One-Way Ping (OWAMP): An Internet2 Cookbook

Disclosure/Disclaimer

This material is based in part on work supported by the National Science Foundation (NSF) under Grant No. ANI-0314723. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

This document was developed to be used in conjunction with an Internet2 Network Performance Workshop; for more information on these workshops (upcoming and past), see: http://e2epi.internet2.edu/network-perf-wk/.

One-Way Ping (OWAMP) is heavily used in the E2E piPEs project and the Abilene Measurement Infrastructure. More information on this tool can be found at http://e2epi.internet2.edu/owamp/.

This cookbook has two parts: an Overview of the tool, with examples of its usefulness, and an Installation Guide, that walks you through setting up One-Way Ping (owampd) servers at your location.

An Overview of OWAMP

What Is It?

OWAMP is a sample implementation of the One-Way Active Measurement Protocol (OWAMP) being developed by the Internet Engineering Task Force's (IETF's) IPPM Working Group for latency and delay measurements. (Following the fine tradition of FTP, the application has been named after the protocol.) Information on the version of the protocol that OWAMP, the application, currently implements can be found at: http://www.ietf.org/internet-drafts/draft-ietr-ippm-owdp-014.txt.

Motivation

The main motivation for developing OWAMP was to find problems in the network:

- Congestion usually happens in one direction first…
- Routing (asymmetric, or just changes)
- SNMP polling intervals mask high queue levels that active probes can show

There have been many implementations of One-Way delay over the years (Surveyor, Ripe, etc.). The largest barrier to adoption has been interoperability of these implementations. The solution to these kinds of problems is to develop implementations that conform to accepted standards. This effort is our attempt to do that.

Methodology

OWAMP relies on the fact that time sources are much easier to come by than they once were. This makes it possible for one-way latency measurements to be collected across a broad mesh of network paths. Additionally, the open source nature of this implementation makes it possible for one-way metrics to become as common as round-trip metrics (from tools like ping).

Congestion typically only happens in one direction of a given network path. One-way metrics are the most straightforward way to isolate these effects. Active measurements such as one-way latency measurements from end-to-end are arguably one of the best ways to determine if a given application will work because the diagnostic tool is basically performing as close to the same actions as the real application.

Control Protocol

The client makes requests for tests with a server. The protocol has:

- Support for authentication and authorization
- Ability to configure tests
- Receiver end-point controlled port numbers
- Extremely configurable send schedule
- Configurable packet sizes
- Ability to start/stop tests
- Ability to retrieve results
- Provisions for dealing with partial session results

Test Protocol

Packets can be open, authenticated, or encrypted. The protocol is effectively a packet format.

Sample Implementation

The OWAMP applications are:

- owampd daemon
- owping client

Functions and Features

There are different functions and features for the client and daemon.

Client (owping)

The command line arguments were made as similar to ping as possible.

- owping client requests OWD tests from an OWAMP server
- Client can be sender or receiver
- Communication can be “open”, “authenticated”, or “encrypted”
- Supports the setup of many tests concurrently
- Supports the buffering of results on the server for later retrieval

Daemon (owampd)

owampd is a standard accept/fork style Unix daemon:

- Accepts requests for OWD tests
- Responds with accepted/denied
- Tests are formally started with a StartSessions message from the client.
- Runs tests
- Sessions with packets received at the server are buffered for later retrieval

Resource Allocation

The parent owampd keeps track of current resource utilization needed to implement policy. Each connection is 'classified' (authentication) and each classification is associated with a set of hierarchical limits that are used to make policy decisions (authorization):

- Bandwidth (bandwidth)
- Session buffer (disk)
- Data retention (delete_on_fetch)
- Connection policy (allow_open_mode)

There is no time-dependent dimension to resource allocation in owampd. It currently treats all allocations as immediate but, since it has a complete schedule as part of the request, there is no reason this could not be added in the future.

