Network Working Group
Internet Draft
Expiration Date: June 2005

Stanislav Shalunov
Benjamin Teitelbaum
Anatoly Karp
Jeff W. Boote
Matthew J. Zekauskas
Internet2
December 2004

A One-way Active Measurement Protocol (OWAMP)
<draft-ietf-ippm-owdp-12.txt>

Status of this Memo

Copyright Notice

Abstract

With growing availability of good time sources to network nodes, it becomes increasingly possible to measure one-way IP performance metrics with high precision. To do so in an interoperable manner, a common protocol for such measurements is required. The One-Way Active Measurement Protocol (OWAMP) can measure one-way delay, as well as other unidirectional characteristics, such as one-way loss.

Table of Contents

1. Introduction

The IETF IP Performance Metrics (IPPM) working group has proposed draft standard metrics for one-way packet delay [RFC2679] and loss [RFC2680] across Internet paths. Although there are now several measurement platforms that implement collection of these metrics [SURVEYOR], [RIPE], there is not currently a standard that would permit initiation of test streams or exchange of packets to collect singleton metrics in an interoperable manner.

With the increasingly wide availability of affordable global positioning systems (GPS) and CDMA-based time sources, hosts increasingly have available to them very accurate time sources--either directly or through their proximity to Network Time Protocol (NTP) primary (stratum 1) time servers. By standardizing a technique for collecting IPPM one-way active measurements, we hope to create an environment where IPPM metrics may be collected across a far broader mesh of Internet paths than is currently possible. One particularly compelling vision is of widespread deployment of open OWAMP servers that would make measurement of one-way delay as commonplace as measurement of round-trip time using an ICMP-based tool like ping.

Additional design goals of OWAMP include being hard to detect and manipulate, security, logical separation of control and test functionality, and support for small test packets.

OWAMP test traffic is hard to detect because it is simply a stream of UDP packets from and to negotiated port numbers, with potentially nothing static in the packets (size is negotiated, as well). OWAMP also supports an encrypted mode that further obscures the traffic, at the same time making it impossible to alter timestamps undetectably.

Security features include optional authentication and/or encryption of control and test messages. These features may be useful to prevent unauthorized access to results or man-in-the-middle attackers

who attempt to provide special treatment to OWAMP test streams or who attempt to modify sender-generated timestamps to falsify test results.

The key words "MUST", "REQUIRED", "SHOULD", "RECOMMENDED", and "MAY" in this document are to be interpreted as described in [RFC2119].

1.1. Relationship of Test and Control Protocols

OWAMP actually consists of two inter-related protocols: OWAMP-Control and OWAMP-Test. OWAMP-Control is used to initiate, start, and stop test sessions and fetch their results, while OWAMP-Test is used to exchange test packets between two measurement nodes.

Although OWAMP-Test may be used in conjunction with a control protocol other than OWAMP-Control, the authors have deliberately chosen to include both protocols in the same draft to encourage the implementation and deployment of OWAMP-Control as a common denominator control protocol for one-way active measurements. Having a complete and open one-way active measurement solution that is simple to implement and deploy is crucial to assuring a future in which inter-domain one-way active measurement could become as commonplace as ping. We neither anticipate nor recommend that OWAMP-Control form the foundation of a general-purpose extensible measurement and monitoring control protocol.

OWAMP-Control is designed to support the negotiation of one-way active measurement sessions and results retrieval in a straightforward manner. At session initiation, there is a negotiation of sender and receiver addresses and port numbers, session start time, session length, test packet size, the mean Poisson sampling interval for the test stream, and some attributes of the very general RFC 2330 notion of packet type, including packet size and per-hop behavior (PHB) [RFC2474], which could be used to support the measurement of one-way network characteristics across differentiated services networks. Additionally, OWAMP-Control supports per-session

encryption and authentication for both test and control traffic, measurement servers that can act as proxies for test stream endpoints, and the exchange of a seed value for the pseudo-random Poisson process that describes the test stream generated by the sender.

We believe that OWAMP-Control can effectively support one-way active measurement in a variety of environments, from publicly accessible measurement beacons running on arbitrary hosts to network monitoring deployments within private corporate networks. If integration with Simple Network Management Protocol (SNMP) or proprietary network management protocols is required, gateways may be created.

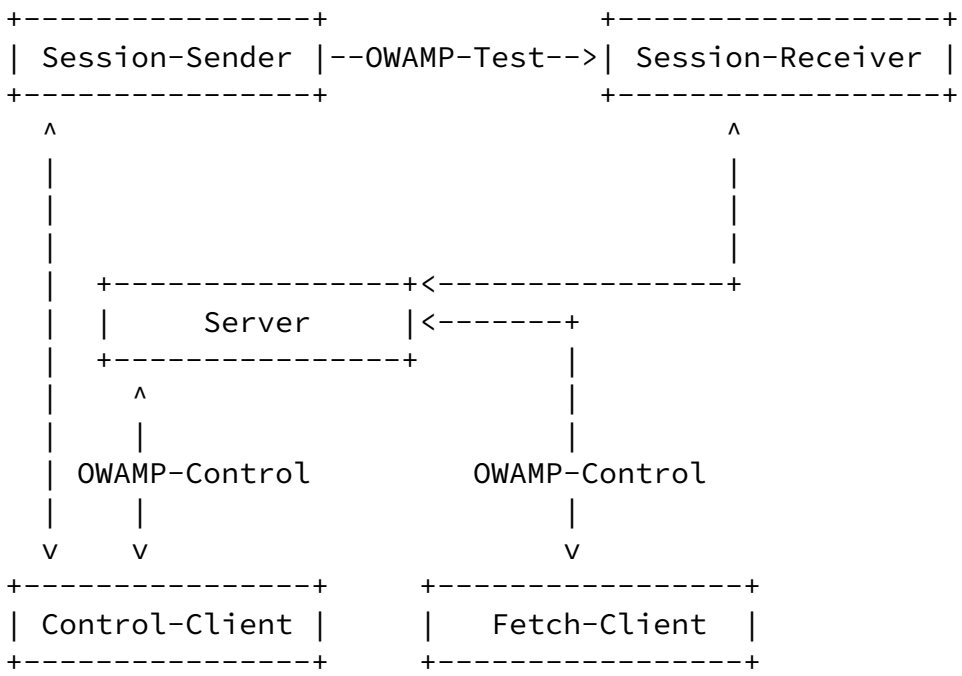
1.2. Logical Model

Several roles are logically separated to allow for broad flexibility in use. Specifically, we define:

Session-Sender	the sending endpoint of an OWAMP-Test session;
Session-Receiver	the receiving endpoint of an OWAMP-Test session;
Server	an end system that manages one or more OWAMP-Test sessions, is capable of configuring per-session state in session endpoints, and is capable of returning the results of a test session;
Control-Client	an end system that initiates requests for OWAMP-Test sessions, triggers the start of a set of sessions, and may trigger their termination; and
Fetch-Client	an end system that initiates requests to fetch the results of completed OWAMP-Test sessions.

One possible scenario of relationships between these roles is shown

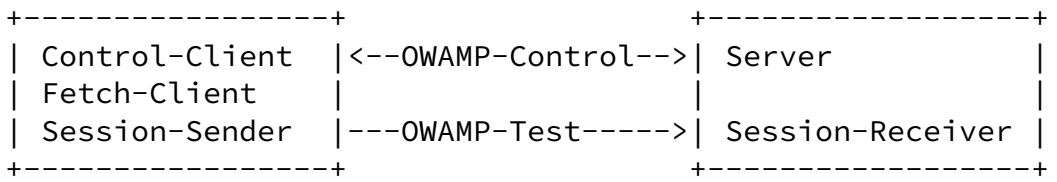
below.



(Unlabeled links in the figure are unspecified by this draft and may be proprietary protocols.)

Different logical roles can be played by the same host. For example, in the figure above, there could actually be only two hosts: one

playing the roles of Control-Client, Fetch-Client, and Session-Sender, and the other playing the roles of Server and Session-Receiver. This is shown below.



Finally, because many Internet paths include segments that transport IP over ATM, delay and loss measurements can include the effects of ATM segmentation and reassembly (SAR). Consequently, OWAMP has been designed to allow for small test packets that would fit inside the payload of a single ATM cell (this is only achieved in

unauthenticated and encrypted modes).

2. Protocol Overview

As described above, OWAMP consists of two inter-related protocols: OWAMP-Control and OWAMP-Test. The former is layered over TCP and is used to initiate and control measurement sessions and to fetch their results. The latter protocol is layered over UDP and is used to send singleton measurement packets along the Internet path under test.

The initiator of the measurement session establishes a TCP connection to a well-known port on the target point and this connection remains open for the duration of the OWAMP-Test sessions. IANA will be requested to allocate a well-known port number for OWAMP-Control sessions. An OWAMP server SHOULD listen to this well-known port.

OWAMP-Control messages are transmitted only before OWAMP-Test sessions are actually started and after they complete (with the possible exception of an early Stop-Sessions message).

The OWAMP-Control and OWAMP-Test protocols support three modes of operation: unauthenticated, authenticated, and encrypted. The authenticated or encrypted modes require endpoints to possess a shared secret.

All multi-octet quantities defined in this document are represented as unsigned integers in network byte order unless specified otherwise.

3. OWAMP-Control

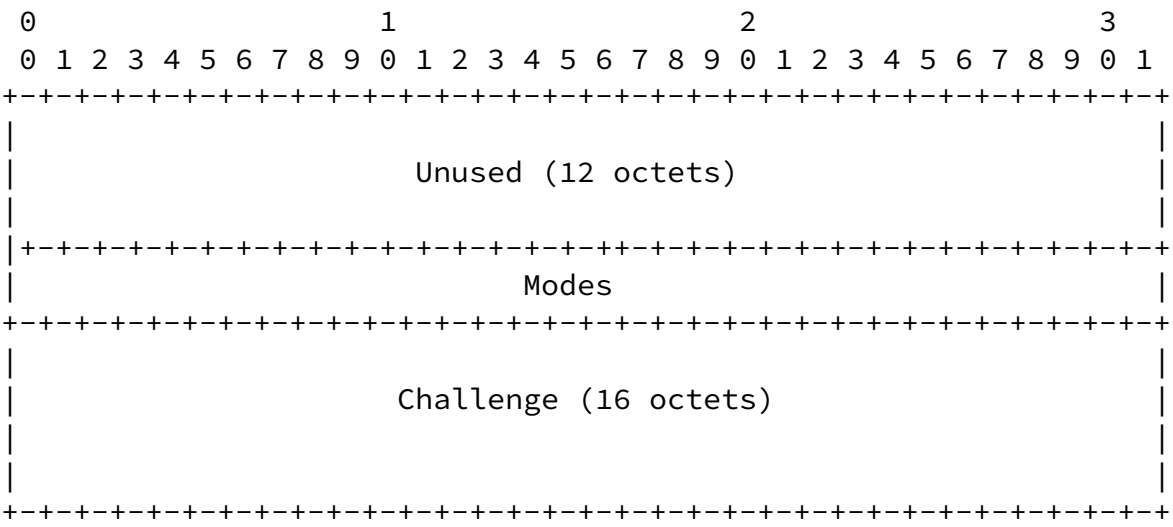
Each type of OWAMP-Control message has a fixed length. The recipient will know the full length of a message after examining the first 16 octets of it. No message is shorter than 16 octets.

If the full message is not received within 30 minutes after it is expected, connection SHOULD be dropped.

3.1. Connection Setup

Before either a Control-Client or a Fetch-Client can issue commands of a Server, it has to establish a connection to the server.

First, a client opens a TCP connection to the server on a well-known port. The server responds with a server greeting:



The following Mode values are meaningful: 1 for unauthenticated, 2 for authenticated, and 4 for encrypted. The value of the Modes field sent by the server is the bit-wise OR of the mode values that it is willing to support during this session. Thus, the last three bits of the Modes 32-bit value are used. The first 29 bits MUST be zero. A client MUST ignore the values in the first 29 bits of the Modes value. (This way, the bits are available for future protocol extensions. This is the only intended extension mechanism.)

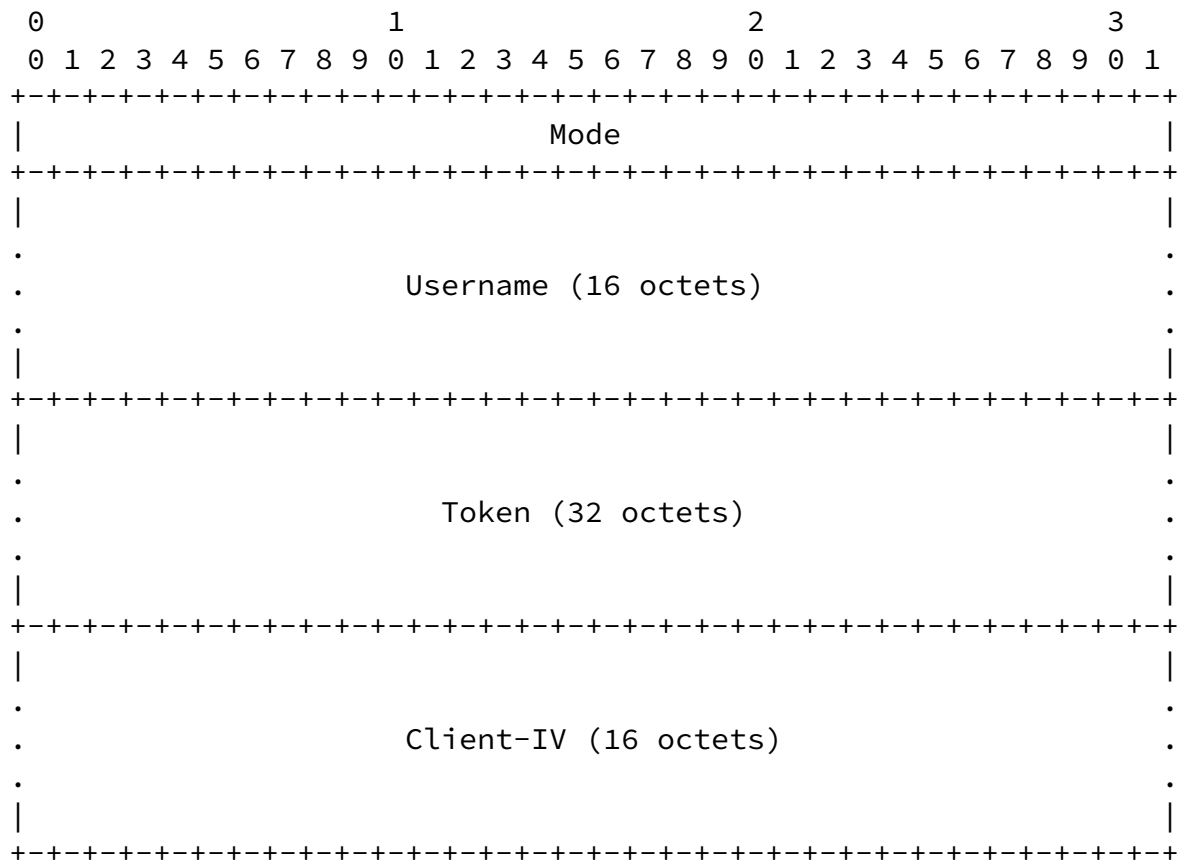
Challenge is a random sequence of octets generated by the server; it is used subsequently by the client to prove possession of a shared secret in a manner prescribed below.

