Network Working Group   Stanislav Shalunov
Internet Draft   Benjamin Teitelbaum
Expiration Date: November 2004   Anatoly Karp
                                               Jeff W. Boote
                                          Matthew J. Zekauskas
                                                     Internet2
                                                      May 2004

A One-way Active Measurement Protocol (OWAMP)
<draft-ietf-ippm-owdp-08.txt>

1. Status of this Memo

2. 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.

3. Motivation and Goals

   The IETF IP Performance Metrics (IPPM) working group has proposed
   draft standard metrics for one-way packet delay [RFC2679] and loss
   [RFC 2680] 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 system (GPS) and CDMA based time sources, hosts
   increasingly have available to them very accurate time
   sources--either directly or through their proximity to 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, too).
   Additionally, OWAMP 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.

3.1. 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 active across diff-serv networks.
Additionally, OWAMP-Control supports per-session encryption and
authentication for both test and control traffic, measurement servers
which may 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 SNMP or proprietary network management protocols is
required, gateways may be created.

3.2. **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;

      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.

```
     +----------------+                    +-----------------+
     | Session-Sender |--OWAMP-Test-->| Session-Receiver |
     +----------------+                    +-----------------+
       ^                                        ^
       |                                        |
       |                                        |
       |                                        |
       |   +----------------+<----------------+
       |   |     Server     |<-------+
       |   +----------------+        |
       |      ^                      |
       |      |                      |
       | OWAMP-Control          OWAMP-Control
       |      |                      |
       v      v                      v
     +----------------+    +-----------------+
     | Control-Client |    |   Fetch-Client  |
     +----------------+    +----------------+
```
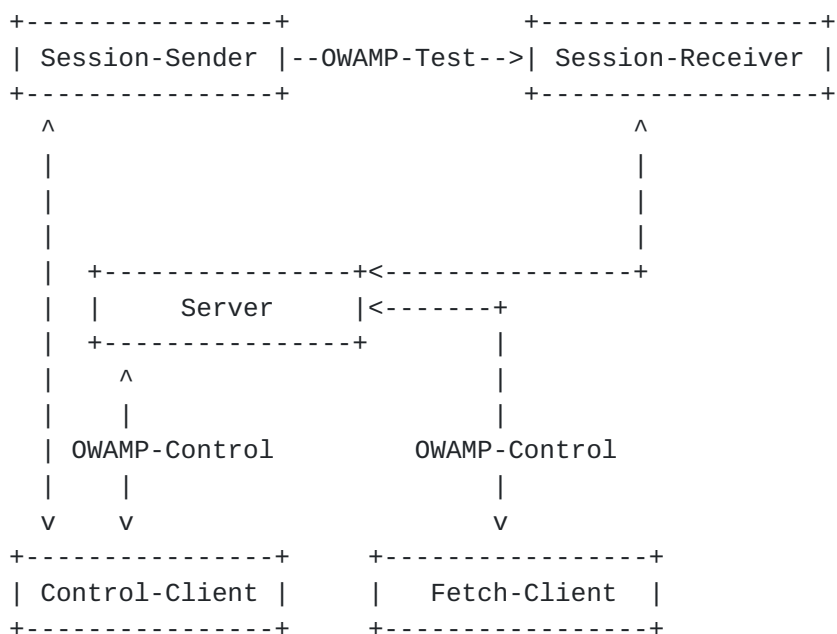
   (Unlabeled links in the figure are unspecified by this draft and may
   be proprietary protocols.)

   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.

```
     +-----------------+                    +-----------------+
     | Control-Client  |<--OWAMP-Control-->| Server          |
     | Fetch-Client    |                    |                 |
     | Session-Sender  |---OWAMP-Test----->| Session-Receiver |
     +-----------------+                    +-----------------+
```
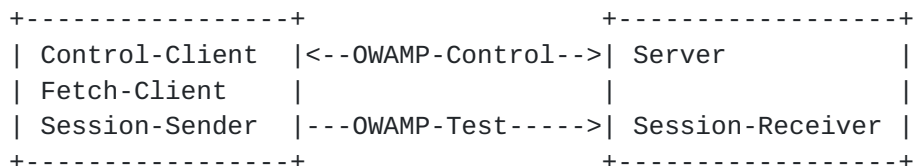
   Finally, because many Internet paths include segments that transport
   IP over ATM, delay and loss measurements can include the effects of
   ATM segmentation and reassembly (SAR). Consequently, OWAMP has been
   designed to allow for small test packets that would fit inside the

payload of a single ATM cell (this is only achieved in
unauthenticated and encrypted modes).