Architecture

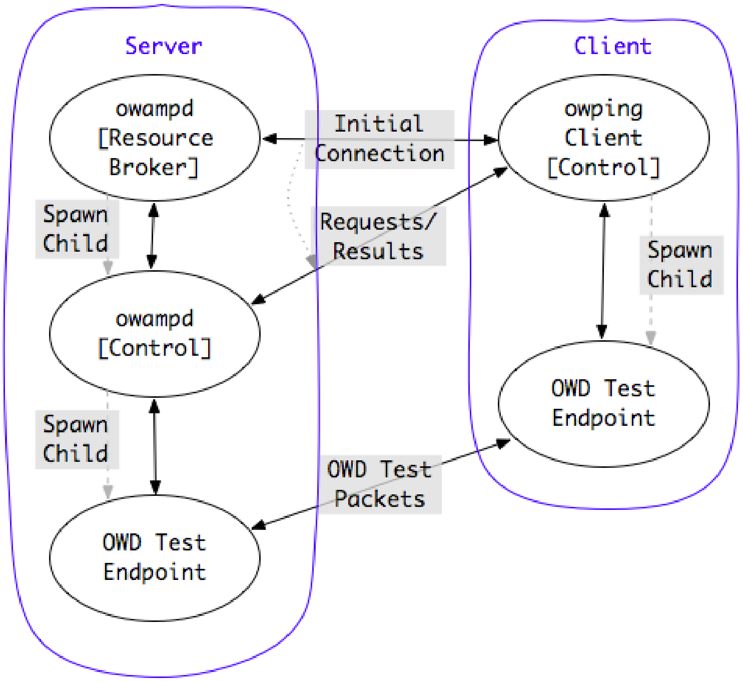


Figure 1: Control Flow

Overview

OWAMP is a typical client-server application. The owping client contacts the owampd daemon on the peer host to request a specific test. The request includes an indication of the complete send schedule as well as parameters to indicate packet characteristics. owampd is responsible for accepting or denying the request.

owampd has been developed as a classic accept/fork daemon. The master daemon process listens for new network connections and also manages the resources for all child owampd processes. When a connection comes in, owampd forks a child process to handle the requests from that connection.

The child process handles all encryption and communication issues with the client, as well as all static resource limits. Static resource limits are those not dependent upon what is currently happening on the node. For example, the request broker can easily determine if the given client is allowed to do open mode (unauthenticated) tests without talking to the master daemon.

Once the request broker process determines the request is valid, it makes a request to the master daemon for the resources and time period requested by the client. If the master daemon has the resources available it grants the request.

Once a given test session is accepted, the client and server both fork off One-Way Delay (OWD) Test Endpoint children to run that test. A single control connection can spawn off any number of test sessions. However, once pending test sessions are started by the start session message all sessions must be completed before more sessions can be requested using the same control connection. Additionally, the test sessions need to be completed before data can be retrieved using the same control connection. Of course, a client could open another control connection to the peer to start additional sessions, or to request intermediate results from any of the active sessions.

Test Endpoints

The OWD Test Endpoint processes are implemented using identical code on both the server and the client. The OWD Test Endpoint processes send and receive packets according to the agreed upon schedule. OWD Test Endpoint processes exit and returns the exit status of the test to the Control parent process when the test sessions are complete. Test sessions are declared complete by both sides of the test a specified timeout period after the last packet is sent. The client can then fetch the results for the send test sessions that are buffered by the server. (The client will already have the results for receive sessions since the client receives those packets.) The server can be configured to delete the buffered data when the data is fetched the first time, or it can be buffered indefinitely and cleaned by an external process.

General Requirements

Getting a good stable NTP configuration is the most challenging task for obtaining good owamp results. You need:

- NTP (ntpd) synchronized clock on the local system
- Specific configuration requirements as specified in the NTP cookbook

Operational Concerns

Concerns include time and firewalls; time issues include:

- NTP issues predominate the problems
- Determining an accurate timestamp "error" is in many ways more difficult than getting a "very good" timestamp
- Working as an "open" server requires UTC time source (For predefined test peers, other options available)

Firewall issues include:

- TCP ports need to be opened for control communication
- UDP ports need to be opened for test traffic

There is a trade-off decision that needs to be made here. System administrators like to have a single port-range defined for applications so the traffic generated by those applications can be classified. However, using a well-known predefined port range would also allow network hardware vendors to also classify the test traffic. This would allow vendors to prioritize test traffic to make their hardware look better. Therefore, because using a single predefined port range would be problematic, owampd allows the installer to define a specific port range for test sessions that are received on the host as a compromise position.

