# OSG Debugging Documentation

Edited By: J. Zurawski – Internet2, S. McKee – University of Michigan

February 4th 2013

# Abstract

Scientific progress relies on intricate systems of computers and the networks that connect them. Often it is the case that scientific data is gathered via a well defined process, information is digitized, stored, transmitted, and processed by members of large and distributed collaborations. The Open Science Grid advances science through the concept of distributed computing – the process for sharing resources through a unified software framework focused on the common tasks of data movement, processing, and analysis.

Networks are an integral part of the distributed computing process. Similar to the computational and storage resources, it is crucial that all networking components, on the complete end-to-end path, are functional and free of physical and logical flaws. A rich set of measurement and monitoring tools exists to provide network operations staff and end users a window into the functionality of networks, despite the fact that these actors do not have direct control over the complete path their data may travel.

This document discusses common measurement and monitoring tools available to the OSG community, and strategies to deploy, use, and interpret the results they produce. The end goal is to give end users more insight into network behavior, and assist local and remote networking staffs as they correct damaging performance problems that will impact the scientific process.

# Introduction

The process of science is often complicated when viewed as a complete system. At the core of any project, there is a mechanism to observe or simulate some system, and produce meaningful results that will be interpreted and scrutinized between experimentation runs. The machinery that surrounds this process can be as benign as simple cameras, or as complex as the Large Hadron Collider and its associated experiments. Other common components include ways to digitize, store, process, and share the end result of experimentation – often done using computational systems.

Computation falls into 3 broad classifications, all of which are required to implement the paradigm of scientific computing:

* ***Storage*** – Readable and writable physical medium used for temporary or long term residency of gathered data.
* ***Processing*** – Specially designed hardware and software that iterates over collected data sets looking for pre-defined triggers and results.
* ***Networking*** – Interconnecting hardware and software used to facilitate communication between storage and processing components both on a local, and fully distributed basis.

When fully realized, even a small facility can contribute a great amount of resources to the overall goal of scientific advancement. In practice it may be the case that a lab consisting of a single researcher can pull data sets from a centralized location, perform carefully selected segments of an entire set of analysis that is required, and return any relevant results as they are discovered. When used in an inductive fashion, one can imagine the overall throughout that a VO such as the LHC project is able to attain through 100s of distributed facilities and 1000s of collaborating researchers.

Complexity is present as we travel down the individual technology items in the above scenario. Often it is the case that ideal performance is hard to attain due to the intertwined nature of the mechanisms involved. For example, data must be written and read from physical medium. Often this step is slower due to the mechanical nature of the process, and struggles to keep up with faster technologies such as processing or transmission on network infrastructure. Equally, it may be the case that a flaw in the network infrastructure, such as a failing component, can introduce data loss that must be compensated for through retransmission. Retransmission implies additional work for storage and processing components that must waste resources to overcome a fixable, but often unnoticed, problem.

Network performance monitoring is a relatively unseen, but still extremely necessary, practice. This statement is true due to the nature of network use through application software and communication protocols. Application developers wish to unburden the user with details about “how” data may be moved between facilities. Care is taken to design applications in such a way that the user is simply presented with options related to a source and destination only, and little or no insight into the path taken or the current conditions that may be present. The aforementioned situation where a failing component institutes data loss results in only one symptom to the end user: lower than expected throughput. Many users may not notice, or have become complacent, with low performance situations. Some may write this off as “the network is slow”, or perhaps will not notice at all due to experiences with home connections that are often 2 orders of magnitude slower than what is possible in a typical academic environment.

Software exists to monitor network performance in many different ways. For example it is possible take a measure of network throughput, and simulate the behavior of a file transfer application. It is also possible to observe network stability (e.g. jitter) over time to simulate video or audio transmission. These basic observations are powerful when used both in a local environment, as well as on an end-to-end basis. In either case, software must be deployed and available to the community on points of interest: specifically on the local and shared network infrastructure distributed around the world. perfSONAR is a software framework that simplifies network debugging activities by making it easier to deploy measurement tools, and facilities the sharing of results. It is currently deployed on many communal networking resources in the R&E community, including backbones, regional networks, campuses, and individual laboratories.