4. 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-Session 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.

5. OWAMP-Control

Each type of OWAMP-Control message has a fixed length. The recipient
will know the full length of a message after examining 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.

5.1. Connection Setup

Before either a Control-Client or a Fetch-Client can issue commands
of a Server, it must 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:

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   |                    Unused (12 octets)                         |
   |                                                               |
   |+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-++-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                         Modes                                 |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   |                   Challenge (16 octets)                       |
   |                                                               |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The following mode values are meaningful: 1 for unauthenticated, 2
for authenticated, 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, 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 doesn't wish to communicate with
the client and MAY close the connection immediately. The client
SHOULD close the connection if it gets 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:

```
         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
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |                             Mode                              |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |                                                               |
        .                                                               .
        .                     Username (16 octets)                      .
        .                                                               .
        |                                                               |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |                                                               |
        .                                                               .
        .                      Token (32 octets)                        .
        .                                                               .
        |                                                               |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |                                                               |
        .                                                               .
        .                    Client-IV (16 octets)                      .
        .                                                               .
        |                                                               |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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 of which shared secret
the client wishes to use to authenticate or encrypt and 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. 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.

The server MUST respond with the following message:

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   |                   Unused, MBZ (15 octets)                     |
   |                                                               |
   |                                       +-+-+-+-+-+-+-+-+
   |                                       |    Accept     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   |                    Server-IV (16 octets)                      |
   |                                                               |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                     Uptime (Timestamp)                        |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |              Integrity Zero Padding (8 octets)                |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Unused 15-octet part MUST be zero. The client MUST ignore its
value.

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

A zero value in the Accept field means that the server accepts the
authentication and is willing to conduct further transactions. A
value of 1 means that the server does not accept the authentication
provided by the client or, for some other reason, is not willing to
conduct further transactions in this OWAMP-Control session. All
other values are reserved. The client MUST interpret all values of
Accept other than 0 and 1 as 1. This way, other values are available
for future extensions. If a negative 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, 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 can not 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 is treated the same way as Integrity Zero
Padding in the next section and beyond.

The previous transactions constitute connection setup.

5.2. 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-Session, Fetch-Session. The command Stop-
Session 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-Session (order unspecified),
it is said to be conducting active measurements.

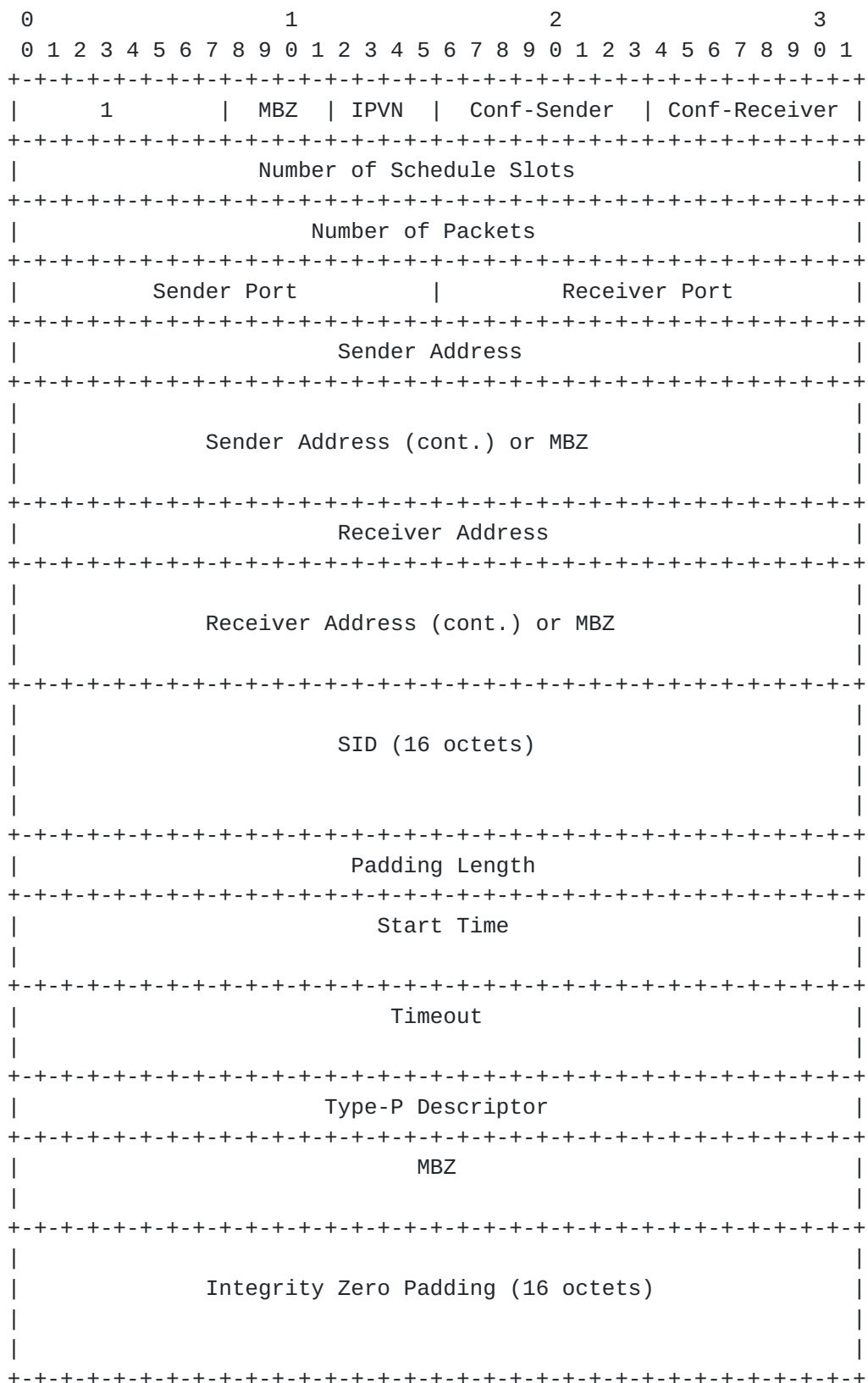While conducting active measurements, the only command available is
Stop-Session.

These commands are described in detail below.

5.3. 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:

```
        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
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |       1       |  MBZ  | IPVN  |  Conf-Sender  | Conf-Receiver |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                   Number of Schedule Slots                    |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                      Number of Packets                        |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |          Sender Port          |         Receiver Port         |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                        Sender Address                         |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                                                               |
       |                 Sender Address (cont.) or MBZ                 |
       |                                                               |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                       Receiver Address                        |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                                                               |
       |                Receiver Address (cont.) or MBZ                |
       |                                                               |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                                                               |
       |                        SID (16 octets)                        |
       |                                                               |
       |                                                               |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                        Padding Length                         |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                          Start Time                           |
       |                                                               |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                           Timeout                             |
       |                                                               |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                       Type-P Descriptor                       |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                             MBZ                               |
       |                                                               |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                                                               |
       |             Integrity Zero Padding (16 octets)               |
       |                                                               |
       |                                                               |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   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):

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |    Slot Type  |                                               |
   +-+-+-+-+-+-+-+-+                  MBZ                           |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    Slot Parameter (Timestamp)                 |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

These are immediately followed by Integrity Zero Padding:

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   |                 Integrity Zero Padding (16 octets)            |
   |                                                               |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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. In the case
of IP version number being 4, twelve octets follow the four-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 Integrity Zero

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

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 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 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 Type-P Descriptor are 00, then subsequent 6
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 subsequent 16 bits
specify the requested Per Hop Behavior 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, Type-P Descriptor is to be used to configure
the sender to send packets according to its value. If Conf-Sender is
not set, Type-P Descriptor is a declaration of how the sender will be
configured.

If Conf-Sender is set and the server doesn't recognize Type-P
Descriptor, 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 the
session regardless of the value of Type-P Descriptor.

Integrity Zero Padding MUST be all zeros in this and all subsequent
messages that use zero padding. The recipient of a message where
zero padding 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, Integrity Zero
Padding 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
Integrity Zero Padding 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:

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |    Accept     |    Unused     |             Port              |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-|
   |                                                               |
   |                       SID (16 octets)                         |
   |                                                               |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   |             Integrity Zero Padding (12 octets)                |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
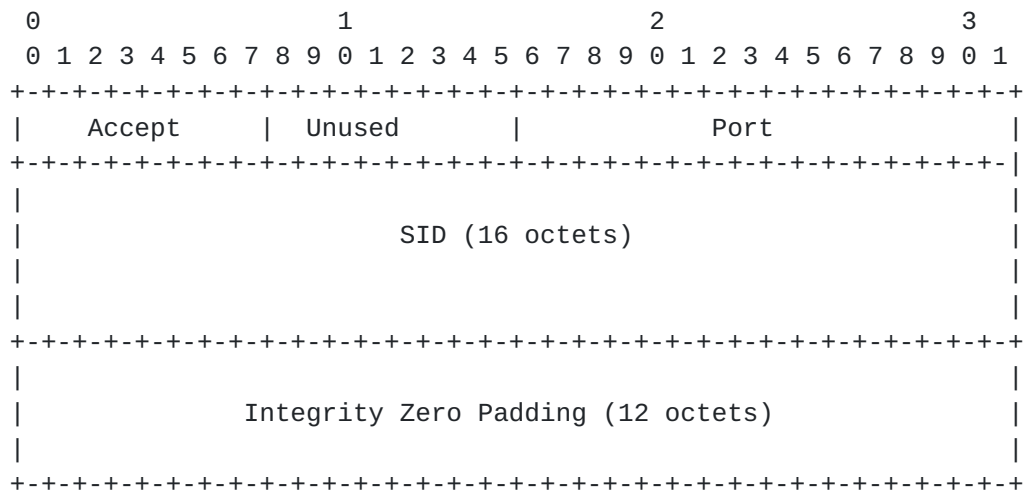
In this message, zero in the Accept field means that the server is
willing to conduct the session. A value of 1 indicates rejection of
the request. All other values are reserved.

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

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, Port field is unused. If only Conf-Sender was set,
Port is the port to expect OWAMP-Test packets from. If only Conf-
Receiver was set, Port is the port to send OWAMP-Test packets to.

If only Conf-Sender was set, 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 4-octet IPv4 IP number
belonging to the generating machine, 8-octet timestamp, and 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 4 octets of an IPv6 address can be used instead. Note that SID
is always chosen by the receiver. If truly random values are not
available, it is important that SID be made unpredictable as
knowledge of SID might be used for access control.

5.4. **Send Schedules**

The sender and the receiver need to both know the same send schedule.
This way, when packets are lost, the receiver knows when they were
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 slots 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 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 exponentially
distributed time with 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 must 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:

+ 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.

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.

5.5. 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:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      2        |                                               |
+-+-+-+-+-+-+-+-+                                               |
|                    Unused (15 octets)                         |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                Integrity Zero Padding (16 octets)             |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Accept     |                                               |
+-+-+-+-+-+-+-+-+                                               |
|                    Unused (15 octets)                         |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                Integrity Zero Padding (16 octets)             |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

If Accept is 1, the Start-Sessions request was rejected; zero means
that the command was accepted. All other values are reserved. The
server MAY and the client SHOULD close the connection in the case of
a negative response.

The server SHOULD start all OWAMP-Test streams immediately after it
sends the response or immediately after their specified start times,
whichever is later. (Note that a client can effect an immediate
start by specifying in Request-Session a Start Time in the past.) If
the client represents a Sender, the client SHOULD start its OWAMP-
Test streams immediately after it sees the Control-Ack response from
the Server.

. **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:

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |      3        |    Accept     |           Unused              |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    Number of Sessions                         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    Unused (8 octets)                          |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   |              Integrity Zero Padding (16 octets)               |
   |                                                               |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
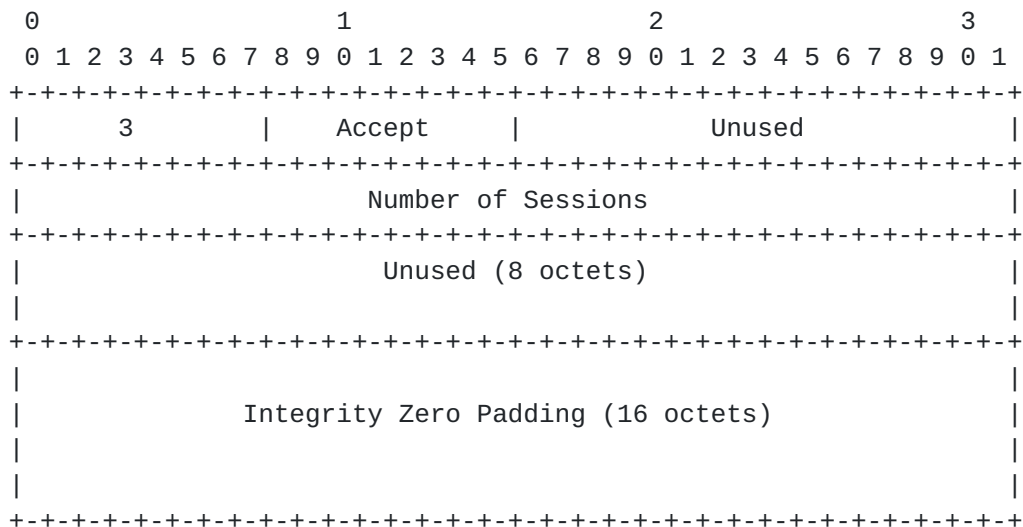
   This is immediately followed by 0 or more session packets sent
   descriptions (the number of session packets sent records is specified
   in the 'Number of Sessions' field above):