Policy Issues

The policy issues can best be grouped into two categories. First, it is important to ensure that an owampd server is a good network citizen, that it does not use more local host and network resources than it should, and the integrity of the owampd server and the data produced is protected (*see* ***Security Considerations****, below*). And second, controls need to be in place to allow the available resources to be partitioned among the valid users of the server (*see* ***Resource Consumption****, below*).

Security Considerations

You need to be concerned about not becoming a 3rd-party Denial of Service (DoS) source or a DoS target; other areas to take into consideration are resource consumption, memory (primary and secondary), and network bandwidth.

DoS Source

A compromised owampd server could be used to send packets toward others. The implementation ensures that sessions **cannot** be directed to random hosts in unauthenticated mode. (Only toward the OWAMP-control client.) Reasonable bandwidth limits and well protected AES keys should limit this risk.

DoS Target

Packets directed toward an owampd server can/will affect the precision of the valid test traffic. Someone might try to affect data plots by targeting hosts that do one-way measurements.

Resource Consumption

owampd has policy controls to allocate resources to appropriate users. This is done by classifying each new incoming request either by IP/netmask or using a known AES key. Each classification is associated with a set of resource limits.

Policy Recommendations

On Abilene, we attempt to be open until we can’t. We recommend that new users restrict overall bandwidth to something relatively small (most OWAMP sessions do not require much) and limit “open” tests to ensure they do not interfere with precision of other tests.

Methodological Errors

Our tests indicate a methodological error of 73 usec for the following hardware:

- Intel SCB2 motherboard
- 2x512 MB ECC registered RAM
- Intel PRO/100+ integrated NIC

This error was determined using:

- Two systems connected via cross-over cable
- Two concurrent sessions between the systems (send, recv)
- 10 packets/second
- 95% confidence level (RFC 2679)
- Old version of OWAMP; the tool should be even better now.

Error is specific to this hardware and intensity level. Basically, you should expect your results to be valid within 100 usec’s of the error reported. (The error reported represents the NTP error, but does not include this error.)

Availability

The tool, as well as the source code, is available at:
http://e2epi.internet2.edu/owamp/download.html.

Email-based discussion lists are available; go to the http://e2epi.internet2.edu/owamp/ web site and click:

- owamp-users – General discussion on the OWAMP tool
- owamp-announce – Announcements on new features/releases

Publicly-Accessible Servers

Below is a list of publicly-accessible servers. Note that this is not a complete list and more are being added when they become available. (A more up-to-date list can be derived by looking at http://e2epi.internet2.edu/pipes/pmp/pmp-dir.html.) Several institutions also run private servers.

Institution / Network	**Location**	**Information Page**
APAN	Japan	APAN PMP Info
DANTE/GEANT	Europe	GEANT PMP Info
ESnet	US Nationwide	ESnet PMP Info
Hawai'i GigaPoP/University of Hawai'i	Honolulu, HA	Hawai'i PMP Info
Internet2 / Abilene	US Nationwide	Abilene PMP Info
KISTI/KREONet2	Korea	KISTI PMP Info
MIT / Haystack Observatory	Westford, MA	Haystack PMP Info
NC-ITEC	Raleigh, NC	NC-ITEC PMP Info
NOAA Boulder Laboratories	Boulder, CO	NOAA PMP Info
NORDUnet	Sweden	NORDUnet PMP Info
Ohio State University	Columbus, OH	OSU PMP Info
RNP Measurement WG/RNP2	Brazil	RNP PMP Info
Southern Crossroads GigaPoP (SoX)	Atlanta, GA	SoX PMP Info
Swedish University Network	Sweden	Sunet PMP Info
Swiss Education and Research Network	Switzerland	SWITCH PMP Info
TWAREN	Hsinchu, Taiwan	TWAREN PMP Info

Figure 2: Publicly-Accessible owampd Servers

Internet2 E2E piPEs Project

The focus of this effort is to develop an end-to-end measurement infrastructure capable of finding network problems. The tools used by this project include the Bandwidth Test Controller (BWCTL – manages throughput tests), OWAMP (one-way latency), and NDT (last mile issues). Each of these tools has a cookbook similar to this one. They can all be accessed through http://e2epi.internet2.edu/library-list.html.

