSTP)" registry group. 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 16.

Range	Registration Procedures
A-Y	IETF Review

Range	Registration Procedures
Z	Reserved

Table 16: Registration Procedures for the Test Activation PDU Rate Adjustment Algo Registry

IANA has added the following note under the "Test Activation PDU Rate Adjustment Algo" registry:

> Note: The algorithm identifier is a capitalized alphabetic UTF-8 value (A-Z), specified by the corresponding incremental numeric.

IANA has assigned capitalized alphabetic UTF-8 values, as well as the corresponding incremental numeric values, in the "Test Activation PDU Rate Adjustment Algo" registry as follows:

Value(Numeric)	Description	Reference
A(n/a)	Not used	RFC 9946
B(0)	Rate algorithm Type B	[Y.1540Amd2]
C(1)	Rate algorithm Type C	[TR-471]

Table 17: Initial Values of the Test Activation PDU Rate Adjustment Algo Registry

12.3.9. Test Activation PDU Command Response Field Registry

IANA has created the "Test Activation PDU Command Response Field" registry under the "UDP Speed Test Protocol (UDPSTP)" registry group. 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 18.

Range	Registration Procedures
0-127	IETF Review
128-239	Specification Required
240-249	Experimental Use
250-254	Private Use

Range	Registration Procedures
255	Reserved

Table 18: Registration Procedures for the Test Activation PDU Command Response Field Registry

IANA has assigned decimal values in the "Test Activation PDU Command Response Field" registry as follows:

Value	Description	Reference
0	None (used by client in Request)	RFC 9946
1	Server acknowledgment	RFC 9946
2	Server indicates an error	RFC 9946
240-249	Reserved for Experimental Use	RFC 9946
250-254	Reserved for Private Use	RFC 9946
255	Reserved	RFC 9946

Table 19: Initial Values of the Test Activation PDU Command Response Field Registry

12.4. Guidelines for Designated Experts

It is suggested that multiple designated experts be appointed for registry change requests.

Criteria that should be applied by the designated experts include determining whether the proposed registration duplicates existing entries and whether the registration description is clear and fits the purpose of this registry.

Registration requests are evaluated within a two-week review period on the advice of one or more designated experts. Within the review period, the designated experts will either approve or deny the registration request, communicating this decision to IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful.

13. References

13.1. Normative References

[C-Prog] ISO/IEC, "Programming languages -- C", ISO/IEC 9899:1999, 1999, <https://www.iso.org/standard/29237.html>.

[NIST800-108] Chen, L., "Recommendation for Key Derivation Using Pseudorandom Functions", DOI 10.6028/NIST.SP.800-108r1-upd1, NIST SP 800-108r1-upd1, August 2022, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-108r1-upd1.pdf>.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[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>.

[RFC5044] Culley, P., Elzur, U., Recio, R., Bailey, S., and J. Carrier, "Marker PDU Aligned Framing for TCP Specification", RFC 5044, DOI 10.17487/RFC5044, October 2007, <https://www.rfc-editor.org/info/rfc5044>.

[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>.

[RFC8899] Fairhurst, G., Jones, T., Tüxen, M., Rüngeler, I., and T. Völker, "Packetization Layer Path MTU Discovery for Datagram Transports", RFC 8899, DOI 10.17487/RFC8899, September 2020, <https://www.rfc-editor.org/info/rfc8899>.

[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>.

[TR-471] Broadband Forum, "Maximum IP-Layer Capacity Metric, Related Metrics, and Measurements", TR-471, Issue 4, September 2024, <https://www.broadband-forum.org/pdfs/tr-471-4-0-0.pdf>.

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

[Y.1540Amd2] ITU-T, "Internet protocol data communication service - IP packet transfer and availability performance parameters, Amendment 2 - Revised Annex B: Additional search algorithms for IP-based capacity parameters and methods of measurement", ITU-T Recommendation Y.1540 Amd. 2, March 2023, <https://www.itu.int/rec/T-REC-Y.1540-202303-I!Amd2/en>.

13.2. Informative References

[EVP_KDF-KB] "EVP_KDF-KB - The Key-Based EVP_KDF implementation", OpenSSL Documentation, <https://docs.openssl.org/master/man7/EVP_KDF-KB/>.

[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>.

[RFC9145] Boucadair, M., Reddy.K, T., and D. Wing, "Integrity Protection for the Network Service Header (NSH) and Encryption of Sensitive Context Headers", RFC 9145, DOI 10.17487/RFC9145, December 2021, <https://www.rfc-editor.org/info/rfc9145>.