Once perfSONAR is deployed, it becomes possible to troubleshoot situations that result in low throughout for the end user in a straightforward manner. It is important to note that when something like this occurs, it is not the sole responsibility of the end user to debug and solve a networking problem; rather it is their duty to report the problem and provide as much information as possible to local or remote network staffs so they may learn about the issue, and interpret the results so as to lead to quick and efficient problem resolution. Locating this staff may be challenging, but many organizations maintain a dedicated Network Operations Center (NOC) whose staff are ready to accept trouble reports and act in an appropriate manner. Section 8 details locations you may turn to for additional support.

This document will expand upon these topics in the remaining sections, and conclude with information where additional resources beyond a simple introduction to these topics can be found.

# The Scientific Networking Process

There is a rich ecosystem of components available for monitoring and managing the scientific networking process. This myriad of hardware and software must work together to complete the overall goal of interpreting gathered or simulated observations. Each component we will discuss has the ability to be installed, operated, and maintained in different ways. Individual brands or versions are not important, but the overall idea of each will be explored.

## Hosts

Server or “host” hardware and software can be used in many different ways. Often it is the case that these components serve as computational resources for processing data, or provide access to underlying data stored on physical media. It may also be the case that software designed to “glue” components of a framework together (e.g. processing mail, authenticating users, providing mappings between names and addresses) is installed on a dedicated or shared host resource.

Hosts must contain an operating system: software designed to control and maintain the underlying hardware such as storage media, network interface cards, processors, and other peripheral devices. Operating system hardware can vary in functionality; completely interactive systems such as those found on laptops can be more pleasing for a human user vs. that of a no frills batch processing system designed to only iterate over scientific data. The choice of operating system will vary from use case to use case.

The footing provided by the hardware and operating system serves as the basis for the remainder of the components in this discussion.

## Protocols

Protocols are software algorithms implemented on hosts and networking components, and are used to facilitate communication strategies. Protocols are constructed in a “layered” fashion, and are designed to handle certain aspects of the overall communication plan. For instance a protocol may be used to communicate between two network devices, and may involve the use of different patterns of electrical or optical signal. On top of this basic system of signals we may construct a different protocol that is focused on communication between end hosts, and is able to break up the notion of a user’s files into small chunks so they can be sent reliably end-to-end.

Protocols evolve with the underlying technology, and often can be tuned to specific use cases. One such protocol, Transmission Control Protocol (TCP), is widely used in applications that many users are familiar with such as web browsing, mail transfer, and file exchange. Early incarnations of TCP were designed for the networks of the 1980s; slower, less reliable, constructions than what is present in the networks of the 2010s. As such TCP must be instructed, via configuration on a host or network devices operating environment, that it should behave in a different and more efficient manner.

With the protocol in place, we can now begin to discuss applications that are able to use the network to communicate in an automated fashion.

## Applications

Applications are specific use cases, programmed as software, and made available to end users via a host’s operating environment. There are numerous applications we use on a daily basis – web browsers to fetch remote content, word processing applications to type papers, mail and instant messaging clients to exchange information in near real time. Scientific applications normally focus on performing a single task (e.g. end-to-end data movement, visualization, data transformation, data analysis) on either a local or distributed basis.

In the case of distribution, it becomes necessary to interact with the underlying network through the use of a protocol. File transfer is a specialized application that takes is interested in either sending a local file to a remote location, or retrieving remote data to bring locally. In either case the application must broker with a protocol, such as TCP, that is available on both ends of the transfer. Through a series of API calls information is segmented into transmission chunks and sent reliably though the communication medium. Most of this is handled transparently from the user’s perspective, and as such they are not given much in the way of feedback beyond a pass or failure, and some notion of how long it took.

Understanding more about the network can be enlightening exercise for users who are unaware of the complexity and span of components that are required for operation. This will be discussed in the remaining sections.