Installation Guide: Establishing an owampd Server

This section contains information on installation and configuration. More information on the tool can be found at: http://e2epi.internet2.edu/owamp/.

Components

Everything is contained in a single downloadable tar file. The file is stored on the Internet2 web site at: http://e2epi.internet2.edu/owamp/download.html.

Supported Systems

- FreeBSD 4.x, 5.x, 6.0 (64-bit)
- Linux 2.4, 2.6 (64-bit)
- Solaris 10.x
- MacOS X 10.4.5
- (Most recent versions of UNIX should work)

Requirements and Recommendations

This section covers the hardware requirements, software requirements, and recommended settings.

Hardware Requirements

- Stable System Clock
  - Temperature controlled environment
  - No power management of CPU
- No strict requirements for CPU, Memory, Bus speed
- More tasking schedules will require more capable hardware

A stable system clock is the most important feature. On Abilene, we use: an Intel SCB2 motherboard in the following configuration:

- 2 x 1.266 GHz PIII, 512 KB L2 cache, 133 MHz FSB
- 2 x 512 MB ECC registered RAM (one/slot to enable interleaving)
- 2 x Seagate 18 GB SCSI (ST318406LC) Inter Ethernet Pro
- 10/100+ (i82555) (on-motherboard)

We use systems configured like this to support a minimum of 44 concurrent streams of 10 packets/second (990 Mbps TCP flows) on a system co-located with each Abilene PoP. The 44 concurrent streams represent intra-Abilene testing. The Abilene measurement hosts that are doing tests with hosts on external networks are participating in additional streams. The specific system requirements are highly dependent upon the specific network tests. Higher intensity schedules will require more capable hardware. For more information on the network measurement computers at each Abilene PoP, see http://abilene.ucaid.edu/observatory/.

Software and System Requirements

One-way latency measurements are only generally meaningful if the clocks on the two involved systems are synchronized. (some things, like jitter, are meaningful even without synchronization) OWAMP relies on NTP (ntpd) to synchronize the system clocks needed to provide the high accuracy timing between systems. The clocks must be stable for OWAMP to be accurate; therefore NTP must be configured with stability and resilience in mind. (For more details, see the NTP cookbook at: http://e2epi.internet2.edu/library-list.html). OWAMP uses NTP-specific system calls, if they are available.

If you are working with firewalls, you will need to open the appropriate ports for communication and testing:

- TCP/8424 (Control communication)
- UDP/ephemeral (Settable using testports configuration in owampd.conf)

To do this using iptables, the additional arguments would look something like:

```
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport
4824 -j ACCEPT
-A RH-Firewall-1-INPUT -p udp -j ACCEPT
```

Recommended Settings

On Abilene, we attempt to be as open as we can, until we can't anymore. We suggest:

- Limit bandwidth
- Limit disk space available for buffering
- Allow modest amount of anonymous testing
- If you use AES keys to allow more intense tests, protect them!

General Security Concerns

As discussed earlier in this manual, the biggest issue for Abilene is: No DoS attacks! The general approach is: 1) do no harm (we don't want machines to be a source of DoS attacks but we would like them to be as available as possible and useful as possible for debugging) and 2) avoid being an attractive nuisance (obscurity lessens usefulness but do harden machines themselves).

Regarding hardening machines: don't run anything you don't have to. Keep up to date with security patches. Perhaps run a local firewall (on the machine) if it makes sense. But see if it affects your measurement results and realize that, by default, OWAMP will want to use the "ephemeral" UDP ports for testing (to a rough approximation, all those over 1024, but it varies by OS). Consider restricting logins, and where logins can occur from. If you're really good, audit programs on the machine.

OWAMP Security Concerns

These are the resources that are at risk directly from the use of OWAMP (issues the configuration must solve.)

