INTERNET2®

www.internet2.edu

Bandwidth Test Controller: 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 a Network Performance Workshop; for more information on these workshops (upcoming and past), see: http://e2epi.internet2.edu/network-perf-wk/.

The Bandwidth Test Controller (BWCTL) tool was developed for use with E2E piPEs and the Abilene Measurement Infrastructure. More information on this tool can be found at: http://e2epi.internet2.edu/bwctl/.

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 Bandwidth Test Controller servers at your location.

An Overview of the Bandwidth Test Controller

This section has information on the motivation for the tool, benefits, functions and features, and implementation. It also includes a flow chart for the tool and information on the availability of the code and publicly-accessible servers.

Motivation

Users want to verify available bandwidth from their site to another site. Currently, Iperf is one of the most commonly used tools for throughput tests. The major objective for the Bandwith Test Controller was to create a resource allocation and scheduling daemon for the arbitration of Iperf tests so that regularly-scheduled and on-demand tests could coexist.

Methodology

The approach taken by the Bandwidth Test Controller was to verify available bandwidth from each endpoint to points in the middle to determine the problem area. This is, arguably, one of the best ways to determine if an application will work because it is doing the throughput test end-to-end exactly as the application would be doing it.

Benefits

The usual manner this type of testing was performed, before the development of the Bandwidth Test Controller, was to run Iperf or a similar tool on two endpoints and hosts on intermediate paths. [If NOC operators had a dime for every time someone asked them to start an Iperf server somewhere…]

The Bandwidth Test Controller schedules tests in the presence of scarce resources, which means that you never have to worry about tests running on top of each other (skewing the results). It allows authorized testers to request and run tests at the first free testing slot without hands on support from NOC staff. Because the NOC can control how frequent and long a test may be run, they can allow more end-users the authority to run tests on their own – instead of having to give users accounts on machines.
There is a good example of how this assisted a scientific community: MIT Haystack researchers setup Bandwidth Test Controller servers at several of the telescope locations from which they receive data flows so that when troubles arose, they could immediately begin testing to locate the problem – instead of having to wait for NOC staff at the remote locations to assist them. (See http://e2epi.internet2.edu/case-studies/VLBI/cs-index.html for more details.)

Implementation

There are pluggable API interfaces to allow different authentication and policy functions to be incorporated at a later time.
Iperf is the "tester" because it is well-known and widely used. However, Iperf has some integration problems:

- Iperf server initialization (port number allocation)
- Iperf error conditions
- End of session

These problems will likely show up in your test results just as they do when you normally run Iperf. The difference is that, when you run Iperf by hand and cancel the test (by typing Cntrl-C), you are not surprised when you don't have results from the test. And almost anyone who has used Iperf has seen particular sessions that take much longer than expected. Unfortunately, because the Bandwidth Test Controller is scheduling tests and needs to make sure the "next" test can run, it must kill those particular sessions. Since Iperf is not closely integrated into the Bandwidth Test Controller, intermediate results are unavailable for these sessions. Future releases of the Bandwidth Test Controller are likely to be more integrated with the actual test engine (Iperf).

Along these same lines, the Bandwidth Test Controller has specific difficulties for scheduling UDP:

- Iperf doesn't always send at the requested rate
- Iperf sender hangs intermittently
- End of session is difficult to detect, which is problematic for a "scheduled" timeslot

Functions and Features

The tool contains two applications: bwctld (daemon) and bwctl (client).

Client

The Bandwidth Test Controller client application, bwctl, makes requests to both endpoints of a test. Communication can be open or authenticated. Requests include a request for a time slot, as well as a full parameterization of the test.

The Bandwidth Test Controller allows third-party requests between two unrelated servers using potentially unique authentication to each side. For bandwidth tests from the current client, the Bandwidth Test Controller offers a server-less option. If no server is available on the localhost, the client handles the test endpoint. The Bandwidth Test Controller offers most of the same command line options as does Iperf; some options are limited or just not implemented. The command line options are as similar to Iperf as possible. (Sometimes infuriatingly so!)

Daemon

This is where the policy is implemented; bwctld is a traditional accept/fork style daemon with the parent process also listening for resource requests from child processes. The bwctld on each test host:

- Accepts requests for “Iperf” tests including time slot and parameters for test
- Responds with a tentative reservation or a denied message (Note: Reservations by a client must be confirmed with a “start session” message)
- Brokers resources
- Runs tests
- Returns results to both sides of the test (and to the requesting party if it was a third party request).