If Modes value is zero, the server does not wish to communicate with the client and MAY close the connection immediately. The client

SHOULD close the connection if it receives a greeting with Modes

equal to zero. The client MAY close the connection if the client's desired mode is unavailable.

Otherwise, the client MUST respond with the following message:



Here Mode is the mode that the client chooses to use during this OWAMP-Control session. It will also be used for all OWAMP-Test sessions started under control of this OWAMP-Control session. In Mode, one or zero bits MUST be set within last three bits. The first 29 bits of Mode MUST be zero. A server MUST ignore the values of the first 29 bits.

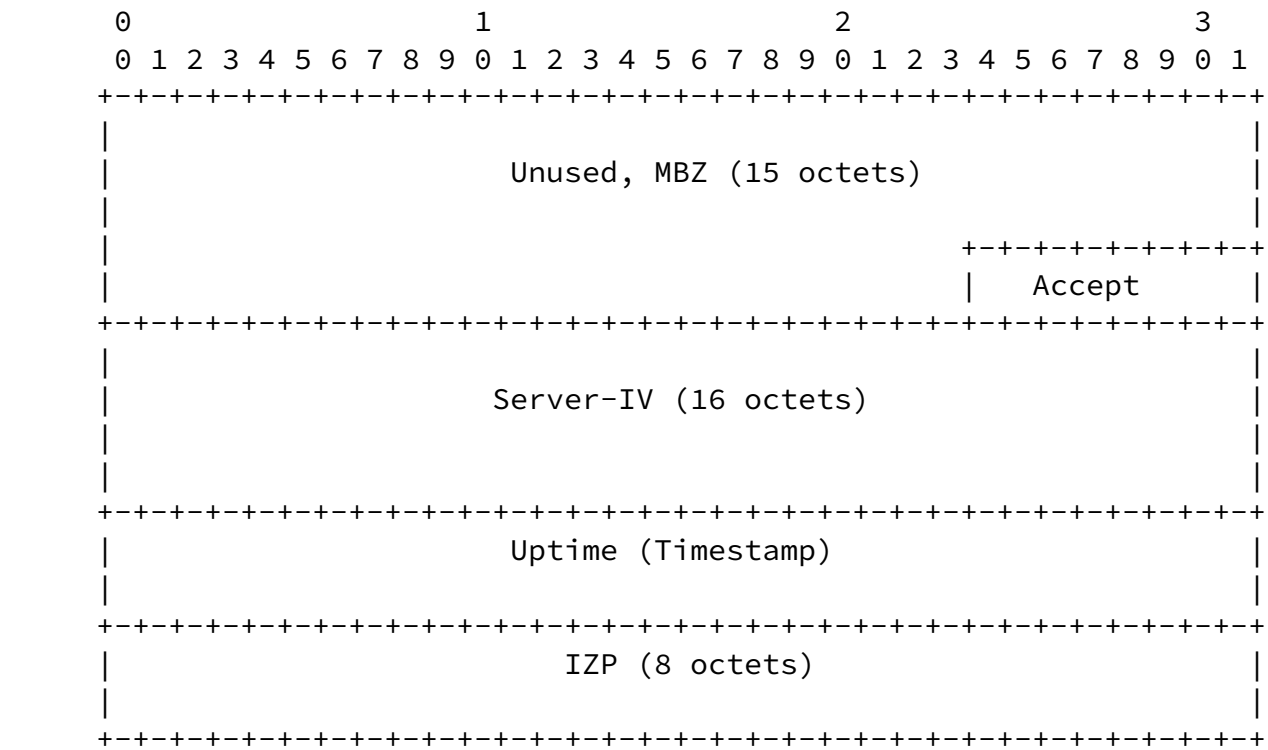
In unauthenticated mode, Username, Token, and Client-IV are unused.

Otherwise, Username is a 16-octet indicator that tells the server which shared secret the client wishes to use to authenticate or encrypt, while Token is the concatenation of a 16-octet challenge and a 16-octet Session-key, encrypted using the AES (Advanced Encryption Standard) [AES] in Cipher Block Chaining (CBC). Encryption MUST be performed using an Initialization Vector (IV) of zero and a key value that is the shared secret associated with Username. (Both the server and the client use the same mappings from user names to secret keys. The server, being prepared to conduct sessions with more than one

client, uses user names to choose the appropriate secret key; a client would typically have different secret keys for different servers. The situation is analogous to that of passwords, except that secret keys, rather than being the typical low-entropy passwords, are suitable for use as AES keys.) The shared secret will typically be provided as a passphrase; in this case, the MD5 sum [RFC1321] of the passphrase (without possible newline character(s) at the end of the passphrase) SHOULD be used as a key for encryption by the client and decryption by the server (the passphrase also SHOULD NOT contain newlines in the middle).

Session-key and Client-IV are generated randomly by the client. Session-key MUST be generated with sufficient entropy not to reduce the security of the underlying cipher. Client-IV merely needs to be unique (i.e., it MUST never be repeated for different sessions using the same secret key; a simple way to achieve that without the use of cumbersome state is to generate the Client-IV strings using a cryptographically secure pseudo-random number source: if this is done, the first repetition is unlikely to occur before 2^64 sessions with the same secret key are conducted).

The server MUST respond with the following message:



The Unused 15-octet part MUST be zero. The client MUST ignore its value. MBZ (MUST be zero) fields here and hereafter have the same

semantics: the party that sends the message MUST set the field to a string of zero bits; the party that interprets the message MUST

ignore the value. (This way the field could be used for future extensions.)

Server-IV is generated randomly by the server. In unauthenticated mode, Server-IV is unused.

The Accept field indicates the server's willingness to continue communication. A zero value in the Accept field means that the server accepts the authentication and is willing to conduct further transactions. Non-zero values indicate that the server does not accept the authentication or, for some other reason, is not willing to conduct further transactions in this OWAMP-Control session. The full list of available Accept values is described in the ``Values of the Accept Field'' section.

If a negative (non-zero) response is sent, the server MAY and the client SHOULD close the connection after this message.

Uptime is a timestamp representing the time when the current instantiation of the server started operating. (For example, in a multi-user general purpose operating system (OS), it could be the time when the server process was started.) If Accept is non-zero, Uptime SHOULD be set to a string of zeros. In authenticated and encrypted modes, Uptime is encrypted as described in the next section, unless Accept is non-zero. (Authenticated and encrypted mode cannot be entered unless the control connection can be initialized.)

Timestamp format is described in ``Sender Behavior'' section below. The same instantiation of the server SHOULD report the same exact Uptime value to each client in each session.

Integrity Zero Padding (IZP) is treated the same way as IZP in the next section and beyond.

The previous transactions constitute connection setup.

3.2. Values of the Accept Field

Accept values are used throughout the OWAMP-Control protocol to communicate the server response to client requests. The full set of valid Accept field values are:

0 OK.

1 Failure, reason unspecified (catch-all).

2 Internal error.

3 Some aspect of request is not supported.

4 Cannot perform request due to permanent resource limitations.

5 Cannot perform request due to temporary resource limitations.

All other values are reserved. The sender of the message MAY use the value of 1 for all non-zero Accept values. A message sender SHOULD use the correct Accept value if it is going to use other values. The message receiver MUST interpret all values of Accept other than these reserved values as 1. This way, other values are available for future extensions.

3.3. OWAMP-Control Commands

In authenticated or encrypted mode (which are identical as far as OWAMP-Control is concerned, and only differ in OWAMP-Test) all further communications are encrypted with the Session-key, using CBC mode. The client encrypts its stream using Client-IV. The server encrypts its stream using Server-IV.

The following commands are available for the client: Request-Session, Start-Sessions, Stop-Sessions, and Fetch-Session. The command Stop-Sessions is available to both the client and the server. (The server can also send other messages in response to commands it receives.)

After Start-Sessions is sent/received by the client/server, and before it both sends and receives Stop-Sessions (order unspecified), it is said to be conducting active measurements.

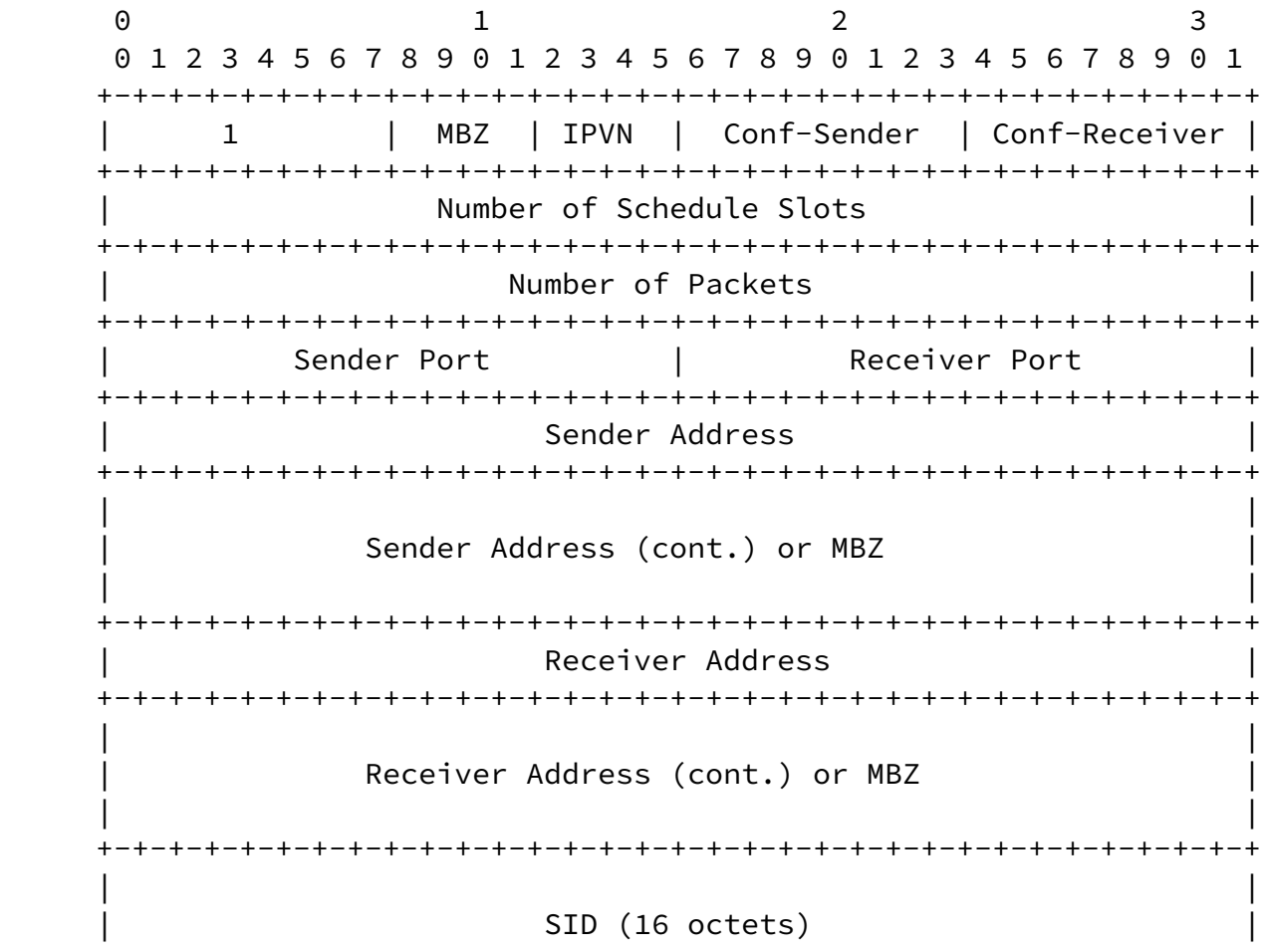
While conducting active measurements, the only command available is Stop-Sessions.