- Limit the bandwidth that can be consumed
- Limit the memory/disk that can be consumed on the test host

Building the OWAMP Programs

To unpack, build, and install OWAMP, first grab the latest tarball at http://e2epi.internet2.edu/owamp/download.html, unpack the tar file, and use the provided configure script and make to create and install the executables:

```
% gzip -cd owamp-$VERS.tar.gz | tar xf -
% cd owamp-$VERS
% ./configure --prefix=/ami
     #  --prefix is only needed if you don't like the default
     #  (/usr/local on most systems)
% make
% make install
```

This does not install configuration files.

Partitioning Resources

To protect resources you must decide how many of those resources you are willing to have this activity use, and who you want to use it.

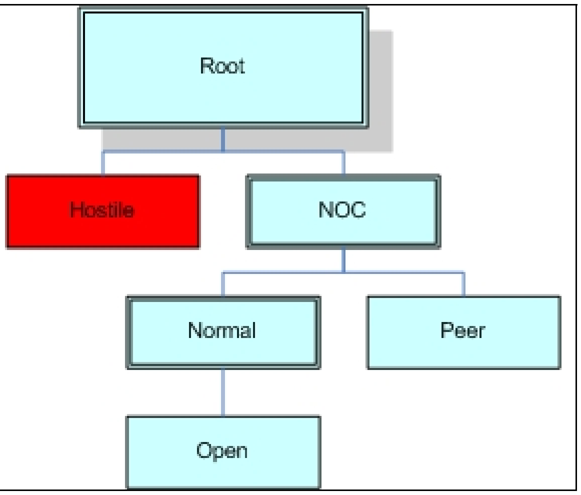
- Decide upon complete amount of resources it is acceptable for the test host to consume
- Decide how to allocate those resources among users
- How much disk space can be dedicated? Per group?
- How much bandwidth total? Per group?
- Keep system load in mind as well as network. The data accuracy will suffer if the system is too loaded.

Resources Allocated Using Hierarchical Limitclasses

OWAMP allows hierarchical limitclasses to be defined so available resources can be partitioned in a hierarchical model.

- Users are grouped into hierarchical limitclasses
- One parent-less class allowed, it defines the total amount of resources available
- When limitclasses are defined, limits of the one and only parent are inherited
- When consumable resources are requested, the limits of the limitclass and all parent limitclasses must be satisfied (memory/bandwidth)

An example of hierarchically organized limitclasses would be:

Root	Complete set of resources available	
NOC	Super-user limits	
Peer	Extended limits for peer NOC tests	
Normal	Reasonable limits for end users	
Open	Conservative limits for **anyone**	
Hostile	Used to “jail” hostile users	

You will define the hierarchy in a way that makes sense for the particular groups of users you have. (It is of course possible to define a flat space where all groups are direct children of the “root” group if your groups of users are completely unrelated.) Another probable hierarchy would be defined by creating sub-limitclasses from “normal” for users from other domains.

Classifying Connections

This was kept as simple as possible for now. There is no DNS matching of any kind. There are two methods used to classify connections.

IP/netmask

- The IP address of the client is matched against a list of IP netmask specified subnets and assigned to a limitclass based on the address of the client
- The most specific matching mask wins in the matching algorithm

This does not need to be a “real” sub-net from a routing perspective. The netmask here is only a way of expressing a range of addresses.

Username and AES key

- Client specifies a username, the server must already know the associated AES key
- AES key is used as a symmetric session key (Client and Server use the key as a shared secret)

This is basically a static symmetric session key setup. OWAMP is a fairly low-level tool and its use of authentication is fairly simplistic. The current authentication scheme was chosen because it was easy to implement and should be fairly easy to integrate into a more complete solution. (For example, PKI could be used in combination with a Diffie-Helman-style key agreement to dynamically allocate the AES session keys that are used within the OWAMP protocol.)

Configuring the owampd Server

The basic procedure to configure owampd is to create an owampd.conf and, optionally, an owampd.limits file and an owampd.keys file. These files need to be installed in the same directory that is specified with the -c option to owampd. The recommended directory is /ami/etc. (The etc directory below your install root.) There are examples of these files in the owamp-$VERS/conf subdirectory of the distribution.