## Lab Local Area Network

The first step in the networking tree is often the interconnections between components local to the user. This may consist of the storage and processing nodes in a single rack or data center used for scientific processing, connected via technology consistent with the tasks they are performing. Cluster nodes may use a high speed interconnect such as InfiniBand; servers may also just use typical Ethernet at 1Gbps or 10Gbps.

In either case, there will be dedicated network equipment with the task of aggregating and controlling traffic flow to the local devices, and serving as an uplink to the next network in the chain (the campus). Monitoring and management of this local environment is a good idea, either through passive means such as using the SNMP system, or active tests that check the health of transfers on a local basis.

## Campus Local Area Network

The first hop beyond a laboratory environment is a network maintained by campus/local support staff. It is often the case that this group is maintaining the infrastructure for the use of ***all*** end users, and as such will design and maintain things to preserve uniform functionality and performance.

Campus networks are an even larger ecosystem of devices given the area they may cover. It may be the case that the network in the previous section is behind several devices before it has a clean path to the outside world. It may also be the case that traffic aggregation is extremely high, and congestion becomes a factor during certain parts of the day or times of the year. These nuances make local performance monitoring crucial for operational soundness.

This group is also the first contact that should be exercised in the event of a network performance problem. While they may not be able to answer for the status of the entire path, they can escalate the problem to the regional or backbone support staff as needed.

## Regional Network

A regional network provider aggregates the networks of numerous campuses in a state, country, or pre-defined region. Examples include provider for states of the US (e.g. KanREN), countries (SWITCH, the network of Switzerland), or collaboration between parties without a political boundary (the SOX regional network in the United States). Regional networks may cover a large geographical area, but often have less equipment than a campus. The role of a regional aggregator is to take connections (large and small) and condense them into long-haul links that will uplink to a backbone or exchange point as a next step.

Regional operations teams have similar performance concerns to that of a campus network. The aggregation point of several networks can be a critical component, and one of the more likely places that a failing piece of equipment or congestion can impact downstream network users. Monitoring local and remote components (e.g. maintaining active testing between networks) critical.

Regional support teams can be likely candidates for assistance on performance problems, but users are reminded to discuss options with their local staff first before engaging with these groups.

## R&E Backbone

An R&E backbone consists of an aggregation of numerous regional providers. Capacity must reflect the number and expectations of this group of customers, and often is orders of magnitude higher than other networks that are downstream.

As an aggregation point, normally spread over a very large geographical area, traffic flows will be numerous, of mixtures of size and duration, and be destined for diverse destinations domestically and internationally. Each Point of Presence (PoP) could have a large number of customers integrating, and thus increases the chance of an issue local to this device.

As a service to customers, the R&E backbone should consider making test instances available to help bisect and debug challenging problems that may cross the domain. Backbone support teams are also well trained and have knowledge of performance monitoring. Some providers such as ESnet and Internet2 have dedicated staffs just to support the troubleshooting of network problems for customers. An end user is encouraged to seek out these resources, as well as those that are local, when debugging a problem impacting scientific work.

## Exchange Point

An exchange point is normally a location where multiple backbone networks and international transit links (e.g. trans-oceanic links) combine and transit to other domains. An exchange point is a special case of an aggregation network like a regional in that policies may be different depending on the membership structure.

International exchange points suffer from the aforementioned problems of traffic aggregation wherein congestion or equipment failure will have a severe impact on all traffic. Monitoring these devices is crucial, as in other use cases.

# Actor & Agent Definitions

There are many actors involved in the process of network management and debugging. We will highlight three here, as they represent the most critical members of the support team that OSG has available when problems are discovered.

## End User

The end user is understood to be the primary user and beneficiary of OSG software to process and operate on scientific data sets. The sophistication of this end user is assumed to be beginner to average in matters related to system and network administration. In general we assume they are knowledgeable enough to install and maintain OSG software, and connect devices to the networking infrastructure.

This actor is assumed to be the primary user of the perfSONAR end user tools, packaged in the OSG VDT distribution. These tools are meant to be run from a system to upstream test machines provided by the campus, regional, or backbone network.