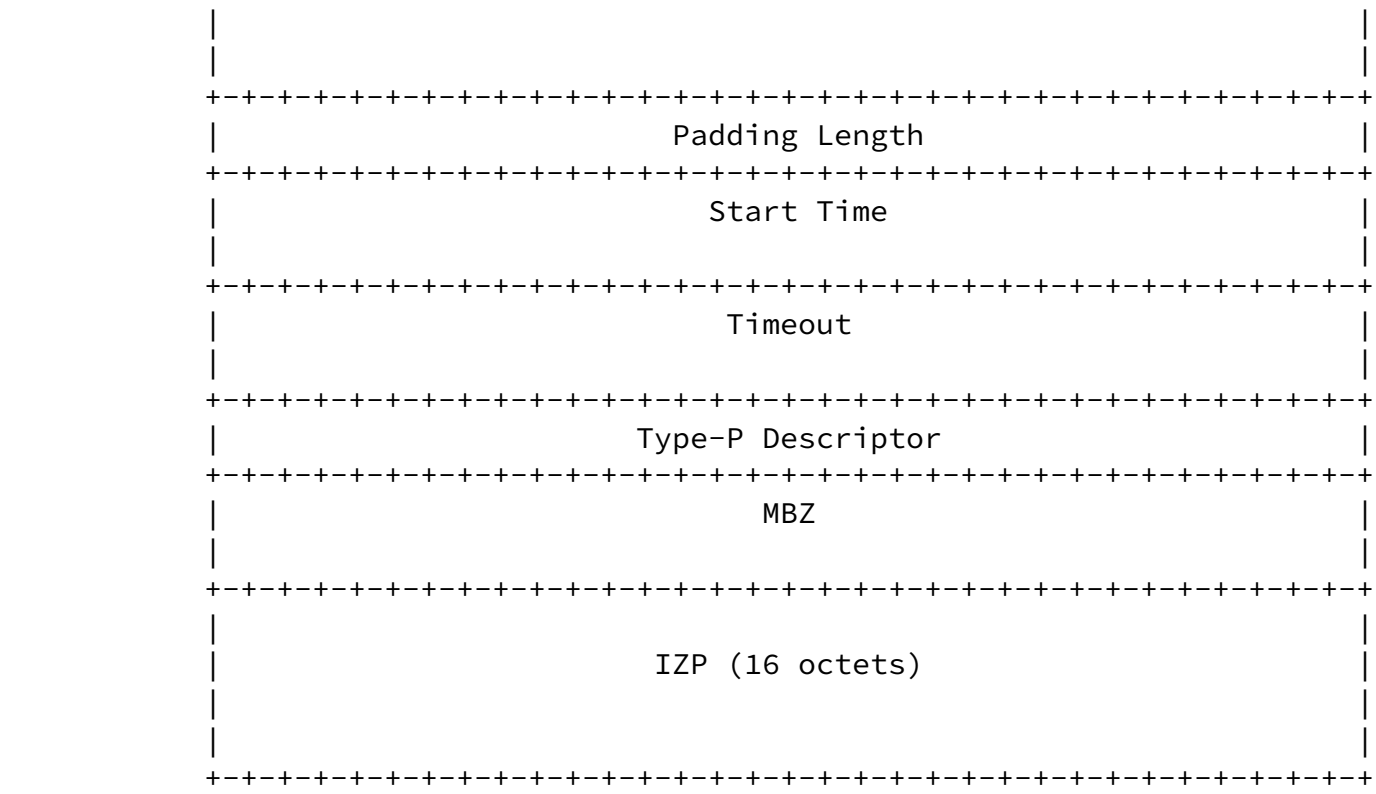
These commands are described in detail below.

3.4. Creating Test Sessions

Individual one-way active measurement sessions are established using a simple request/response protocol. An OWAMP client MAY issue zero or more Request-Session messages to an OWAMP server, which MUST respond to each with an Accept-Session message. An Accept-Session message MAY refuse a request.

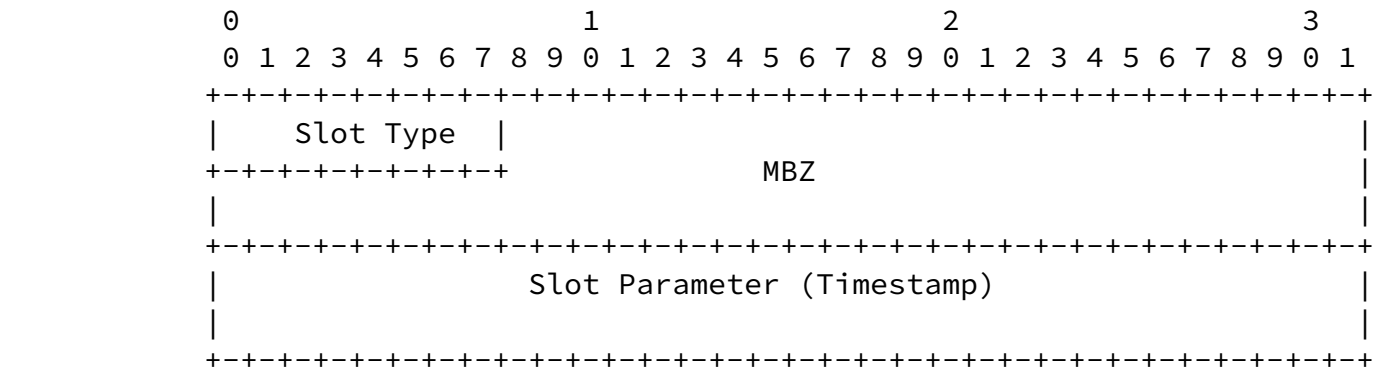
The format of Request-Session message is as follows:



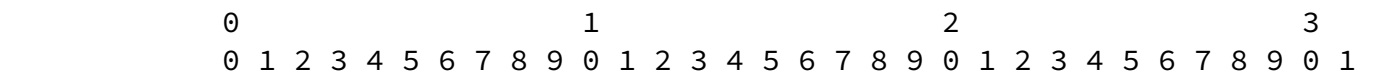


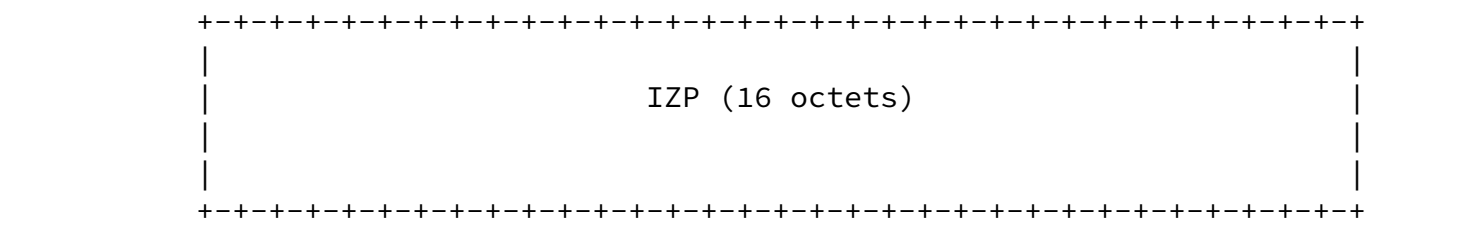
This is immediately followed by one or more schedule slot

descriptions (the number of schedule slots is specified in the `Number of Schedule Slots' field above):



These are immediately followed by IZP:





All these messages comprise one logical message: the Request-Session command.

Above, the first octet (1) indicates that this is Request-Session command.

IPVN is the IP version numbers for Sender and Receiver. When the IP version number is 4, 12 octets follow the 4-octet IPv4 address stored in Sender Address and Receiver Address. These octets MUST be set to zero by the client and MUST be ignored by the server. Currently meaningful IPVN values are 4 and 6.

Conf-Sender and Conf-Receiver MUST be set to 0 or 1 by the client. The server MUST interpret any non-zero value as 1. If the value is 1, the server is being asked to configure the corresponding agent (sender or receiver). In this case, the corresponding Port value SHOULD be disregarded by the server. At least one of Conf-Sender and Conf-Receiver MUST be 1. (Both can be set, in which case the server is being asked to perform a session between two hosts it can configure.)

Number of Schedule Slots, as mentioned before, specifies the number of slot records that go between the two blocks of IZP. It is used by

the sender to determine when to send test packets (see next section).

Number of Packets is the number of active measurement packets to be sent during this OWAMP-Test session (note that both server and client can abort the session early).

If Conf-Sender is not set, Sender Port is the UDP port from which OWAMP-Test packets will be sent. If Conf-Receiver is not set, Receiver Port is the UDP port OWAMP-Test to which packets are requested to be sent.

The Sender Address and Receiver Address fields contain, respectively, the sender and receiver addresses of the end points of the Internet path over which an OWAMP test session is requested.

SID is the session identifier. It can be used in later sessions as an argument for the Fetch-Session command. It is meaningful only if Conf-Receiver is 0. This way, the SID is always generated by the receiving side. See the end of the section for information on how the SID is generated.

Padding length is the number of octets to be appended to the normal OWAMP-Test packet (see more on padding in discussion of OWAMP-Test).

Start Time is the time when the session is to be started (but not before Start-Sessions command is issued). This timestamp is in the same format as OWAMP-Test timestamps.

Timeout (or a loss threshold) is an interval of time (expressed as a timestamp). A packet belonging to the test session that is being set up by the current Request-Session command will be considered lost if it is not received during Timeout seconds after it is sent.

Type-P Descriptor covers only a subset of (very large) Type-P space. If the first two bits of the Type-P Descriptor are 00, then subsequent six bits specify the requested Differentiated Services Codepoint (DSCP) value of sent OWAMP-Test packets, as defined in RFC 2474. If the first two bits of Type-P descriptor are 01, then the subsequent 16 bits specify the requested PHB Identification Code (PHB ID), as defined in RFC 2836.

Therefore, the value of all zeros specifies the default best-effort service.

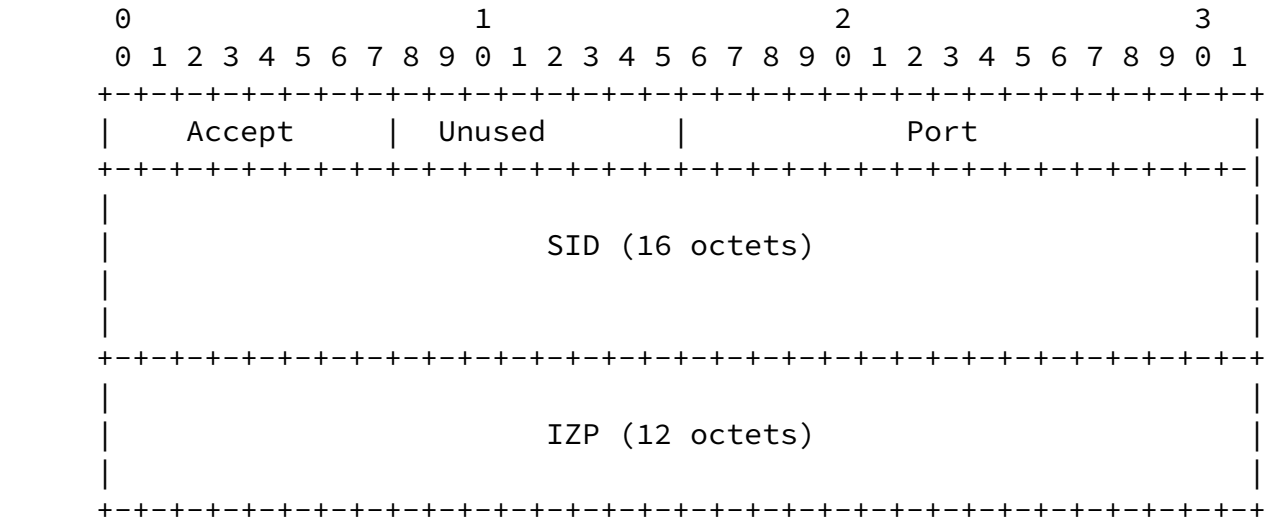
If Conf-Sender is set, the Type-P Descriptor is to be used to configure the sender to send packets according to its value. If Conf-Sender is not set, the Type-P Descriptor is a declaration of how the sender will be configured.

If Conf-Sender is set and the server does not recognize the Type-P Descriptor, or it cannot or does not wish to set the corresponding attributes on OWAMP-Test packets, it SHOULD reject the session request. If Conf-Sender is not set, the server SHOULD accept or

reject the session paying no attention to the value of the Type-P Descriptor.

IZP MUST be all zeros in this and all messages that use IZP. The recipient of a message where IZP is not zero MUST reject the message, as it is an indication of tampering with the content of the message by an intermediary (or brokenness). If the message is part of OWAMP-Control, the session MUST be terminated and results invalidated. If the message is part of OWAMP-Test, it MUST be silently ignored. This will ensure data integrity. In unauthenticated mode, IZP is nothing more than a simple check. In authenticated and encrypted modes, however, it ensures, in conjunction with properties of CBC chaining mode, that everything received before was not tampered with. For this reason, it is important to check the IZP field as soon as possible, so that bad data doesn't get propagated.

To each Request-Session message, an OWAMP server MUST respond with an Accept-Session message:



In this message, zero in the Accept field means that the server is willing to conduct the session. A non-zero value indicates rejection of the request. The full list of available Accept values is described in the ``Values of the Accept Field'' section.

If the server rejects a Request-Session message, it SHOULD not close the TCP connection. The client MAY close it if it receives negative response to the Request-Session message.

The meaning of Port in the response depends on the values of Conf-Sender and Conf-Receiver in the query that solicited the response. If both were set, the Port field is unused. If only Conf-Sender was set, Port is the port from which to expect OWAMP-Test packets. If only Conf-Receiver was set, Port is the port to which OWAMP-Test packets are sent.

If only Conf-Sender was set, the SID field in the response is unused. Otherwise, SID is a unique server-generated session identifier. It can be used later as handle to fetch the results of a session.

SIDs SHOULD be constructed by concatenation of the 4-octet IPv4 IP number belonging to the generating machine, an 8-octet timestamp, and a 4-octet random value. To reduce the probability of collisions, if the generating machine has any IPv4 addresses (with the exception of loopback), one of them SHOULD be used for SID generation, even if all communication is IPv6-based. If it has no IPv4 addresses at all, the last four octets of an IPv6 address MAY be used instead. Note that SID is always chosen by the receiver. If truly random values are not available, it is important that the SID be made unpredictable, as knowledge of the SID might be used for access control.

3.5. Send Schedules

The sender and the receiver both need to know the same send schedule. This way, when packets are lost, the receiver knows when they were supposed to be sent. It is desirable to compress common schedules and still to be able to use an arbitrary one for the test sessions. In many cases, the schedule will consist of repeated sequences of packets: this way, the sequence performs some test, and the test is repeated a number of times to gather statistics.

To implement this, we have a schedule with a given number of slots. Each slot has a type and a parameter. Two types are supported: exponentially distributed pseudo-random quantity (denoted by a code of 0) and a fixed quantity (denoted by a code of 1). The parameter is expressed as a timestamp and specifies a time interval. For a type 0 slot (exponentially distributed pseudo-random quantity) this interval is the mean value (or 1/lambda if the distribution density function is expressed as lambda*exp(-lambda*x) for positive values of x). For a type 1 (fixed quantity) slot, the parameter is the delay itself. The sender starts with the beginning of the schedule, and executes the instructions in the slots: for a slot of type 0, wait an exponentially distributed time with a mean of the specified parameter and then send a test packet (and proceed to the next slot); for a slot of type 1, wait the specified time and send a test packet (and

proceed to the next slot). The schedule is circular: when there are

no more slots, the sender returns to the first slot.

The sender and the receiver need to be able to reproducibly execute the entire schedule (so, if a packet is lost, the receiver can still attach a send timestamp to it). Slots of type 1 are trivial to reproducibly execute. To reproducibly execute slots of type 0, we need to be able to generate pseudo-random exponentially distributed quantities in a reproducible manner. The way this is accomplished is discussed later.