Scheduling

A time slot is simply a time-dependant resource that needs to be allocated just like any other resource. It, therefore, follows the normal resource allocation model described below. The Bandwidth Test Controller scheduling currently only allows a single test to happen at a time.

Resource Allocation

The parent bwctld keeps track of current resource utilization needed to implement policy. Each connection is “classified” (authentication); each classification is hierarchical and has an associated set of hierarchical limits:

- Connection policy (allow_open_mode)
- Bandwidth (allow_tcp,allow_udp,bandwidth)
- Scheduling (duration,event_horizon,pending)

Architecture

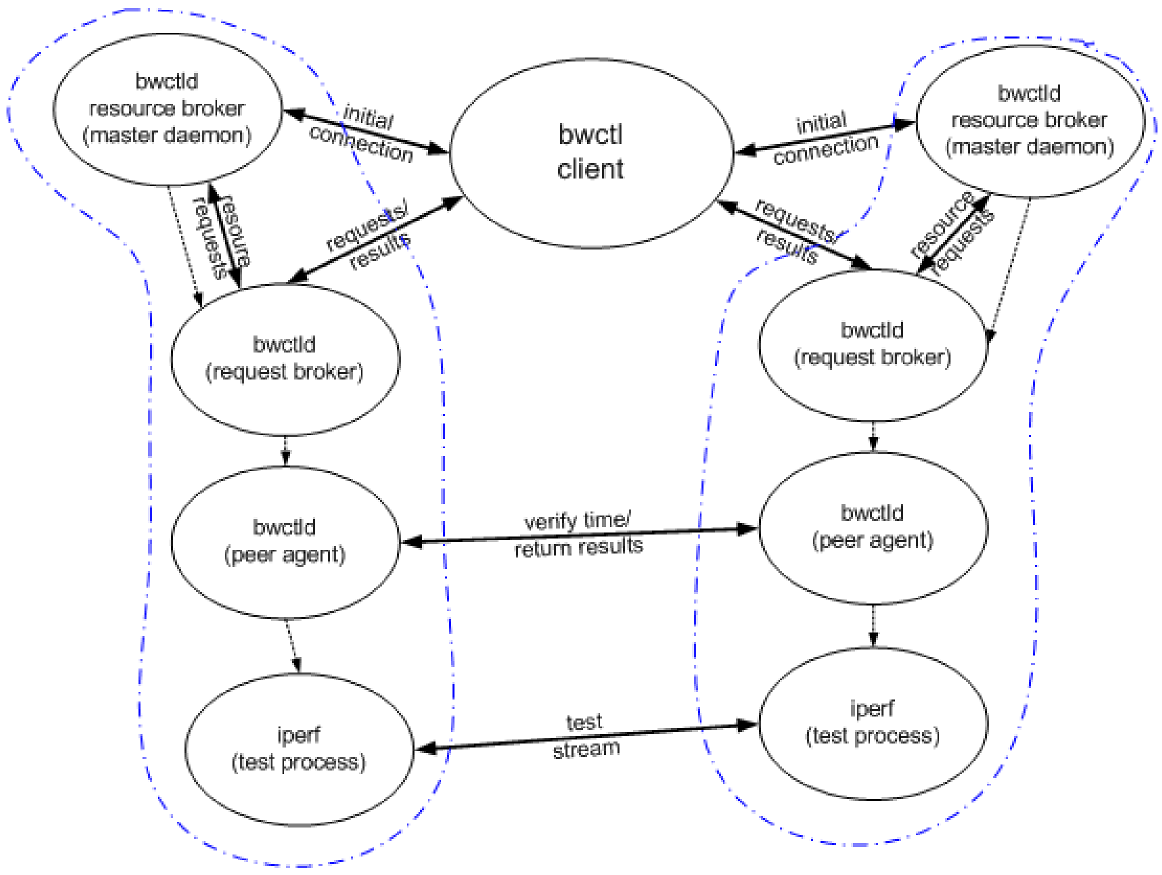


Figure 1: Control Flow for BWCTL Architecture

Overview

bwctld 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 bwctld processes. Once a connection comes in, bwctld forks a child process to handle that request.

In the event that the local host is one of the endpoints and there is no local bwctld running, bwctl will spawn additional processes to execute the local side of the test directly (basically, emulating the functions of the daemon).