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-|
   |                                                               |
   |                     SID (16 octets)                           |
   |                                                               |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                  Session Packets Sent                         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   |              Integrity Zero Padding (12 octets)               |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
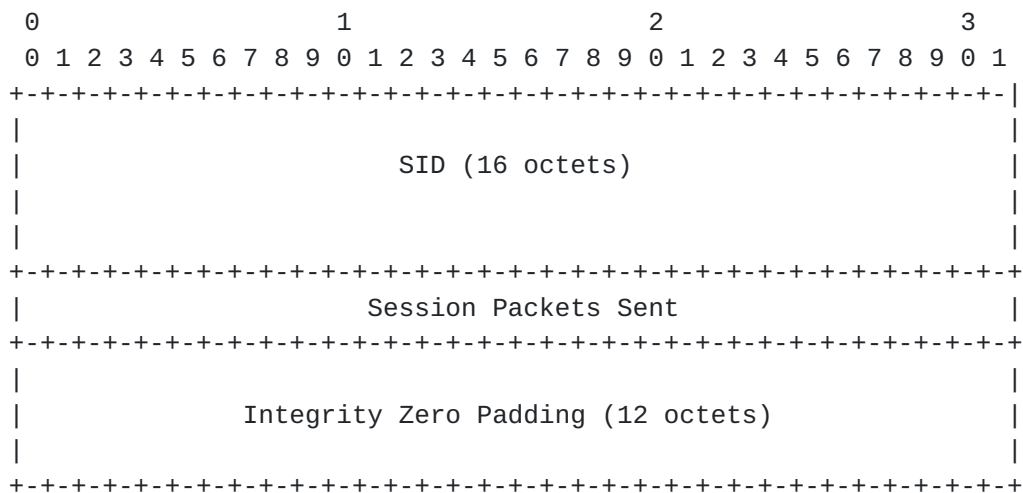
   All these messages comprise one logical message: the Stop-Session
   command.

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

   Accept values of 1 indicate a failure of some sort. Zero values
   indicate normal (but possibly premature) completion. All other

values are reserved.

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 packets sent
records that immediately follow the Stop-Sessions message.

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 all 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 packets sent record represents one OWAMP-Test session
and contains the session identifier (SID) and the number of packets
sent in that session. For completed sessions, Session Packets Sent
will equal NumPackets from the Request-Session. Session Packets Sent
MAY be all ones (0xFFFFFFFF); in this case, the sender of the Stop-
Sessions command could not determine the number of packets sent
(perhaps, due to some internal error such as a process crash); this
special value SHOULD NOT be sent under normal operating conditions.

If the OWAMP-Control connection associated with an OWAMP-Test
receiver receives the (0xFFFFFFFF) special value for the Session
Packets Sent, or 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 any records of lost
packets that follow (in other words, have greater sequence number
than) the last packet that was actually received. This will help
differentiate between packet losses that occurred in the network and
the sender crashing. When the results of such an OWAMP-Test session
or an OWAMP-Test session that was prematurely aborted successfully
(with confirmation) are later fetched using Fetch-Session, the
original number of packets MUST be supplied in the reproduction of
the Request-Session command.

If a receiver of an OWAMP-Test session learns through 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.

[5.7](#). **Fetch-Session**

The format of this client command is as follows:

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |      4         |                                             |
   +-+-+-+-+-+-+-+-+                                              |
   |                     Unused (7 octets)                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                       Begin Seq                              |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                        End Seq                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                              |
   |                      SID (16 octets)                         |
   |                                                              |
   |                                                              |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                              |
   |              Integrity Zero Padding (16 octets)              |
   |                                                              |
   |                                                              |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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 Control-Ack message. Again, 1 in the
Accept field means rejection of command. Zero means that data will
follow. All other values are reserved.