Using this mechanism one can easily specify common testing scenarios. Some examples include:

+ Poisson stream: a single slot of type 0;

+ Periodic stream: a single slot of type 1;

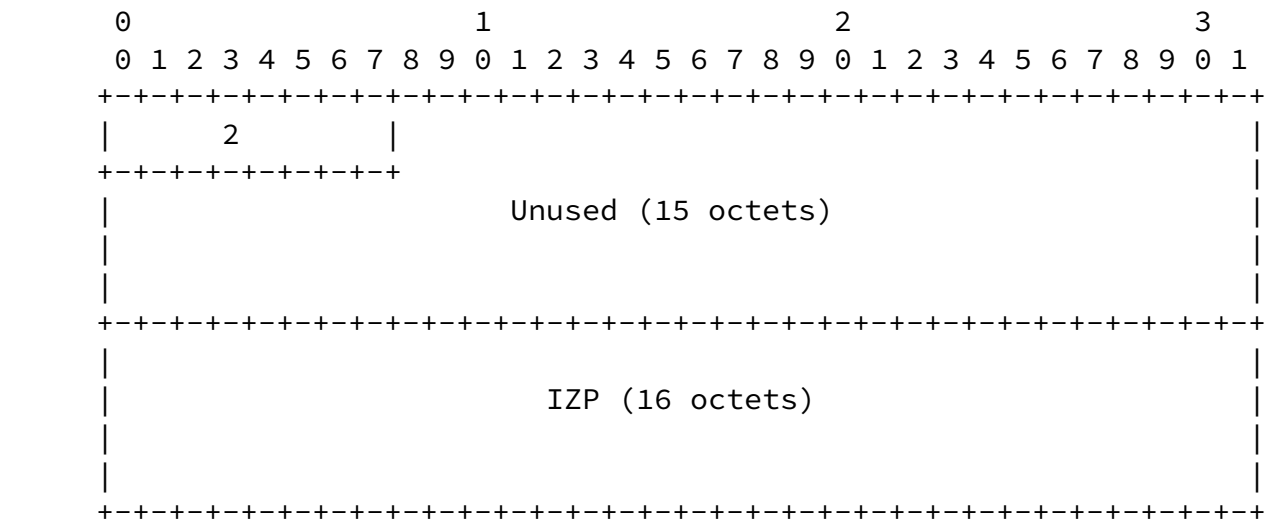
+ Poisson stream of back-to-back packet pairs: two slots -- type 0 with a non-zero parameter and type 1 with a zero parameter.

Further, a completely arbitrary schedule can be specified (albeit inefficiently) by making the number of test packets equal to the number of schedule slots. In this case, the complete schedule is transmitted in advance of an OWAMP-Test session.

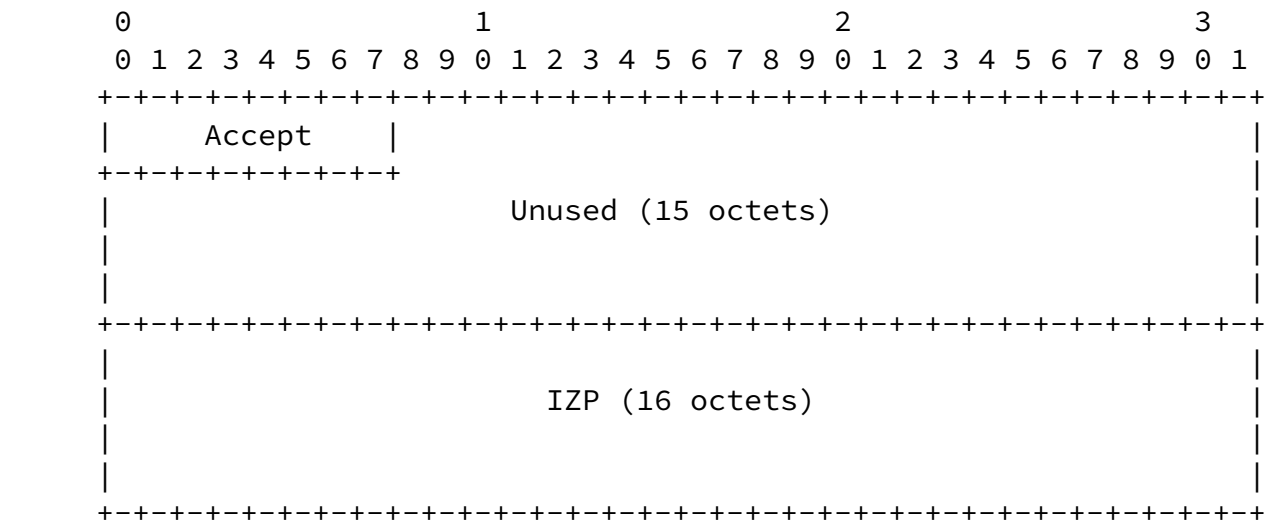
3.6. Starting Test Sessions

Having requested one or more test sessions and received affirmative Accept-Session responses, an OWAMP client MAY start the execution of the requested test sessions by sending a Start-Sessions message to the server.

The format of this message is as follows:



The server MUST respond with an Start-Ack message (which SHOULD be sent as quickly as possible). Start-Ack messages have the following format:

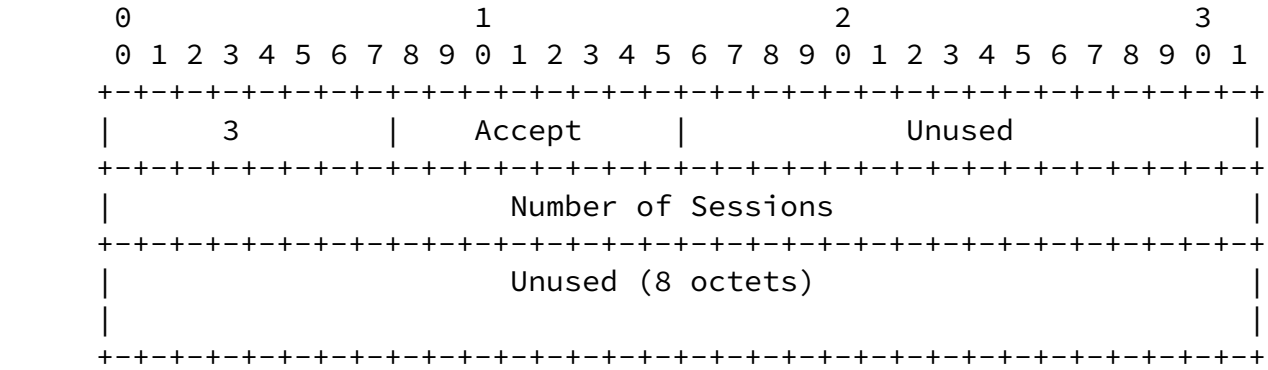


If Accept is non-zero, the Start-Sessions request was rejected; zero means that the command was accepted. The full list of available Accept values is described in the ``Values of the Accept Field'' section. The server MAY, and the client SHOULD, close the connection in the case of a rejection.

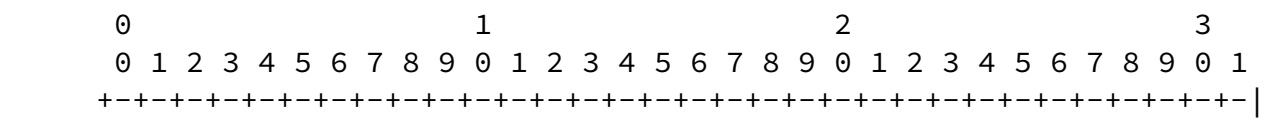
The server SHOULD start all OWAMP-Test streams immediately after it sends the response or immediately after their specified start times, whichever is later. If the client represents a Sender, the client SHOULD start its OWAMP-Test streams immediately after it sees the Start-Ack response from the Server (if the Start-Sessions command was accepted) or immediately after their specified start times, whichever is later. See more on OWAMP-Test sender behavior in a separate section below.

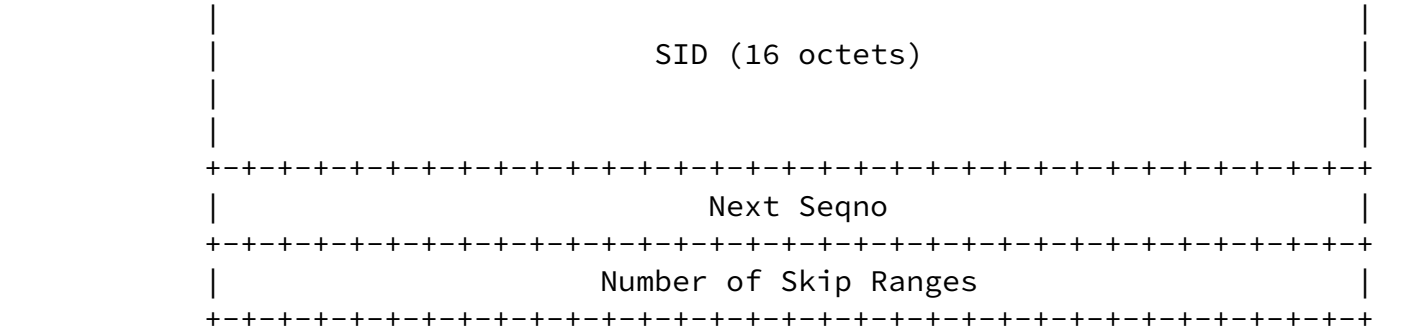
3.7. Stop-Sessions

The Stop-Sessions message may be issued by either the Control-Client or the Server. The format of this command is as follows:

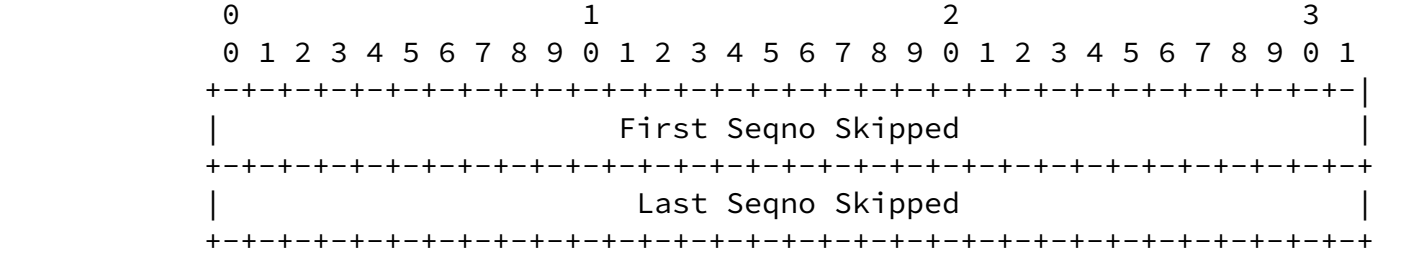


This is immediately followed by zero or more session description records (the number of session description records is specified in the ``Number of Sessions'' field above). The session description record is used to indicate which packets were actually sent by the sender process (rather than skipped). The header of the session description record is as follows:



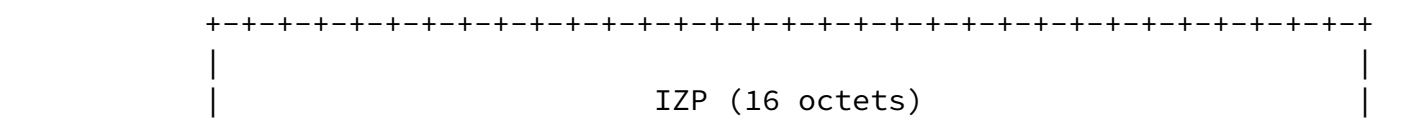


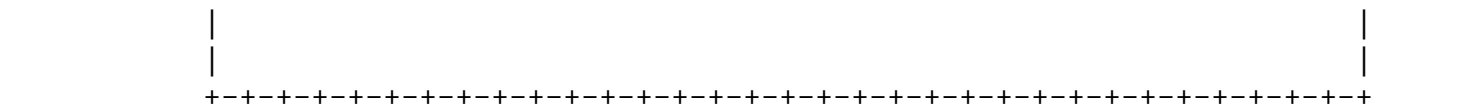
This is immediately followed by zero or more Skip Range descriptions as specified by the ``Number of Skip Ranges'' field above. Skip Ranges are simply two sequence numbers that, together, indicate a range of packets that were not sent:



Skip Ranges MUST be in order. The last (possibly full, possibly incomplete) block (16 octets) of data MUST be padded with zeros, if necessary. This ensures that the next session description record starts on a block boundary.

Finally, a single block (16 octets) of IZP is concatenated on the end to complete the Stop-Sessions message.





All these records comprise one logical message: the Stop-Sessions command.

Above, the first octet (3) indicates that this is the Stop-Sessions command.

Non-zero Accept values indicate a failure of some sort. Zero values indicate normal (but possibly premature) completion. The full list of available Accept values is described in the ``Values of the Accept Field'' section.

If Accept had a non-zero value (from either party), results of all OWAMP-Test sessions spawned by this OWAMP-Control session SHOULD be considered invalid, even if a Fetch-Session with SID from this session works for a different OWAMP-Control session. If Accept was not transmitted at all (for whatever reason, including the TCP connection used for OWAMP-Control breaking), the results of all OWAMP-Test sessions spawned by this OWAMP-control session MAY be considered invalid.

Number of Sessions indicates the number of session description records that immediately follow the Stop-Sessions header.

Number of Sessions MUST contain the number of send sessions started by the local side of the control connection that have not been

previously terminated by a Stop-Sessions command (i.e., the Control-Client MUST account for each accepted Request-Session where Conf-Receiver was set; the Control-Server MUST account for each accepted Request-Session where Conf-Sender was set). If the Stop-Sessions message does not account for exactly the send sessions controlled by that side, then it is to be considered invalid and the connection SHOULD be closed and any results obtained considered invalid.

Each session description record represents one OWAMP-Test session.

SID is the session identifier (SID) used to indicate which send

session is being described.

Next Seqno indicates the next sequence number that would have been sent from this send session. For completed sessions, this will equal NumPackets from the Request-Session.

Number of Skip Ranges indicates the number of holes that actually occurred in the sending process. This is a range of packets that were never actually sent by the sending process. For example, if a send session is started too late for the first 10 packets to be sent and this is the only hole in the schedule, then ``Number of Skip Ranges'' would be 1. The single Skip Range description will have First Seqno Skipped equal to 0 and Last Seqno Skipped equal to 9. This is described further in the ``Sender Behavior'' section.