Configure owampd.conf

The owampd.conf file is the configuration file for the owampd daemon. It is used to configure the basic operation of the server. For example, what addresses and ports it should listen on, where it should send error messages, and where it should save files.

The example owampd.conf file from the conf subdirectory of the distribution is fairly well annotated to explain all the available options and the owampd.conf manual page http://e2epi.internet2.edu/owamp/owampd.conf.man.html also describes all the available configuration options.

Most installations will only need to modify the following options:

datadir	Directory that holds buffered data for test sessions received by the server
vardir	Directory where owampd.pid and owampd.info files will be stored
user	Specifies the uid the owampd process will run as
group	Specifies the gid the owampd process will run as

Configure owampd.limits

The owampd.limits file is used to define the policy configuration for the owampd program. It allows the system administrator to allocate the resources in a variety of ways.

There are two parts to the policy configuration:

Authentication

Who is making the request? This can be very specific to an individual user or it can be more general in that the connection is coming from some particular network.

Authorization

Now that the connection has been generally identified, what will owampd allow it to do?

The authentication is done by assigning a limitclass to each new connection as it comes in. Authorization is accomplished by using the set of limits each limitclass has associated with it. The limits assigned to each limitclasses are hierarchical, so a connection must pass the limit restrictions of the assigned limitclass as well as all parent classes.

Within the owampd.limits file, assign lines are used to assign a limitclass to a given connection. Limit lines are used to define a limitclass and set the limits associated with that limitclass. The file is read sequentially, and it is not permitted to use a limitclass before it is defined using a limit line.

An example of limitclass definition would be:

```
# total available
limit root with \
      disk=100M, \
      bandwidth=0, \
      delete_on_fetch=on, \
      allow_open_mode=off

# Hostile
limit hostile with parent=root, \
      disk=1, \
      bandwidth=1

# NOC
limit noc with parent=root, \
      allow_open_mode=on
```

This example just shows three of the possible limitclasses from the hierarchy described above. The full set of configuration options available to limit a given limitclass are described in the owampd.limits(5) manual page (http://e2epi.internet2.edu/owamp/owampd.limits.man.html).

The following example shows how you could use IP/netmask assignments to classify connections from specific hosts:

```
# loopback
assign net ::/127 noc
assign net 127.0.0.1/32 noc
# abilene nmslan (observatory systems)
assign net 2001:468:0::/40 noc
assign net 198.32.10.0/23 noc
assign net 10.0.0.0/16 hostile
```

This example shows how any connections to the server from the loopback interface can be assigned the limits associated with the noc limitclass. Additionally, the nmslan systems are assigned to the same limitclass.

This example also illustrates how you can ensure that all connections from a given subnet are denied unless they are authenticated (See the 10.0.0.0/16 line). The *hostile* limitclass has allow_open_mode set to no. Therefore, open mode communications will not be accepted from this address range. However, users on this subnet can still attempt to use the username/AES key method of authentication. (If no communication at all is wanted with a given subnet, that functionality is better provided with a firewall application.)

Netmask assignments should not be trusted too heavily. Loopback is reasonable, and probably “local” networks, but great care should be taken before extending the model beyond that.

The following example shows how you could use username assignments to classify connections from specific users:

```
# network admins
              assign user joe root
              assign user jim root
              assign user bob root

# measurement geeks
      assign user boote noc
```

The owampd server needs to be able to authenticate that a given user is who they say they are. This is done using a 128-bit shared key. The username to key association is made known to owampd using the owampd.keys file described below. The user must have an entry in the owampd.keys file to be used in the owampd.limits file. The owampd process will refuse to start if a user is listed in the owampd.limits file if they do not have a key associated in the owampd.keys file.

Configure owampd.keys

The owampd.keys file is used to hold the identity/AES keys pairs needed for owampd to authenticate users. The format of this file is described in the aespasswd(1) manual page. The location of the owampd.keys file is controlled by the -c option to owampd.

owampd uses symmetric AES keys for authentication. Therefore, the owping client must have access to the exact same AES key for authentication by AES to work. Most likely, the user will simply just know the passphrase that generated the AES key in the first place. Additionally, it is important that the system administrator and end user ensure the key is not compromised.