Authentication and Authorization

The child (request broker) process deals with all encryption and communication issues with the client, as well as dealing with 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 UDP 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 (resource broker) process for the resources and time period requested by the client. If the resource broker has the resources available (including the open time period), it grants the request.

Scheduling

The client always requests the time period for the test. It makes a request to the request broker process on each endpoint of the test in turn. The time period is requested by specifying two timestamps. The earliest time and the latest time the client is willing to allow the test. The request broker either returns a tentative reservation with the first open time that will fulfill the parameters of the test or a ‘request denied’ message. The client can then use the tentative reservation time as the earliest time in the request to the request broker on the other host. Eventually, an agreed upon time will be reached or the latest time will be reached. If the request broker cannot fulfill the request before the latest time specified in the request, it will return a ‘server too busy’ message. If the client gets the same valid time slot from both servers, it must then confirm the reservation with a ‘start session’ message. The ‘start session’ message must be received before a configurable timeout period and before the reservation time, or the server will disallow the reservation.

Execution of the Test

Once the client sends the ‘start session’ message, the request broker forks off a peer agent that is responsible for verifying the time offset to the other endpoint of the test and initializing the communication socket that will be used to trade results of the test.

If the test endpoint systems have a reasonably close idea of the time, and they can communicate, the peer agent forks off the test process. The test process waits until the scheduled start time and then executes Iperf with the correct command-line parameters.

General Requirements

- Iperf version 2.0
- NTP (ntpd) synchronized clock on the local system
- NTP system calls
- To get good results, the end hosts will almost certainly need to be well-tuned

Operational Concerns

Concerns include time and firewalls; time issues include:

- NTP only has a good idea of the time error if it is configured correctly. If it is not configured correctly, the Bandwidth Test Controller will be forced to terminate tests before they are completed, making the results unavailable. For the NTP algorithms to work correctly, NTP ***must*** be configured with no fewer than four (4) clocks. (See http://twiki.ntp.org/bin/view/Support/SelectingOffsiteNTPServers for more details.)

Firewall issues include:

- TCP ports need to be opened for control communication. Both from the client to the server and for peer connections between the servers. (Details are provided in the installation guide.)

Policy Issues

The policy issues can best be grouped into two categories. First, it is important to ensure that the bwctld 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 bwctld server and the data produced is protected (Security Considerations). Second, controls need to be in place to allow the available resources to be partitioned among the valid users of the server (Resource Consumption).

Security Considerations

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

DoS

Basically, avoid being an attractive nuisance. Unfortunately, obscurity lessens usefulness so this can only be taken so far. If these machines are not generally available to valid users, they are not useful. But we encourage you to harden the machines, themselves.

Source

Imagine a large number of compromised bwctld servers being used to direct traffic! A compromised bwctld server could be used to send packets toward others. The implementation ensures that sessions can**not** be directed to random hosts in unauthenticated mode. (Only toward the Bandwidth Test Controller-control client.)

Target

Someone might attempt to affect statistics web pages to see how much impact they can generate. Packets directed toward a bwctld server can and will affect the results of the valid test traffic.

Resource Consumption

Two controls on resource consumption are time slots and network bandwidth; bwctld 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 UDP completely, or at least limit the bandwidth. We require AES key authentication for all users.

Availability

The tool and source code is available at:
http://e2epi.internet2.edu/bwctl/download.html.

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

- bwctl-users – General discussion on the Bandwidth Test Controller tool
- bwctl-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 One-Way Ping 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 (throughput), One-Way Ping (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 a Bandwidth Test Control Server

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

Components

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

- FreeBSD 4.x, 5.x
- Linux 2.4, 2.6
- (Most recent versions of UNIX should work)

Requirements and Recommendations

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

Hardware Requirements

- No strict requirements for CPU, Memory, Bus speed, NIC
- Your hardware will dictate the possible intensity of tests you can perform
    - More tasking tests require more capable hardware

On Abilene, we use: an Intel SCB2 motherboard with:

- 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)
- SysConnect Gigabit Ethernet SK-9843 SX

We use systems configured like this to support 990 Mbps TCP flows between systems co-located with each Abilene PoP. The specific system requirements you will want are highly dependent upon the specific Iperf tests you want to perform.
See http://abilene.ucaid.edu/observatory/ for more information on the network measurement computers co-located at each Abilene PoP.