If Accept was 0, the server then MUST send the OWAMP-Test session
data in question, followed by 16 octets of Integrity Zero Padding.

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.

+ The number of packet records that will follow represented as an
   unsigned 4-octet integer. 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.

+ 12 octets of Integrity Zero Padding.

+ Zero or more (as specified) packet records.

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, and 2 octets of receive
timestamp error estimate and 1 octet of TTL (in this order):

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
00|                          Seq Number                           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
04|                        Send Timestamp                         |
08|                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
12|      Send Error Estimate      |                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
16|                        Receive Timestamp                      |
   +                               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
20|                               |     Receive Error Estimate    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
24|     TTL       |
   +-+-+-+-+-+-+-+-+
```

Packet records are sent out in the same order they are made when the
results of the session are recorded. 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 by the receive
timestamp consisting of a string of zero bits and an error estimate
with Multiplier=1, Scale=64, and S=0 (see OWAMP-Test description for
definition of these quantities and explanation of timestamp format
and error estimate format).

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 Integrity
Zero Padding that follow the session data and conclude the response
to Fetch-Session.)

6. OWAMP-Test

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

As OWAMP-Control, OWAMP-Test has three modes: unauthenticated,
authenticated, and encrypted. All OWAMP-Test sessions 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.)

6.1. Sender Behavior

The sender sends the receiver a stream of packets with schedule as
specified in the Request-Session command. The sender SHOULD set the
TTL 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:

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    Sequence Number                           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                        Timestamp                             |
   |                                                              |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |        Error Estimate         |                              |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                              |
   |                                                              |
   .                                                              .
   .                     Packet Padding                          .
   .                                                              .
   |                                                              |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

For authenticated and encrypted modes:

```
     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
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                     Sequence Number                           |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    |              Integrity Zero Padding (12 octets)               |
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                       Timestamp                               |
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |         Error Estimate        |                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               |
    |              Integrity Zero Padding (6 octets)                |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    .                                                               .
    .                       Packet Padding                          .
    .                                                               .
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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:

```
     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
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                   Integer part of seconds                     |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                  Fractional part of seconds                   |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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

```
     0                   1
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |S|Z|   Scale   |   Multiplier  |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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 0 and are incremented by 1 for each
subsequent packet.

The minimum data segment length is therefore 14 octets in
unauthenticated mode, and 32 octets in 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 to reduce the
time between a timestamp is obtained by a sender and 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 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 ECB mode. The key to use is the same key as is
used for the corresponding OWAMP-Control session (where it is used in
a different chaining mode). Electronic Cookbook (ECB) 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 the same key as is used for
the corresponding OWAMP-Control session. 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.

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.

7. Receiver Behavior

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 don't travel for longer than
TTL seconds). 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 first block (16
   octets) of packet body.

+ Store the packet sequence number, send time, receive time, and the
   TTL 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 seqno, presumed send time, receive
   time consisting of a string of zero bits, and TTL of 255.

Implementations SHOULD fetch the TTL 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 seqno.

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

8. 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.

8.1. 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). 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 decimal 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.

8.2. **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 which is
guaranteed to be the same on any implementation. Any further
interpretation (such as given by (1)) is done by the application, and
is not part of this specification.

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

8.3. **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].

+ This function initializes a Rijndael key with bytes from 'seed'

   void KeyInit(unsigned char seed[16]);

+ This function encrypts the 16-octet block 'inblock' with the '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.

9. **Security Considerations**

   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.

   Additional design objective of 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
   doesn't coincide with neither sender nor 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.

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

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

   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.

10. **IANA Considerations**

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

11. Internationalization Considerations

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

12. Appendix A: Sample Implementation of Exponential Deviate Computation

```
/*
** 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]; /* 16-octet counter (network byte order)  */
        keyInstance key;           /* key used to encrypt the counter.       */
```

```
        unsigned char out[16];    /* the encrypted block is kept there.    */
} 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 a mean 1 exponential deviate.
*/
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 find 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);
}
```

[13](#). [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)
```

## [14](#). 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.

## 15. Informative References

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

     [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

## 16. Authors' Addresses

Stanislav Shalunov <shalunov@internet2.edu>

Benjamin Teitelbaum <ben@internet2.edu>

Anatoly Karp <ankarp@yahoo.com>

Jeff Boote <boote@internet2.edu>

Matthew J. Zekauskas <matt@internet2.edu>

Expiration date: November 2004