If the OWAMP-Control connection breaks when the Stop-Sessions command is sent, the receiver MAY not completely invalidate the session results. It MUST discard all record of packets that follow (in other words, have greater sequence number than) the last packet that was actually received before before any lost packet records. This will help differentiate between packet losses that occurred in the network and packets the sending process may have never sent.

If a receiver of an OWAMP-Test session learns, through an OWAMP-Control Stop-Sessions message, that the OWAMP-Test sender's last sequence number is lower than any sequence number actually received, the results of the complete OWAMP-Test session MUST be invalidated.

A receiver of an OWAMP-Test session, upon receipt of an OWAMP-Control Stop-Sessions command, MUST discard any packet records -- including lost packet records -- with a (computed) send time that falls between the current time minus Timeout and the current time. This ensures statistical consistency for the measurement of loss and duplicates in the event that the Timeout is greater than the time it takes for the Stop-Sessions command to take place.

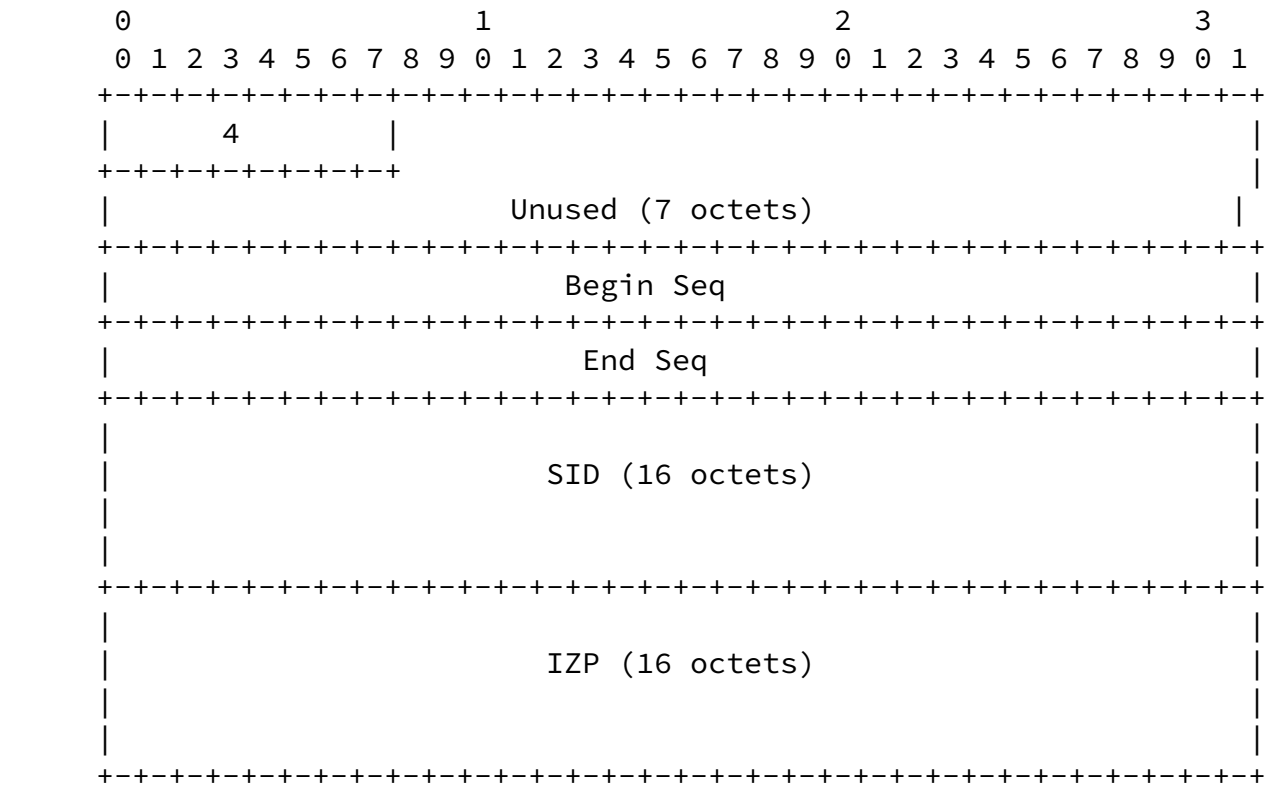
To effect complete sessions, each side of the control connection SHOULD wait until all sessions are complete before sending the Stop-Sessions message. The completed time of each sessions is determined as Timeout after the scheduled time for the last sequence number. Endpoints MAY add a small increment to the computed

completed time for send endpoints to ensure the Stop-Sessions message reaches the receiver endpoint after Timeout.

To effect a premature stop of sessions, the party that initiates this command MUST stop its OWAMP-Test send streams to send the Session Packets Sent values before sending this command. That party SHOULD wait until receiving the response Stop-Sessions message before stopping the receiver streams so that it can use the values from the received Stop-Sessions message to validate the data.

3.8. Fetch-Session

The format of this client command is as follows:

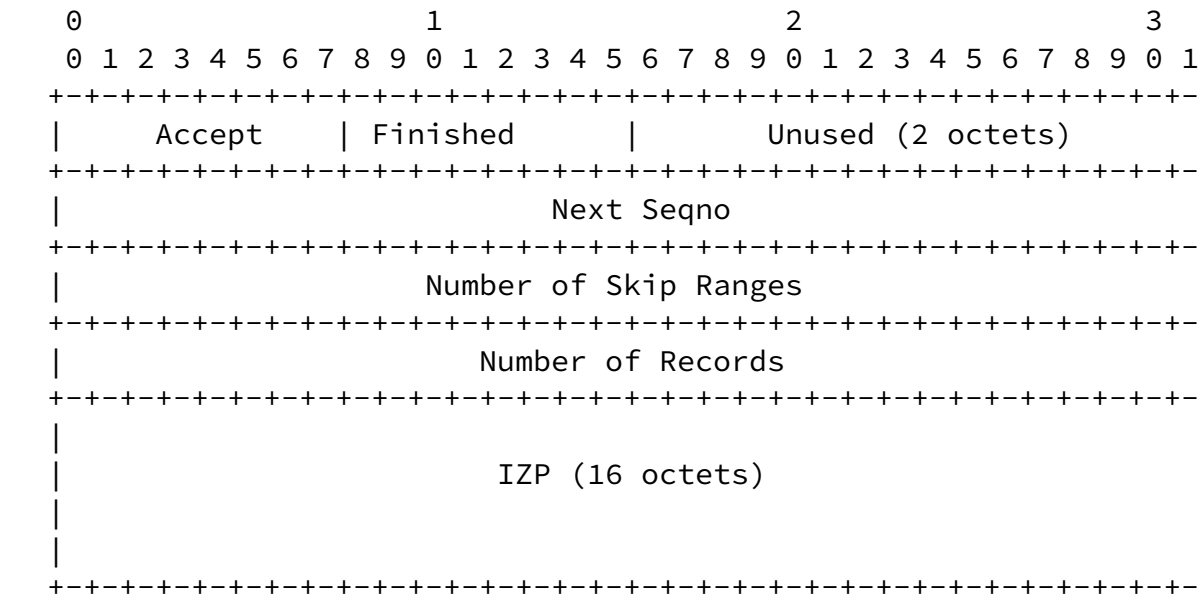


Begin Seq is the sequence number of the first requested packet. End Seq is the sequence number of the last requested packet. If Begin Seq is all zeros and End Seq is all ones, complete session is said to be requested.

If a complete session is requested and the session is still in

progress, or has terminated in any way other than normal, the request to fetch session results MUST be denied. If an incomplete session is requested, all packets received so far that fall into the requested range SHOULD be returned. Note that, since no commands can be issued between Start-Sessions and Stop-Sessions, incomplete requests can only happen on a different OWAMP-Control connection (from the same or different host as Control-Client).

The server MUST respond with a Fetch-Ack message. The format of this server response is as follows:



Again, non-zero in the Accept field means a rejection of command. The server MUST specify zero for all remaining fields if Accept is non-zero. The client MUST ignore all remaining fields (except for the IZP) if Accept is non-zero. The full list of available Accept values is described in the ``Values of the Accept Field'' section.

Finished is non-zero if the OWAMP-Test session has terminated.

Next Seqno indicates the next sequence number that would have been sent from this send session. For completed sessions, this will equal NumPackets from the Request-Session. This information is only available if the session has terminated. If Finished is zero, then Next Seqno MUST be set to zero by the server.

Number of Skip Ranges indicates the number of holes that actually occurred in the sending process. This information is only available if the session has terminated. If Finished is zero, then Skip Ranges MUST be set to zero by the server.

Number of Records is the number of packet records that fall within

the requested range. This number might be less than the Number of Packets in the reproduction of the Request-Session command because of a session that ended prematurely or it might be greater because of duplicates.

If Accept was non-zero, this concludes the response to the Fetch-Session message. If Accept was 0, the server then MUST immediately send the OWAMP-Test session data in question.

The OWAMP-Test session data consists of the following (concatenated):

+ A reproduction of the Request-Session command that was used to start the session; it is modified so that actual sender and receiver port numbers that were used by the OWAMP-Test session always appear in the reproduction.

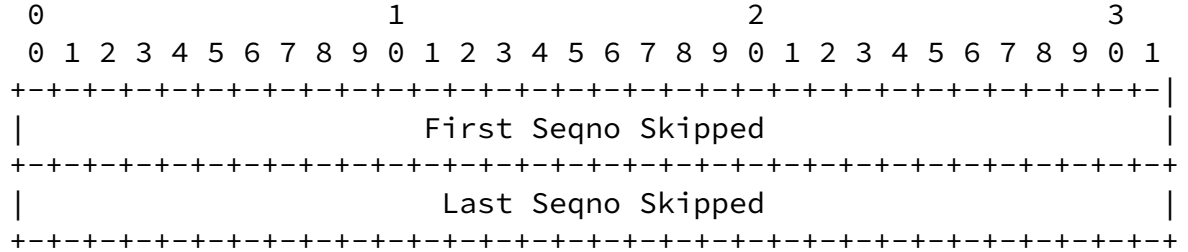
+ Zero or more (as specified) Skip Range descriptions. The last (possibly full, possibly incomplete) block (16 octets) of Skip Range descriptions is padded with zeros if necessary. (These zeros are simple padding and should be distinguished from the 16 octets of IZP that follow.)

+ 16 octets of IZP.

+ Zero or more (as specified) packet records. The last (possibly full, possibly incomplete) block (16 octets) of data is padded with zeros if necessary. (These zeros are simple padding and should be distinguished from the 16 octets of IZP that follow.)

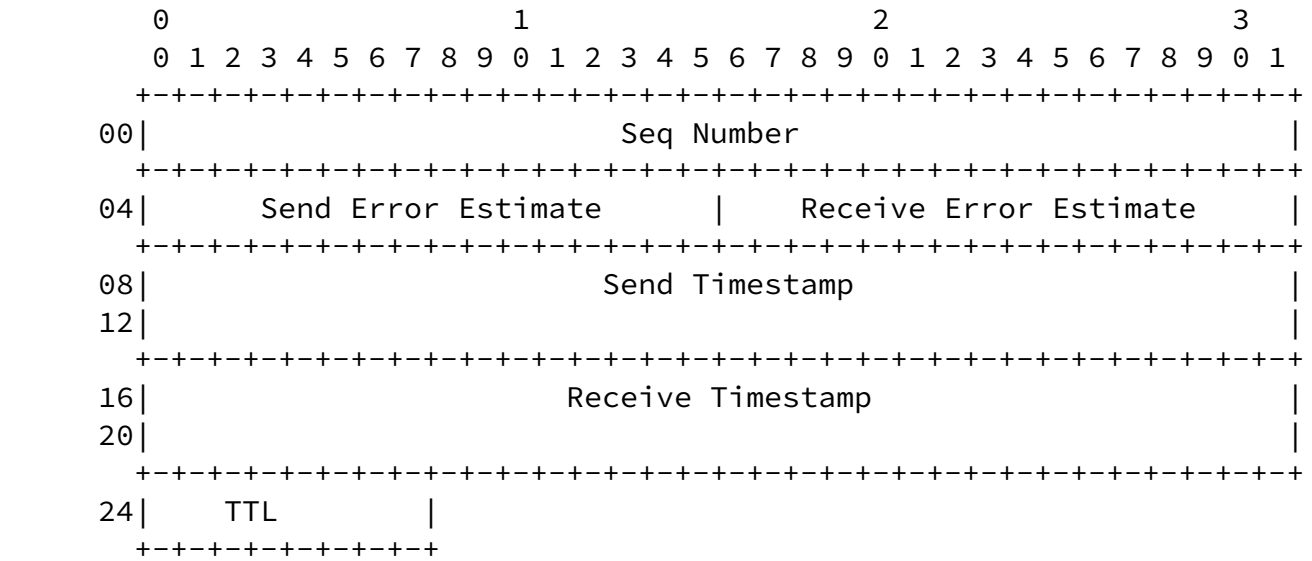
+ 16 octets of IZP.

Skip Range descriptions are simply two sequence numbers that, together, indicate a range of packets that were not sent:



Skip Range descriptions should be sent out in order, as sorted by First Seqno. If any Skip Ranges overlap, or are out of order, the session data is to be considered invalid and the connection SHOULD be closed and any results obtained considered invalid.

Each packet record is 25 octets, and includes 4 octets of sequence number, 8 octets of send timestamp, 2 octets of send timestamp error estimate, 8 octets of receive timestamp, 2 octets of receive timestamp error estimate, and 1 octet of Time To Live (TTL), or Hop Limit in IPv6:



Packet records are sent out in the same order the actual packets were received. Therefore, the data is in arrival order.

Note that lost packets (if any losses were detected during the OWAMP-Test session) MUST appear in the sequence of packets. They can appear either at the point when the loss was detected or at any later point. Lost packet records are distinguished as follows:

+ A send timestamp filled with the presumed send time (as computed by the send schedule).

+ A send error estimate filled with Multiplier=1, Scale=64, and S=0

(see the OWAMP-Test description for definition of these quantities and explanation of timestamp format and error estimate format).

+ A normal receive error estimate as determined by the error of the clock being used to declare the packet lost. (It is declared lost if it is not received by the Timeout after the presumed send time, as determined by the receiver's clock.)

+ A receive timestamp consisting of all zero bits.

+ A TTL value of 255.

4. OWAMP-Test

This section describes OWAMP-Test protocol. It runs over UDP using sender and receiver IP and port numbers negotiated during the Request-Session exchange.