Software Requirements

- Iperf versions 1.7.0 or 2.0
- NTP (ntpd) synchronized clock on the local system
- Firewalls: leave lots of ports for communication and testing
- End hosts must be tuned! See:
  http://www.psc.edu/networking/perf_tune.htmlhttp://www-didc.lbl.gov/TCP-tuning/buffers.html

NTP may be a surprising requirement but scheduling the tests without wasting too much time between tests requires a reasonable estimate for the accuracy of the local clock.

Network Requirements

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

- TCP/8423 (Control communication – client to server)
- TCP/ephemeral (Control communication – server to server: Specific range settable using peerports in bwctld.conf)
- TCP/5001 (Iperf testing port – Settable to a range using testports in bwctld.conf)
- UDP/5001 (Iperf testing port – Settable to a range using testports in bwctld.conf)

Recommended Settings

On Abilene, we attempt to be as open as we can, until we can't anymore. For the Bandwidth Test Controller we suggest:

- Limit tests to TCP only (UDP is usually only useful in cases where TCP is not working well, so TCP is a reasonable place to start.)
- Require authenticated communication for all tests (AES Keys)
- Protect the AES Keys. They are clear-text passwords!

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 the 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, the Bandwidth Test Controller will want to use the "ephemeral" TCP ports for peer connections (to a rough approximation, all those over 1024, but it varies by OS). Consider restricting logins and where logins can occur. If you're really good, audit programs on the machine.

Bandwidth Test Controller Security Concerns

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

- Limit the bandwidth that can be consumed
- Limit the type of tests that can be done (TCP/UDP)

Building the Bandwidth Test Controller

To unpack, build, and install the Bandwidth Test Controller, grab the latest tarball at **http://e2epi.internet2.edu/bwctl/download.html**, unpack the tar file, and use the provided configure script and make to create and install the executables:

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

Note: 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.

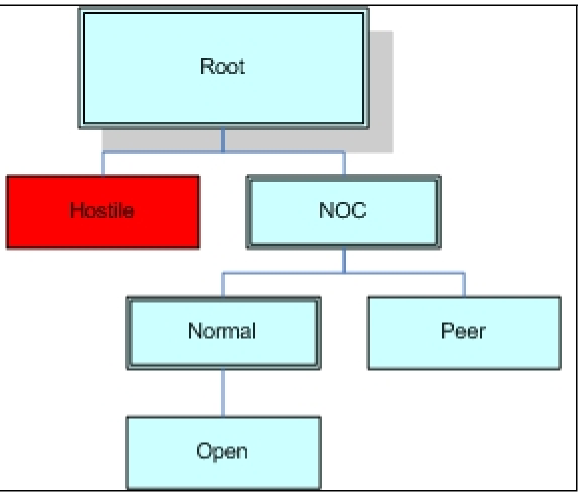
- Decide upon the complete amount of resources it is acceptable for the test host to consume
- Decide how to allocate those resources among users
- How much of the available time slots should be dedicated to each group of users?
- How much bandwidth total? Per group?
- Keep system load in mind as well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

Allocating Resources Using Hierarchical Limitclasses

The Bandwidth Test Controller 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/timeslots)

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. The Bandwidth Test Controller is a fairly low-level tool and its use of authentication primitives 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 then used within the Bandwidth Test Controller protocol.)

Configuring the Bandwidth Test Controller Server

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

Configure bwctld.conf

The bwctld.conf file is the configuration file for the bwcltd daemon. It is used to configure basic operation of the server such as server listening port, the path for Iperf, and error logging.

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

Most installations will only need to modify the following options:

vardir	Directory where bwctld.pid file is stored
user	Specifies the uid the bwctld process will run as
group	Specifies the gid the bwctld process will run as

Configure bwctld.limits

The bwctld.limits file is used to configure the policy limits for the daemon. 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 bwctld 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 limitclasse are hierarchical, so a connection must pass the limit restrictions of the assigned limitclass as well as all parent classes.