## Local Administrator

The local administrator can be campus support staff responsible for the health of servers or network devices across the greater campus ecosystem. Their primary responsibility it to ensure uptime of the network for all users, as well as assist in debugging specific problems caused by performance impacting problems on a local basis.

This actor may not be able to directly address problems on a regional, national, or international basis but can serve as a liaison with individuals within those stewardship organizations.

## Remote Administrator

A remote administrator can be regional, backbone, or exchange point staff responsible for the health of remote networking resources. It is often the case that these individuals may not be aware of a specific use case between remote campuses, but could answer questions about the current health and status of the network they control.

These individuals are assumed to be knowledgeable about performance tools, and can work with local staff as needed to make test points available to assist in debugging.

# Local Preparations

A first step to any OSG software installation to support scientific activity is preparation of the local environment. Given the considerations denoted in the previous sections, we will discuss 3 specific preparation activities:

* End System Operating System and Protocol Tuning
* Network Architecture Adjustments
* Network Configuration Tuning

Each of these steps is considered to be most relevant to the laboratory local environment, although some should be considered for the campus as well. It is assumed that the end user actor, with the help of local administrators, can make these changes.

## End System Tuning

Computer systems are similar to automobiles in that its possible to “tune” certain internal aspects and achieve higher performance when using the network. The operating system and associated protocols like TCP make these changes rather simple to implement. In general there are several options worth considering:

* Network interface cards have an adjustable size for their packet queues
* Kernel buffers can be increased to support long distance transfers
* The TCP congestion control algorithm can be changed

ESnet has made a web based resource available to assist with the task of host tuning, it can be found here:

<http://fasterdata.es.net/host-tuning/>

## Network Architecture

Architectural decisions are often involved and will involve the input of local support staff. In general laboratory architecture should be robust in the following manner:

* Multiple uplinks to the campus network to provide capacity and resiliency
* A limited amount of “fan in” (e.g. number of connections) into a given access switch. It is recommended that as the fan in grows, multiple switches be employed to manage connectivity and congestion
* Elimination of firewalls from the path. Security can be implemented by host-based firewalls that restrict ports as well as Access Control Lists (ACLs) to white list sites that are communicated with. Firewall devices have been known to severely impact traffic for bandwidth intensive applications.
* Choice of device manufacturer that allows for out of band management and monitoring (e.g. SNMP) of devices
* Choice of device manufacturer that allows for per-interface tuning of memory buffers (vs. that of a shared memory region)

Network architectural considerations are far too broad to be represented in a single document for the OSG, and the interested reader is encouraged to read the following resource provided by ESnet:

<http://fasterdata.es.net/science-dmz>

## Network Configuration Tuning

Much like end hosts, network devices have the ability to be “tuned” for specific use cases. This tuning normally centers on enabling or disabling certain features on a router or switch (e.g. policy maps) or adjusting the available memory available to account for a specific use case (e.g. less memory for a video application, more for a throughput intensive tool).

As every manufacturer provides different interfaces to the underlying hardware, we cannot make specific recommendations in this document. The interested reader is encouraged to read this guide provided by ESnet:

<http://fasterdata.es.net/network-tuning/router-tuning/>

# Measurement Software

The available span of network measurement software is vast. A simple web search will reveal 10s of names, some still active and others long dead. The R&E community began to standardize on available tools in the later part of the 2000s with an effort to unify measurement tools and infrastructure to support them: perfSONAR.

perfSONAR is a framework to simplify end to end network debugging. It consists of a layer of middleware, designed to sit between the measurement tools and the visualization and analysis that is useful to human users. A key component of the perfSONAR concept is the pS Performance Toolkit; this completely enclosed operating system packages performance tools and easy to follow GUIs to enable configuration.

perfSONAR focuses on several key metrics:

* Achievable Bandwidth
* One Way Latency
* Round Trip Latency
* Packet Loss, Duplication, Out of Orderness
* Interface Utilization, Errors, Discards
* Layer 3 Path
* Path MTU

Many of these metrics are calculated through simple tests that can be run from the command line. The OSG VDT package contains 3 key measurement tools that will be used as we discuss networking debugging in Section 7:

* BWCTL – A tool for measuring end to end bandwidth availability
* NDT – A tool designed to diagnose several different aspects of a host and network
* OWAMP – A tool designed for measuring one way delays of packets, as well as loss, duplication, and out of orderness.