As with OWAMP-Control, OWAMP-Test has three modes: unauthenticated, authenticated, and encrypted. All OWAMP-Test sessions that are spawned by an OWAMP-Control session inherit its mode.

OWAMP-Control client, OWAMP-Control server, OWAMP-Test sender, and OWAMP-Test receiver can potentially all be different machines. (In a typical case, we expect that there will be only two machines.)

4.1. Sender Behavior

4.1.1. Packet Timings

Send schedules based on slots, described previously, in conjunction with scheduled session start time, enable the sender and the receiver to compute the same exact packet sending schedule independently of each other. These sending schedules are independent for different OWAMP-Test sessions, even if they are governed by the same OWAMP-Control session.

Consider any OWAMP-Test session. Once Start-Sessions exchange is

complete, the sender is ready to start sending packets. Under normal OWAMP use circumstances, the time to send the first packet is in the near future (perhaps a fraction of a second away). The sender SHOULD send packets as close as possible to their scheduled time, with the following exception: if the scheduled time to send is in the past, and separated from the present by more than Timeout time, the sender MUST NOT send the packet. (Indeed, such a packet would be considered lost by the receiver anyway.) The sender MUST keep track of which packets it does not send. It will use this to tell the receiver what packets were not sent by setting Skip Ranges in the Stop-Sessions message from the sender to the receiver upon completion of the test. The Skip Ranges are also sent to a Fetch-Client as part of the session data results. These holes in the sending schedule can happen if a time in the past was specified in the Request-Session command, or if the Start-Sessions exchange took unexpectedly long, or if the sender could not start serving the OWAMP-Test session on time due to internal scheduling problems of the OS. Packets in the past, but separated from the present by less than Timeout value, SHOULD be sent as quickly as possible. With normal test rates and timeout values, the number of packets in such a burst is limited. Nevertheless,

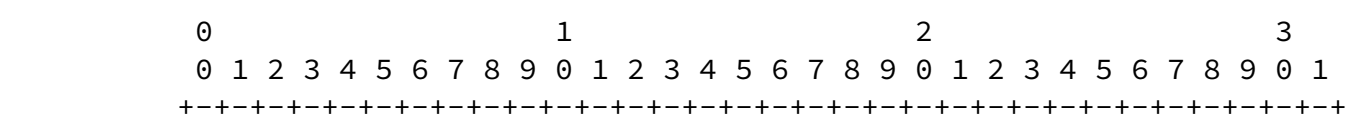
hosts SHOULD NOT intentionally schedule sessions so that such bursts of packets occur.

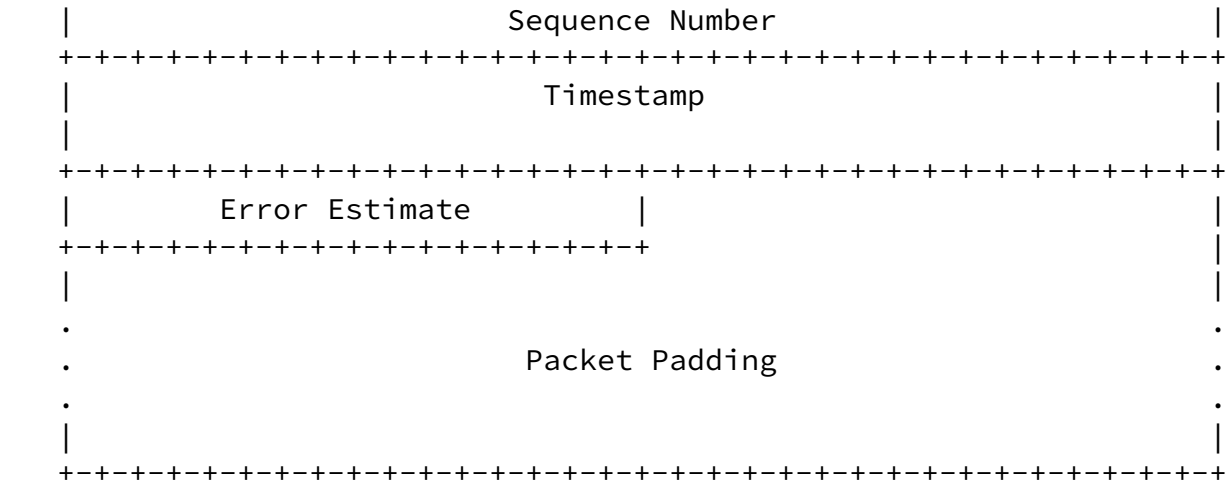
Regardless of any scheduling delays, each packet that is actually sent MUST have the best possible approximation of its real time of departure as its timestamp (in the packet).

4.1.2. Packet Format and Content

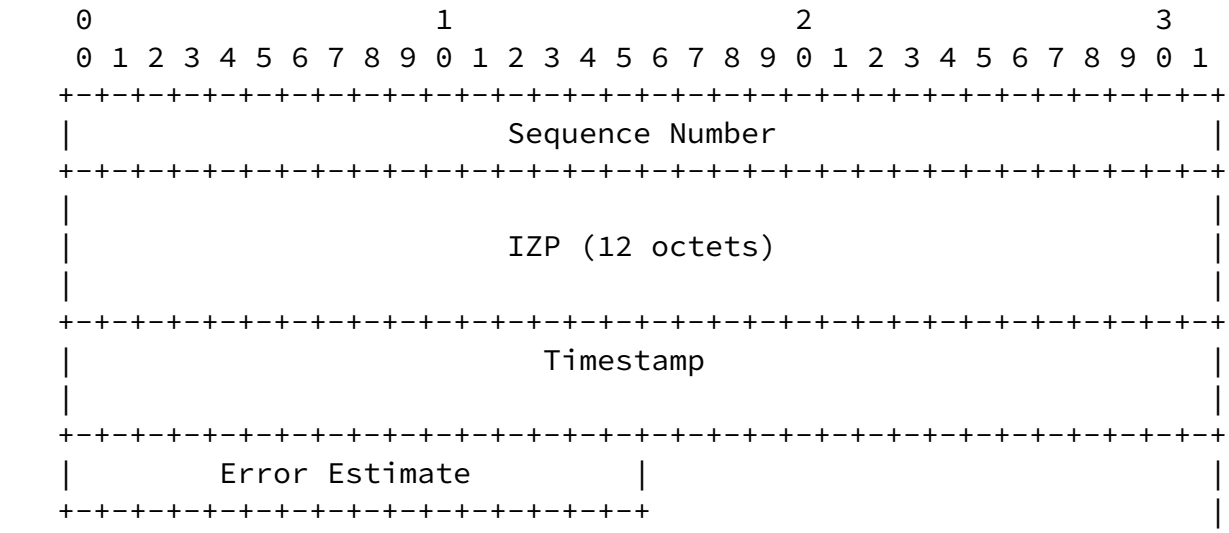
The sender sends the receiver a stream of packets with the schedule specified in the Request-Session command. The sender SHOULD set the TTL in IPv4 (or Hop Limit in IPv6) in the UDP packet to 255. The format of the body of a UDP packet in the stream depends on the mode being used.

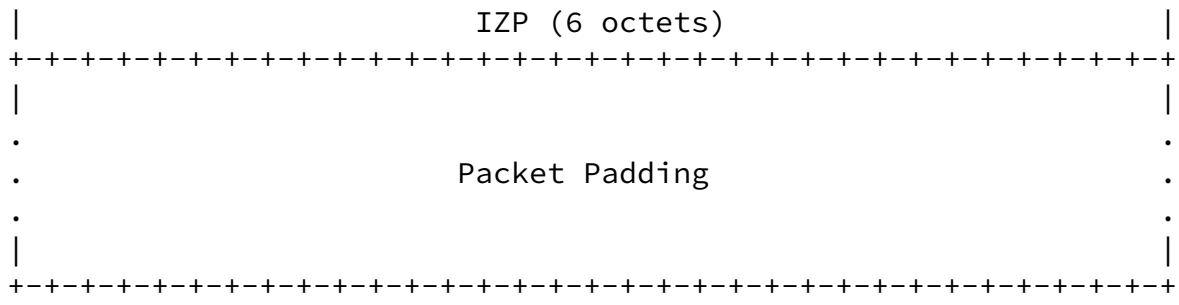
For unauthenticated mode:





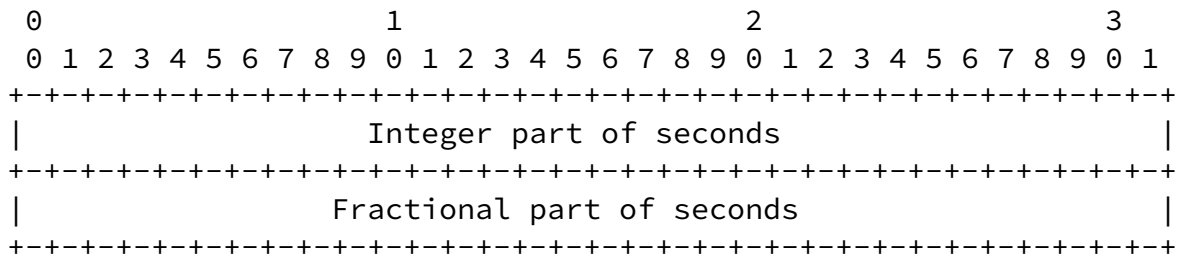
For authenticated and encrypted modes:



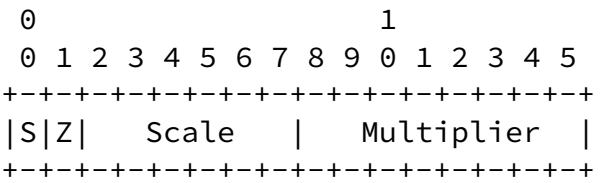


The format of the timestamp is the same as in [RFC 1305] and is as follows: first 32 bits represent the unsigned integer number of seconds elapsed since 0h on 1 January 1900; next 32 bits represent the fractional part of a second that has elapsed since then.

So, Timestamp is represented as follows:



The Error Estimate specifies the estimate of the error and synchronization. It has the following format:



The first bit S SHOULD be set if the party generating the timestamp has a clock that is synchronized to UTC using an external source

(e.g., the bit should be set if GPS hardware is used and it indicates that it has acquired current position and time or if NTP is used and it indicates that it has synchronized to an external source, which includes stratum 0 source, etc.); if there is no notion of external synchronization for the time source, the bit SHOULD NOT be set. The next bit has the same semantics as MBZ fields elsewhere: it MUST be set to zero by the sender and ignored by everyone else. The next six

bits, Scale, form an unsigned integer; Multiplier is an unsigned integer as well. They are interpreted as follows: the error estimate is equal to Multiplier*2^(-32)*2^Scale (in seconds). [Notation clarification: 2^Scale is two to the power of Scale.] Multiplier MUST NOT be set to zero. If Multiplier is zero, the packet SHOULD be considered corrupt and discarded.

Sequence numbers start with zero and are incremented by one for each subsequent packet.

The minimum data segment length is, therefore, 14 octets in unauthenticated mode, and 32 octets in both authenticated mode and encrypted modes.

The OWAMP-Test packet layout is the same in authenticated and encrypted modes. The encryption operations are, however, different. The difference is that in encrypted mode both the sequence number and the timestamp are encrypted to provide maximum data integrity protection while in authenticated mode the sequence number is encrypted and the timestamp is sent in clear text. Sending the timestamp in clear text in authenticated mode allows one to reduce the time between when a timestamp is obtained by a sender and when the packet is shipped out. In encrypted mode, the sender has to fetch the timestamp, encrypt it, and send it; in authenticated mode, the middle step is removed, potentially improving accuracy (the sequence number can be encrypted before the timestamp is fetched).

In authenticated mode, the first block (16 octets) of each packet is encrypted using AES Electronic Cookbook (ECB) mode.

The key to use is obtained as follows: the 16-octet session identifier (SID) is encrypted with the same session key as is used for the corresponding OWAMP-Control session (where it is used in a different chaining mode); this is a single-block ECB encryption; its result is the key to use in encrypting (and decrypting) the packets of the particular OWAMP-Test session.

ECB mode used for encrypting the first block of OWAMP-Test packets in authenticated mode does not involve any actual chaining; this way, lost, duplicated, or reordered packets do not cause problems with deciphering any packet in an OWAMP-Test session.

In encrypted mode, the first two blocks (32 octets) are encrypted using AES CBC mode. The key to use is obtained in the same way as the key for authenticated mode. Each OWAMP-Test packet is encrypted as a separate stream, with just one chaining operation; chaining does not span multiple packets so that lost, duplicated, or reordered packets do not cause problems. The initialization vector for the CBC encryption is a string of zeros.

Implementation note: Naturally, the key schedule for each OWAMP-Test session need only be set up once per session, not once per packet.

In unauthenticated mode, no encryption is applied.

Packet Padding in OWAMP-Test SHOULD be pseudo-random (it MUST be generated independently of any other pseudo-random numbers mentioned in this document). However, implementations MUST provide a configuration parameter, an option, or a different means of making Packet Padding consist of all zeros.

The time elapsed between packets is computed according to the slot schedule as mentioned in Request-Session command description. At that point, we skipped over the issue of computing exponentially distributed pseudo-random numbers in a reproducible fashion. It is discussed later in a separate section.

4.2. Receiver Behavior

The receiver knows when the sender will send packets. The following parameter is defined: Timeout (from Request-Session). Packets that are delayed by more than Timeout are considered lost (or `as good as lost'). Note that there is never an actual assurance of loss by the network: a `lost' packet might still be delivered at any time. The original specification for IPv4 required that packets be delivered within TTL seconds or never (with TTL having a maximum value of 255). To the best of the authors' knowledge, this requirement was never actually implemented (and, of course, only a complete and universal implementation would ensure that packets do not travel for longer than TTL seconds). In fact, in IPv6, the name of this field has actually been changed to Hop Limit. Further, IPv4 specification makes no claims about the time it takes the packet to traverse the last link of the path.

The choice of a reasonable value of Timeout is a problem faced by a user of OWAMP protocol, not by an implementor. A value such as two minutes is very safe. Note that certain applications (such as interactive `one-way ping') might wish to obtain the data faster than that.

As packets are received,

+ Timestamp the received packet.

+ In authenticated or encrypted mode, decrypt the first block (16 octets) of the packet body.

+ Store the packet sequence number, send time, receive time, and the TTL for IPv4 (or Hop Limit for IPv6) from the packet IP header for the results to be transferred.