Within the bwctld.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 each 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 \
      bandwidth=900m, \
      duration=0, \
      allow_tcp=on, \
      allow_udp=on, \
      allow_open_mode=off

# Hostile
limit hostile with parent=root, \
      bandwidth=1 \
      allow_tcp=off, \
      allow_udp=off

# 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 bwctld.limits(5) manual page (http://e2epi.internet2.edu/bwclt/bwcltd.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. It is not possible to run a bandwidth test using bwctl to the local host because you would need an open schedule slot for both the receiver side of the test and the sender side of the test during the same time period and bwctld limits the number of open schedule slots to one at a time. However, the loopback/localhost lines are important if you want to bypass AES authentication for the local authentication when you are running a local bwcltd. (Remember, if there is no local bwctld running, bwctl will run the test directly itself.)

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

Configure bwctld.keys

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

bwctld uses symmetric AES keys for authentication. Therefore, the bwctl client will have to 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 bwctl client is able to authenticate using the identity and AES key presented, bwctld will use the directives found in the bwctld.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 bwctld.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/bwctl/aespasswd.man.html. This same application is used to manage key files for bwctl 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 bwctld.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/bwctl/bwctld.keys.man.html,
http://e2epi.internet2.edu/bwctl/aespasswd.man.html, and
http://e2epi.internet2.edu/bwctl/bwctld.limits.man.html.

Running Bandwidth Test Controller

There must be two hosts involved to test the Bandwidth Test Controller. (You can't schedule both endpoints of a test on a single host; one of the main reasons to use the Bandwidth Test Controller is to ensure only one test happens at a time.) It is, however, possible to run one of the endpoints of the test directly from the client.

Testing the Bandwidth Test Controller Client (bwctl)

First, try a simple test from a client system to one of the Internet2 hosts. (This host is only guaranteed to be available during the workshop – we tend to try new things on here from time to time.) You will need to already have a userid/AESkey configured into the Internet2 bwctld for this to work.

```
% /ami/bin/bwctl –s nmsx-aami.abilene.ucaid.edu A AESKEY jimbob
```

bwctl will prompt the user jimbob for the passphrase that was used to generate the AES key that is being used as the shared secret.

Testing the Bandwidth Test Controller Daemon (bwctld)

The next step is to run a local daemon. Start the daemon, in foreground, during testing by using the '-Z' option:

```
% /usr/local/bin/bwctld -c /usr/local/etc –Z
```

Many of the command-line options are used to override the config file parameters. For example, the '-c' option will almost always be used, unless the daemon is started from the config directory.

For more information, see: http://e2epi.internet2.edu/bwctl/bwctld.man.html.

Two Hosts, Two Daemons, One Client

The next step is to do the same local host to remote host as before, but now that you have a local bwctld running it will manage the local endpoint of the test instead of the client application. (You should use another window to run this test so you can see the output form the bwctld process while the client is running.)

```
% /ami/bin/bwctl –s nmsx-aami.abilene.ucaid.edu A AESKEY jimbob
```

If you did not bypass authentication using localhost IP masks, you will need to specify the localhost side of the test explicitly using the –c option, or, if you used the same userid and AESKEY for your local host as for the Abilene host, you can use the –A flag to specify the authentication for both sides of the test instead of appending the authentication information on the end of the –s flag as above. (This is more fully explained later.)

Other Hosts

The next step is to test between two of your own hosts or between your hosts and other non-Abilene hosts.

Testing Authentication Options

The fact that the bwctl client application supports multiple keys is useful because you probably don't want to share your "internal" key with external organizations since it is a symmetric key.

Within a single authentication domain (same AES key):

```
        bwctl -A AE AESKEY myname -s hostA -c hostB
```

Between different authentication domains you append the authentication information on the end of the –s/-c options:

```
        bwctl -s hostA AE AESKEY myname  -c hostB AE AESKEY othername
```

Troubleshooting

The most frequently-seen problems are:

1. No control connection
   - No daemon running
   - Firewall—open control port (4823)
2. Control connection denied
   - Improper configuration
   - Invalid credentials
3. Initial control connection works - peer connection fails
   - Firewall—set peerports and open that range of TCP ports
4. Scheduling problems
   - No open slots within the window—use the -L flag to wait for a later slot
   - NTP setup incorrectly—set ‘syncfuzz’
5. Iperf connections fail
   - Firewall – set iperfports and open that range of TCP/UDP ports
6. Iperf results are bad—or “no data”
   - Initial window is too large, iperf can’t finish in given time. Probably indicates a real network problem, but it is not possible to get results. Try using ‘-i’ flag to get intermediate data.
   - Iperf sometimes has strange interactions in it’s read loop. Just try again.