Appendix A. KDF Example (OpenSSL)

```
<CODE BEGINS>
//
// Output individual authentication keys of length SHA256_KEY_LEN
// from derived key material.
//
// Return Values: 0 = Failure, 1 = Success
//
int kdf_hmac_sha256(char *Kin, uint32_t authUnixTime,
  unsigned char *cAuthKey,   // Client key
  unsigned char *sAuthKey) { // Server key

  int var, keylen = SHA256_KEY_LEN * 2;
  char context[16];
  unsigned char *keyptr, keybuf[keylen];
  EVP_KDF *kdf = NULL;
  EVP_KDF_CTX *kctx = NULL;
  OSSL_PARAM params[16], *p = params;

  //
  // Fetch KDF algorithm and create context
  //
  if ((kdf = EVP_KDF_fetch(NULL, "KBKDF", NULL)) == NULL) {
    return 0;
  }
  if ((kctx = EVP_KDF_CTX_new(kdf)) == NULL) {
    EVP_KDF_free(kdf);
    return 0;
  }

  //
  // Set parameters for KBKDF
  // ----------------------------------------------------------
  *p++ = OSSL_PARAM_construct_utf8_string(
    OSSL_KDF_PARAM_MODE, "COUNTER", 0);
  *p++ = OSSL_PARAM_construct_utf8_string(
    OSSL_KDF_PARAM_MAC, "HMAC", 0);
  *p++ = OSSL_PARAM_construct_utf8_string(
    OSSL_KDF_PARAM_DIGEST, "SHA256", 0);
  *p++ = OSSL_PARAM_construct_octet_string(
    OSSL_KDF_PARAM_KEY, Kin, strlen(Kin));
  *p++ = OSSL_PARAM_construct_octet_string(
    OSSL_KDF_PARAM_SALT, "UDPSTP", 6);
  var = snprintf(context, sizeof(context), "%u", authUnixTime);
  *p++ = OSSL_PARAM_construct_octet_string(
    OSSL_KDF_PARAM_INFO, context, var);
  //
  // Confirm the following are enabled
  //
  var = 1;
  *p++ = OSSL_PARAM_construct_int(OSSL_KDF_PARAM_KBKDF_USE_L, &var);
  *p++ = OSSL_PARAM_construct_int(
    OSSL_KDF_PARAM_KBKDF_USE_SEPARATOR, &var);
  //
  // Set counter length in bits (available as of OpenSSL 3.1)
  //
  var = 32; // Length of 32 is backward compatible with OpenSSL 3.0
  *p++ = OSSL_PARAM_construct_int(OSSL_KDF_PARAM_KBKDF_R, &var);
```

```
   *p++ = OSSL_PARAM_construct_end();
   // ------------------------------------------------------

   //
   // Derive key material
   //
   if (EVP_KDF_derive(kctx, keybuf, keylen, params) < 1) {
     EVP_KDF_CTX_free(kctx);
     EVP_KDF_free(kdf);
     return 0;
   }

   //
   // Output individual keys
   //
   keyptr = keybuf;
   memcpy(cAuthKey, keyptr, SHA256_KEY_LEN);
   keyptr += SHA256_KEY_LEN;
   memcpy(sAuthKey, keyptr, SHA256_KEY_LEN);

   //
   // Cleanup
   //
   EVP_KDF_CTX_free(kctx);
   EVP_KDF_free(kdf);
   return 1;
}

<CODE ENDS>
```

Figure 12: KDF Example Code Snippet

Acknowledgments

This document was edited by Al Morton, who passed away before being able to finalize this work. Ruediger Geib only joined later to help finalize this specification.

Thanks to Lincoln Lavoie, Can Desem, Greg Mirsky, Bjoern Ivar Teigen, Ken Kerpez, and Chen Li for reviewing this specification and providing helpful suggestions and areas for further development. Mohamed Boucadair's AD review improved comprehensibility of the document, and he provided helpful guidance well through the final review stages. Tommy Pauly shepherded this document. Further comments by Gorry Fairhurst, Éric Vyncke, Roman Danyliw, Gunter Van de Velde, Deb Cooley, Tianran Zhou, Andy Newton, Giuseppe Fioccola, Lars Eggert, Erik Kline, and Benson Muite helped to shape the document. David Dong and Amanda Baber provided early reviews of the IANA Considerations section.

Starting with the early SEC-DIR review, Brian Weis provided constructive guidance regarding numerous security-related protocol issues. The Crypto Forum Research Group reviewed these parts, again providing guidance. Magnus Westerlund's review resulted in further changes and refinements. Ultimately, Paul Wouters' feedback was critical in finalizing the chosen security approach.

Authors' Addresses

Al Morton
AT&T Labs

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

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