+ Packets not received within the Timeout are considered lost. They are recorded with their true sequence number, presumed send time, receive time consisting of a string of zero bits, and TTL (or Hop Limit) of 255.

Implementations SHOULD fetch the TTL/Hop Limit value from the IP header of the packet. If an implementation does not fetch the actual TTL value (the only good reason to not do so is inability to access the TTL field of arriving packets), it MUST record the TTL value as 255.

Packets that are actually received are recorded in the order of arrival. Lost packet records serve as indications of the send times of lost packets. They SHOULD be placed either at the point where the receiver learns about the loss or at any later point; in particular, one MAY place all the records that correspond to lost packets at the very end.

Packets that have send time in the future MUST be recorded normally, without changing their send timestamp, unless they have to be discarded. (Send timestamps in the future would normally indicate clocks that differ by more than the delay. Some data -- such as jitter -- can be extracted even without knowledge of time difference. For other kinds of data, the adjustment is best handled by the data consumer on the basis of the complete information in a measurement session, as well as, possibly, external data.)

Packets with a sequence number that was already observed (duplicate packets) MUST be recorded normally. (Duplicate packets are sometimes introduced by IP networks. The protocol has to be able to measure duplication.)

If any of the following is true, the packet MUST be discarded:

+ Send timestamp is more than Timeout in the past or in the future.

+ Send timestamp differs by more than Timeout from the time when the packet should have been sent according to its sequence number.

+ In authenticated or encrypted mode, any of the bits of zero padding inside the first 16 octets of packet body is non-zero.

5. Computing Exponentially Distributed Pseudo-Random Numbers

Here we describe the way exponential random quantities used in the protocol are generated. While there is a fair number of algorithms for generating exponential random variables, most of them rely on having logarithmic function as a primitive, resulting in potentially different values, depending on the particular implementation of the math library. We use algorithm 3.4.1.S in [KNUTH], which is free of the above-mentioned problem, and guarantees the same output on any implementation. The algorithm belongs to the ziggurat family developed in the 1970s by G. Marsaglia, M. Sibuya and J. H. Ahrens [ZIGG]. It replaces the use of logarithmic function by clever bit manipulation, still producing the exponential variates on output.

5.1. High-Level Description of the Algorithm

For ease of exposition, the algorithm is first described with all arithmetic operations being interpreted in their natural sense. Later, exact details on data types, arithmetic, and generation of the uniform random variates used by the algorithm are given. It is an almost verbatim quotation from [KNUTH], p.133.

Algorithm S: Given a real positive number 'mu', produce an exponential random variate with mean 'mu'.

First, the constants

Q[k] = (ln2)/(1!) + (ln2)^2/(2!) + ... + (ln2)^k/(k!), 1 <= k <= 11

are computed in advance. The exact values which MUST be used by all implementations are given in the reference code (see Appendix A). This is necessary to insure that exactly the same pseudo-random sequences are produced by all implementations.

S1. [Get U and shift.] Generate a 32-bit uniform random binary fraction

U = (.b0 b1 b2 ... b31) [note the binary point]

Locate the first zero bit b_j, and shift off the leading (j+1) bits, setting U <- (.b_{j+1} ... b31)

Note: In the rare case that the zero has not been found, it is prescribed that the algorithm return (mu*32*ln2).

S2. [Immediate acceptance?] If U < ln2, set X <- mu*(j*ln2 + U) and terminate the algorithm. (Note that Q[1] = ln2.)

S3. [Minimize.] Find the least k >= 2 such that U < Q[k]. Generate k new uniform random binary fractions U1,...,Uk and set V <- min(U1,...,Uk).

S4. [Deliver the answer.] Set X <- mu*(j + V)*ln2.

5.2. Data Types, Representation, and Arithmetic

The high-level algorithm operates on real numbers -- typically represented as floating point numbers. This specification prescribes that unsigned 64-bit integers be used instead.

u_int64_t integers are interpreted as real numbers by placing the decimal point after the first 32 bits. In other words, conceptually, the interpretation is given by the map:

```
u_int64_t u;

u  |--> (double)u / (2**32)
```

The algorithm produces a sequence of such u_int64_t integers that, for any given value of SID, is guaranteed to be the same on any implementation.

We specify that the u_int64_t representations of the first 11 values of the Q array in the high-level algorithm be as follows:


```
#1      0xB17217F8,
#2      0xEEF193F7,
#3      0xFD271862,
#4      0xFF9D6DD0,
#5      0xFFF4CFD0,
#6      0xFFFEE819,
#7      0xFFFFE7FF,
#8      0xFFFFFE2B,
#9      0xFFFFFFE0,
#10     0xFFFFFFFE,
#11     0xFFFFFFFF
```

For example, Q[1] = ln2 is indeed approximated by 0xB17217F8/(2**32) = 0.693147180601954; for j > 11, Q[j] is 0xFFFFFFFF.

Small integer j in the high-level algorithm is represented as u_int64_t value j * (2**32).

Operation of addition is done as usual on u_int64_t numbers; however, the operation of multiplication in the high-level algorithm should be replaced by

(u, v) |---> (u * v) >> 32.

Implementations MUST compute the product (u * v) exactly. For example, a fragment of unsigned 128-bit arithmetic can be implemented for this purpose (see sample implementation below).

5.3. Uniform Random Quantities

The procedure for obtaining a sequence of 32-bit random numbers (such as U in algorithm S) relies on using AES encryption in counter mode. To describe the exact working of the algorithm, we introduce two primitives from Rijndael. Their prototypes and specification are given below, and they are assumed to be provided by the supporting Rijndael implementation, such as [RIJN].

+ A function that initializes a Rijndael key with bytes from seed (the SID will be used as the seed):

  void KeyInit(unsigned char seed[16]);

+ A function that encrypts the 16-octet block inblock with the specified key, returning a 16-octet encrypted block. Here keyInstance is an opaque type used to represent Rijndael keys:

void BlockEncrypt(keyInstance key, unsigned char inblock[16]);

Algorithm Unif: given a 16-octet quantity seed, produce a sequence of unsigned 32-bit pseudo-random uniformly distributed integers. In OWAMP, the SID (session ID) from Control protocol plays the role of seed.

U1. [Initialize Rijndael key] key <- KeyInit(seed) [Initialize an unsigned 16-octet (network byte order) counter] c <- 0 U2. [Need more random bytes?] Set i <- c mod 4. If (i == 0) set s <- BlockEncrypt(key, c)

U3. [Increment the counter as unsigned 16-octet quantity] c <- c + 1

U4. [Do output] Output the i_th quartet of octets from s starting from high-order octets, converted to native byte order and represented as OWPNum64 value (as in 3.b).

U5. [Loop] Go to step U2.

6. Security Considerations

6.1. Introduction

The goal of authenticated mode to let one passphrase-protect service provided by a particular OWAMP-Control server. One can imagine a variety of circumstances where this could be useful. Authenticated mode is designed to prohibit theft of service.

An additional design objective of the authenticated mode was to make it impossible for an attacker who cannot read traffic between OWAMP-Test sender and receiver to tamper with test results in a fashion that affects the measurements, but not other traffic.

The goal of encrypted mode is quite different: to make it hard for a party in the middle of the network to make results look `better' than they should be. This is especially true if one of client and server does not coincide with either sender or receiver.

Encryption of OWAMP-Control using AES CBC mode with blocks of zeros after each message aims to achieve two goals: (i) to provide secrecy of exchange; (ii) to provide authentication of each message.

6.2. Preventing Third-Party Denial of Service

OWAMP-Test sessions directed at an unsuspecting party could be used for denial of service (DoS) attacks. In unauthenticated mode, servers SHOULD limit receivers to hosts they control or to the OWAMP-Control client.

6.3. Covert Information Channels

OWAMP-Test sessions could be used as covert channels of information. Environments that are worried about covert channels should take this into consideration.

6.4. Requirement to Include AES in Implementations

Notice that AES, in counter mode, is used for pseudo-random number generation, so implementation of AES MUST be included, even in a server that only supports unauthenticated mode.

6.5. Resource Use Limitations

An OWAMP server can consume resources of various kinds. The two most important kinds of resources are network capacity and memory (primary or secondary) for storing test results.

Any implementation of OWAMP server MUST include technical mechanisms to limit the use of network capacity and memory. Mechanisms for managing the resources consumed by unauthenticated users and users authenticated with a username and passphrase SHOULD be separate. The default configuration of an implementation MUST enable these mechanisms and set the resource use limits to conservatively low values.

One way to design the resource limitation mechanisms is as follows: assign each session to a user class. User classes are partially ordered with ``includes'' relation, with one class (``all users'') that is always present and that includes any other class. The assignment of a session to a user class can be based on the presence of authentication of the session, the user name, IP address range, time of day, and, perhaps, other factors. Each user class would have a limit for usage of network capacity (specified in units of bit/second) and memory for storing test results (specified in units of octets). Along with the limits for resource use, current use would be tracked by the server. When a session is requested by a user in a specific user class, the resources needed for this session

are computed: the average network capacity use (based on the sending

schedule) and the maximum memory use (based on the number of packets and number of octets each packet would need to be stored internally -- note that outgoing sessions would not require any memory use). These resource use numbers are added to the current resource use numbers for the given user class; if such addition would take the resource use outside of the limits for the given user class, the session is rejected. When resources are reclaimed, corresponding measures are subtracted from the current use. Network capacity is reclaimed as soon as the session ends. Memory is reclaimed when the data is deleted. For unauthenticated sessions, memory consumed by an OWAMP-Test session SHOULD be reclaimed after the OWAMP-Control connection that initiated the session is closed (gracefully or otherwise). For authenticated sessions, the administrator who configures the service should be able to decide the exact policy, but useful policy mechanisms that MAY be implemented are the ability to automatically reclaim memory when the data is retrieved and the ability to reclaim memory after a certain configurable (based on user class) period of time passes after the OWAMP-Test session terminates.

6.6. Use of Cryptographic Primitives in OWAMP

At an early stage in designing the protocol, we considered using Transport Layer Security (TLS) and IPsec as cryptographic security mechanisms for OWAMP. The disadvantages of those are as follows (not an exhaustive list):

Regarding TLS:

+ While TLS could be used to secure TCP-based OWAMP-Control, but difficult to use to secure UDP-based OWAMP-Test: OWAMP-Test packets, if lost, are not resent, so packets have to be (optionally) encrypted and authenticated while retaining individual usability. Stream-based TLS is not conducive of this.

+ Dealing with streams, does not authenticate individual messages (even in OWAMP-Control). The easiest way out would be to add some known-format padding to each message and verify that the format of the padding is intact before using the message. The solution would thus lose some of its appeal (``just use TLS''); it would also be much more difficult to evaluate the security of this scheme with the various modes and options of TLS -- it would almost certainly not be secure with all. The capacity of an attacker to replace parts of messages (namely, the end) with random garbage could have serious security implications and would need to be analyzed carefully: suppose, for example, that a parameter that is used in some form to control the rate were

replaced by random garbage -- chances are the result (an unsigned integer) would be quite large.

+ Dependent on the mode of use, one can end up with a requirement for certificates for all users and a PKI. Even if one is to accept that PKI is desirable, there just isn't a usable one today.

+ TLS requires a fairly large implementation. OpenSSL, for example, is larger than our implementation of OWAMP as a whole. This can matter for embedded implementations.

Regarding IPsec:

+ What we now call authenticated mode would not be possible (in IPsec you can't authenticate part of a packet).

+ The deployment paths of IPsec and OWAMP could be separate if OWAMP does not depend on IPsec. After nine years of IPsec, only 0.05% of traffic on an advanced backbone network such as Abilene uses IPsec (for comparison purposes with encryption above layer 4, SSH use is at 2-4% and HTTPS use is at 0.2-0.6%). It is desirable to be able to deploy OWAMP on as large of a number of different platforms as possible.

+ The deployment problems of a protocol dependent on IPsec would be especially acute in the case of lightweight embedded devices. Ethernet switches, DSL ``modems,'' and other such devices mostly do not support IPsec.

+ The API for manipulation IPsec from an application is currently poorly understood. Writing a program that needs to encrypt some packets, authenticate some packets, and leave some open -- for the same destination -- would become more of an exercise in IPsec rather than IP measurement.

For the enumerated reasons, we decided to use a simple cryptographic protocol (based on a block cipher in CBC mode) that is different from TLS and IPsec.

6.7. Required Properties of MD5

The protocol makes use of the MD5 hash function to convert a user-supplied passphrase into a key that will be used to encrypt a

short piece of random data (the session key).

In this document we use cryptographic terminology of [MENEZES].

It has long been suspected, and has been conclusively shown recently that MD5 is not a collision-resistant hash function. Since collision resistance was one of design goals of MD5, this casts strong suspicion on the other design goals of MD5, namely preimage resistance and 2nd preimage resistance.

OWAMP does not rely on any of these properties.

The properties of MD5 that are necessary are as follows: (1) it is a function that maps arbitrary length inputs into 128-bit outputs [fixed-length hash function], (2) a change in any bit of the input usually results in a change of a few bits of output [weakened avalanche property], (3) many 128-bit strings have preimages [almost surjective], and (4) the visible special structure of natural-language text possibly present in the passphrase is concealed after application of the function. These are very weak requirements that many functions satisfy. Something resembling CRC-128 would work just as well.

We chose MD5 here because it has the required properties and is widely implemented, understood, and documented. Alternatives would include (1) a non-cryptographic primitive, such as CRC-128, (2) SHA-1 truncated to 128 bits, or (3) a hash function based on AES (using Matyas-Meyer-Oseas, Davies-Meyer, or Miyaguchi-Preneel constructions; we would probably gravitate towards the last one if a block-cipher-based cryptographically secure hash function were required). Note that option 1 would not have any cryptographically relevant properties. We chose not to use it because of lack of well-documented 128-bit checksums; this specification would incur an unnecessary burden precisely defining one, providing test vectors, etc., with no advantage over MD5. Option 2, SHA-1, belongs to the MD4 family that appears to be under suspicion in light of recent developments. To avoid creating an impression that any potential future changes in the status of SHA-1 can affect the status of OWAMP we chose not to use it. Option 3 would result in a hash function that, with the current state of knowledge, would probably be one of the most cryptographically sound. Our requirements 1-4 from the

preceding paragraph, however, do not call for a cryptographically sound hash function. Just as with CRC-128, this specification would need to define the hash function and provide test vectors (and perhaps sample code); we see no advantage in this approach versus using MD5. (Note that the performance advantages of MD5 are irrelevant for this application, as the hash is computed on a relatively short human-supplied string only once per OWAMP-Control session, so if the Miyaguchi-Preneel construction were documented in an RFC, we might just as well have used that.)