If the owping client is able to authenticate using the identity and AES key presented, owampd will use the directives found in the owampd.limits file to map policy restrictions to this connection.

Username and AES Key Rules

- Usernames are limited to 16 characters
- AES key is a 128 bit session key
- AES key is not encrypted in the keys file, use UNIX permissions to protect it
- Can use a pass phrase to generate the AES key
- Use aespasswd to add pass phrase generated keys into the keys file
- Client: application prompts user for pass phrase

The normal UNIX protection method would be to run the daemon with specific user or group permissions that allow it to read the keys file, but limit the users that have access to it. An example key file might look like:

```
joe   a0167ac6101b360d2f4dd164abba2337
bob   2dc36fc4807894cdfbe180b71d2b4a0f
```

```
sam   3fc763fb270ce6ba6e928bd10d4977d3
```

This is simply a username associated with a hex encoded 128-bit value. By far the easiest way to create and maintain the owampd.keys file is to use the aespasswd application.

aespasswd

This is similar to htpasswd (apache web server); you specify an identity to be added to a key file and are prompted for a passphrase. This is a convenience since users don't remember 128-bit quantities very well. It is used to convert a passphrase into a 128-bit hex key in a portable way across multiple architectures. For more information, see: http://e2epi.internet2.edu/owamp/aespasswd.man.html. This same application is used to manage key files for OWAMP and BWCTL, so take care which of the files you are editing.

To create a new key file use the '-n' option:

```
% aespasswd -n -f owampd.keys demo
```

Additional usernames can be added by omitting the '-n':

```
% aespasswd -f bwctld.keys joe
```

For more complete information, see
http://e2epi.internet2.edu/owamp/owampd.keys.man.html,
http://e2epi.internet2.edu/owamp/aespasswd.man.html, and
http://e2epi.internet2.edu/owamp/owampd.limits.man.html.

Running One-Way Latency Measurements using OWAMP

Starting owampd

Start the daemon in foreground mode during testing:

```
% /ami/bin/owampd -c /ami/etc –Z
```

Many of the command-line options are used to override the config file parameters. The '-c' option should always be used, unless the daemon is started from the config directory. For more information, see: http://e2epi.internet2.edu/owamp/owampd.man.html.

Testing (owping)

First, try a simple localhost test and watch the output from owampd. This is a good test because there are no clock difference issues to the localhost:

```
% /ami/bin/owping localhost
```

Running a test to localhost first helps verify a working configuration. Use the IP address for the interface instead of the localhost interface to test for host based firewall problems.

Now, try a test to one of our hosts. (This host is only guaranteed to be available during a Network Performance Workshop – we tend to try new things on here from time to time.)

```
% /ami/bin/owping nmsy-aami.abilene.ucaid.edu (Internet2 test host)
```

Now, try and test to other hosts. A good candidate would be a new deployment in one of your peer networks or alternatively something off of the global PMP list (http://e2epi.internet2.edu/pipes/pmp/pmp-dir.html).

```
%/ami/bin/owping otherhost
```

For more information, see: http://e2epi.internet2.edu/owamp/owping.man.html.

Once you have verified your install, it is recommended that you install an rc script that automatically starts owampd when the host is booted. You should not be using the –Z option of the daemon in general, but it is a very good debugging tool.

Troubleshooting

The most frequently seen problems are:

1. No Control Connection
   a. No daemon running
   b. Firewall - open control port (4824)
2. Control connection denied
   a. Improper configuration
   b. Invalid credentials
3. Control connection works, all test packets lost
   a. Firewall – open ephemeral UDP ports or use testports in owampd.conf to make owampd use a range of opened ports
   b. Clock offset – If the clock offset between two systems is more than *timeout* all packets will be declared lost. (See the –L option of owping for a definition of *timeout*.) Use a large value for –L to test this. (Larger than any possible clock differences between the systems.)
4. Negative delay values for results
   a. Clock offset – Use NTP to see the clock differences between the systems if possible.