6.8. The Use of AES-CBC-MAC

OWAMP relies on AES-CBC-MAC for message authentication. Random IV choice is important for prevention of a codebook attack on the first block; it is unimportant for the purposes of CBC-MAC authentication (it should also be noted that, with its 128-bit block size, AES is more resistant to codebook attacks than ciphers with shorter blocks; we use random IV anyway).

IZP, when decrypted, MUST be zero. It is crucial to check for this before using the message, otherwise existential forgery becomes possible. The complete message for which IZP is decrypted to non-zero MUST be discarded (for both short messages consisting of a few blocks and potentially long messages, such as a response to the Fetch-Session command).

Since OWAMP messages can have different numbers of blocks, the existential forgery attack described in example 9.62 of [MENEZES] becomes a concern. To prevent it (and to simplify implementation), the length of any message becomes known after decrypting the first block of it.

A special case is the first (fixed-length) message sent by the client. There, the token is a concatenation of the 128-bit challenge (transmitted by the server in the clear) and a 128-bit session key (generated randomly by the client, encrypted with AES-CBC with IV=0. Since IV=0, the challenge (a single cipher block) is simply encrypted with the secret key. Therefore, we rely on resistance of AES to chosen plaintext attacks (as the challenge could be substituted by an attacker). It should be noted that the number of blocks of chosen plaintext an attacker can have encrypted with the secret key is limited by the number of sessions the client wants to initiate. An

attacker who knows the encryption of a server's challenge can produce an existential forgery of the session key and thus disrupt the session; however, any attacker can disrupt a session by corrupting the protocol messages in an arbitrary fashion, therefore no new threat is created here; nevertheless, we require that the server never issues the same challenge twice (if challenges are generated randomly, a repetition would occur, on average, after 2^64 sessions; we deem this satisfactory as this is enough even for an implausibly busy server that participates in 1,000,000 sessions per second to go without repetitions for more than 500 centuries). With respect to the second part of the token, an attacker can produce an existential forgery of the session key by modifying the second half of the client's token while leaving the first part intact. This forgery, however, would be immediately discovered by the client when the IZP on the server's next message (acceptance or rejection of the connection) does not verify.

7. IANA Considerations

IANA is requested to allocate a well-known TCP port number for the OWAMP-Control part of the OWAMP protocol.

8. Internationalization Considerations

The protocol does not carry any information in a natural language.

9. Appendix A: Sample C Code for Exponential Deviates

The values in array Q[] are the exact values that MUST be used by all implementations. The rest of this appendix only serves for illustrative purposes.

```
/*
** Example usage: generate a stream of exponential (mean 1)
** random quantities (ignoring error checking during initialization).
** If a variate with some mean mu other than 1 is desired, the output
** of this algorithm can be multiplied by mu according to the rules
** of arithmetic we described.
```

```
** Assume that a 16-octet 'seed' has been initialized
** (as the shared secret in OWAMP, for example)
** unsigned char seed[16];

** OWPrand_context next;

** (initialize state)
** OWPrand_context_init(&next, seed);

** (generate a sequence of exponential variates)
** while (1) {
**    u_int64_t num = OWPexp_rand64(&next);
      <do something with num here>
                ...
** }
*/

#include <stdlib.h>

typedef u_int64_t u_int64_t;

/* (K - 1) is the first k such that Q[k] > 1 - 1/(2^32). */
#define K 12
```



```
#define BIT31   0x80000000UL    /* See if first bit in the lower
                                   32 bits is zero. */
#define MASK32(n)       ((n) & 0xFFFFFFFFUL)

#define EXP2POW32       0x100000000ULL

typedef struct OWPrand_context {
        unsigned char counter[16]; /* Counter (network byte order). */
        keyInstance key;           /* Key to encrypt the counter. */
        unsigned char out[16];     /* The encrypted block. */
} OWPrand_context;

/*
** The array has been computed according to the formula:
**
**       Q[k] = (ln2)/(1!) + (ln2)^2/(2!) + ... + (ln2)^k/(k!)
**
** as described in algorithm S. (The values below have been
```

```
** multiplied by 2^32 and rounded to the nearest integer.)
** These exact values MUST be used so that different implementation
** produce the same sequences.
*/
static u_int64_t Q[K] = {
        0,          /* Placeholder - so array indices start from 1. */
        0xB17217F8,
        0xEEF193F7,
        0xFD271862,
        0xFF9D6DD0,
        0xFFF4CFD0,
        0xFFFEE819,
        0xFFFFE7FF,
        0xFFFFFE2B,
        0xFFFFFFE0,
        0xFFFFFFFE,
        0xFFFFFFFF
};

/* this element represents ln2 */
#define LN2 Q[1]

/*
** Convert an unsigned 32-bit integer into a u_int64_t number.
*/
u_int64_t
OWPulong2num64(u_int32_t a)
{
        return ((u_int64_t)1 << 32) * a;
}
```



```
/*
** Arithmetic functions on u_int64_t numbers.
*/

/*
** Addition.
*/
u_int64_t
OWPnum64_add(u_int64_t x, u_int64_t y)
{
        return x + y;
```

```
}

/*
** Multiplication.  Allows overflow.  Straightforward implementation
** of Algorithm 4.3.1.M (p.268) from [KNUTH].
*/
u_int64_t
OWPnum64_mul(u_int64_t x, u_int64_t y)
{
        unsigned long w[4];
        u_int64_t xdec[2];
        u_int64_t ydec[2];

        int i, j;
        u_int64_t k, t, ret;

        xdec[0] = MASK32(x);
        xdec[1] = MASK32(x>>32);
        ydec[0] = MASK32(y);
        ydec[1] = MASK32(y>>32);

        for (j = 0; j < 4; j++)
                w[j] = 0;

        for (j = 0; j < 2; j++) {
                k = 0;
                for (i = 0; ; ) {
                        t = k + (xdec[i]*ydec[j]) + w[i + j];
                        w[i + j] = t%EXP2POW32;
                        k = t/EXP2POW32;
                        if (++i < 2)
                                continue;
                        else {
                                w[j + 2] = k;
                                break;
                        }
                }
```



```
        }

        ret = w[2];
        ret <<= 32;
```

```
        return w[1] + ret;
}


/*
** Seed the random number generator using a 16-byte quantity 'seed'
** (== the session ID in OWAMP). This function implements step U1
** of algorithm Unif.
*/

void
OWPrand_context_init(OWPrand_context *next, unsigned char *seed)
{
        int i;

        /* Initialize the key */
        rijndaelKeyInit(next->key, seed);

        /* Initialize the counter with zeros */
        memset(next->out, 0, 16);
        for (i = 0; i < 16; i++)
                next->counter[i] = 0UL;
}


/*
** Random number generating functions.
*/

/*
** Generate and return a 32-bit uniform random string (saved in the less
** significant half of the u_int64_t).  This function implements steps
** U2-U4 of the algorithm Unif.
*/
u_int64_t
OWPunif_rand64(OWPrand_context *next)
{
        int j;
        u_int8_t  *buf;
        u_int64_t  ret = 0;

        /* step U2 */
        u_int8_t i = next->counter[15] & (u_int8_t)3;
        if (!i)
```

```
                rijndaelEncrypt(next->key, next->counter, next->out);

        /* Step U3.  Increment next.counter as a 16-octet single
           quantity in network byte order for AES counter mode. */
        for (j = 15; j >= 0; j--)
                if (++next->counter[j])
                        break;

        /* Step U4.  Do output.  The last 4 bytes of ret now contain the
           random integer in network byte order */
        buf = &next->out[4*i];
        for (j=0; j<4; j++) {
                ret <<= 8;
                ret += *buf++;
        }
        return ret;
}

/*
** Generate an exponential deviate with mean 1.
*/
u_int64_t
OWPexp_rand64(OWPrand_context *next)
{
        unsigned long i, k;
        u_int32_t j = 0;
        u_int64_t U, V, J, tmp;

        /* Step S1. Get U and shift */
        U = OWPunif_rand64(next);

        while ((U & BIT31) && (j < 32)) { /* Shift until first 0. */
                U <<= 1;
                j++;
        }
        /* Remove the 0 itself. */
        U <<= 1;

        U = MASK32(U);  /* Keep only the fractional part. */
        J = OWPulong2num64(j);

        /* Step S2.  Immediate acceptance? */
        if (U < LN2)       /* return  (j*ln2 + U) */
                return OWPnum64_add(OWPnum64_mul(J, LN2), U);

        /* Step S3.  Minimize. */
        for (k = 2; k < K; k++)
```

```
                if (U < Q[k])
```



```
                        break;
        V = OWPunif_rand64(next);
        for (i = 2; i <= k; i++) {
                tmp = OWPunif_rand64(next);
                if (tmp < V)
                        V = tmp;
        }

        /* Step S4.  Return (j+V)*ln2 */
        return OWPnum64_mul(OWPnum64_add(J, V), LN2);
}
```

10. Appendix B: Test Vectors for Exponential Deviates

It is important that the test schedules generated by different implementations from identical inputs be identical. The non-trivial part is the generation of pseudo-random exponentially distributed deviates. To aid implementors in verifying interoperability, several test vectors are provided. For each of the four given 128-bit values of SID represented as hexadecimal numbers, 1,000,000 exponentially distributed 64-bit deviates are generated as described above. As they are generated, they are all added to each other. The sum of all 1,000,000 deviates is given as a hexadecimal number for each SID. An implementation MUST produce exactly these hexadecimal numbers. To aid in the verification of the conversion of these numbers to values of delay in seconds, approximate values are given (assuming lambda=1). An implementation SHOULD produce delay values in seconds that are close to the ones given below.

```
    SID = 0x2872979303ab47eeac028dab3829dab2
    SUM[1000000] = 0x000f4479bd317381 (1000569.739036 seconds)

    SID = 0x0102030405060708090a0b0c0d0e0f00
    SUM[1000000] = 0x000f433686466a62 (1000246.524512 seconds)

    SID = 0xdeadbeefdeadbeefdeadbeefdeadbeef
    SUM[1000000] = 0x000f416c8884d2d3 (999788.533277 seconds)

    SID = 0xfeed0feed1feed2feed3feed4feed5ab
```

```
SUM[1000000] = 0x000f3f0b4b416ec8 (999179.293967 seconds)
```


11. Normative References

[AES] Advanced Encryption Standard (AES), http://csrc.nist.gov/encryption/aes/

[RFC1305] D. Mills, `Network Time Protocol (Version 3) Specification, Implementation and Analysis', RFC 1305, March 1992.

[RFC1321] R. Rivest, `The MD5 Message-Digest Algorithm', RFC 1321, April 1992.

[RFC2026] S. Bradner, `The Internet Standards Process -- Revision 3', RFC 2026, October 1996.

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

[RFC2330] V. Paxon, G. Almes, J. Mahdavi, M. Mathis, `Framework for IP Performance Metrics' RFC 2330, May 1998.

[RFC2474] K. Nichols, S. Blake, F. Baker, D. Black, `Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers', RFC 2474, December 1998.

[RFC2679] G. Almes, S. Kalidindi, and M. Zekauskas, `A One-way Delay Metric for IPPM', RFC 2679, September 1999.

[RFC2680] G. Almes, S. Kalidindi, and M. Zekauskas, `A One-way Packet Loss Metric for IPPM', RFC 2680, September 1999.

[RFC2836] S. Brim, B. Carpenter, F. Le Faucheur, `Per Hop Behavior Identification Codes', RFC 2836, May 2000.

12. Informative References

[ZIGG] G. Marsaglia, M. Sibuya, and J. H. Ahrens, Communications of ACM, 15 (1972), 876-877.

[MENEZES] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, Handbook of Applied Cryptography, CRC Press, revised reprint with updates, 1997.

[KNUTH] D. Knuth, The Art of Computer Programming, vol.2, 3rd edition, 1998.

[RIJN] Reference ANSI C Implementation of Rijndael http://www.esat.kuleuven.ac.be/~rijmen/rijndael/rijndaelref.zip

[RIPE] RIPE NCC Test-Traffic Measurements home, http://www.ripe.net/test-traffic/.

[RIPE-NLUUG] H. Uijterwaal and O. Kolkman, `Internet Delay Measurements Using Test-Traffic', Spring 1998 Dutch Unix User Group Meeting, http://www.ripe.net/test-traffic/Talks/9805_nluug.ps.gz.

[SURVEYOR] Surveyor Home Page, http://www.advanced.org/surveyor/.

[SURVEYOR-INET] S. Kalidindi and M. Zekauskas, `Surveyor: An Infrastructure for Network Performance Measurements', Proceedings of INET'99, June 1999. http://www.isoc.org/inet99/proceedings/4h/4h_2.htm

13. Authors' Addresses

Stanislav Shalunov
Internet2
3025 Boardwalk Dr, Suite 200
Ann Arbor, MI 48108
Telephone: +1-734-995-7060

Email: shalunov@internet2.edu
SIP: shalunov@internet2.edu

Benjamin Teitelbaum
Internet2
3025 Boardwalk Dr, Suite 200
Ann Arbor, MI 48108
Email: ben@internet2.edu
SIP: ben@internet2.edu

Anatoly Karp
4710 Regent St, Apt 81B
Madison, WI 53705
Telephone: +1-608-347-6255
Email: ankarp@charter.net

Jeff W. Boote
Internet2
3025 Boardwalk Dr, Suite 200
Ann Arbor, MI 48108
Email: boote@internet2.edu
SIP: boote@internet2.edu

Matthew J. Zekauskas
Internet2
3025 Boardwalk Dr, Suite 200
Ann Arbor, MI 48108
Email: matt@internet2.edu

Intellectual Property

might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

Copyright Statement

Acknowledgments

We would like to thank Guy Almes, Hamid Asgari, Steven Van den Berghe, Eric Boyd, Robert Cole, Joan Cucchiara, Stephen Donnelly, Susan Evett, Kaynam Hedayat, Petri Helenius, Kitamura Yasuichi, Daniel H. T. R. Lawson, Will E. Leland, Bruce A. Mah, Allison Mankin, Al Morton, Attila Pasztor, Randy Presuhn, Matthew Roughan, Andy Scherrer, Henk Uijterwaal, and Sam Weiler for their comments,

suggestions, reviews, helpful discussion and proof-reading.

Expiration date: June 2005