These 3 command line tools, when installed on a compute or storage node, can be used to launch tests to perfSONAR servers located in the greater R&E networking world, e.g. on the campus, regional, backbone, or exchange point networks.

# Debugging Process

The following sections will discuss the process to install, use, and interpret measurement tools in an OSG software environment. End users are encouraged to try these steps first, but also contact their local support staff at the earliest possible moment. Involving support staff will ensure that expert eyes are available to assist with problems, and funnel the requests for help to the proper area (e.g. GOC, other networks, etc.).

## Software Installation

Client software can be installed in one of two ways, either though the OSG VDT or via RPMs from the perfSONAR web site.

## OSG Software

[INSERT INSTRUCTIONS ON HOW TO INSTALL VDT HERE]

## perfSONAR-PS Software

All perfSONAR software is available through an RPM (Red Hat Package Manager) repository to make for easy installation and updates. The following steps can be taken to install these tools:

* ***Import the Internet2 Signing Key***

The following command will import the key.

sudo rpm --import <http://software.internet2.edu/rpms/RPM-GPG-KEY-Internet2>

* ***Download RPM Software***

The latest version for CentOS 5 and 6 (both x86 and x86\_64 architectures) can be found on the the following web site:

<http://software.internet2.edu>

Note that SL5 andSL6, RHEL 5 and RHEL 6 should work with these builds. Those using other operating systems are suggested to try source builds that can be found at the same location.

* ***Run Yum Update***

The following command will invoke updates to the yum system, and also prepare the newly installed perfSONAR repository:

sudo yum update

* ***Search for Tools***

Yum can be searched in the following manner:

sudo yum search TOOLNAME

* ***Install Tools***

Yum can install tools in a similar fashion, the following command will install the client libraries:

sudo yum install owamp-client bwctl-client ndt-client

Note that some other tools may be pulled in automatically. Note that iperf and nuttcp are required for BWCTL to work.

## Tool Selection

Debugging network problems involves running several tools, and gathering results both an end-to-end basis as well as to points in the middle. Initial tool selection can depend on a couple of factors:

* What servers are available on the other end, as well as in the middle
* What use case is attempting to be debugged
* How sophisticated is the user running the tools

In general we recommend that users try “***all***” available tools, but in a structured and complete fashion before moving on to new tests. In general the following recommendation can be made regarding tool selection:

* Perform NDT client tests to the closest server possible. Additional tests to other points in the middle as needed.
* Perform end-to-end BWCTL tests to establish a baseline bandwidth. Perform a bisected BWCTL test to points on middle networks, testing in areas where performance is bad in favor of where it is good (e.g. narrow down the problem)
* Perform end-to-end OWAMP tests to establish baseline latency and loss. Perform a bisected OWAMP test to points on middle networks, testing in areas where performance is bad in favor of where it is good (e.g. narrow down the problem)

The following are some examples of how to use the tools from the command line:

* ***NDT***

NDT uses a command line client called **web100clt**. There are many options available, but in general you must supply a server name, and some debugging flags to get additional output. Here is a simple invocation:

[user@host ~]$ web100clt -n ndt.chic.net.internet2.edu

Testing network path for configuration and performance problems -- Using IPv6 address

Checking for Middleboxes . . . . . . . . . . . . . . . . . . Done

checking for firewalls . . . . . . . . . . . . . . . . . . . Done

running 10s outbound test (client to server) . . . . . 92.16 Mb/s

running 10s inbound test (server to client) . . . . . . 90.63 Mb/s

The slowest link in the end-to-end path is a 100 Mbps Full duplex Fast Ethernet subnet

Information: Other network traffic is congesting the link

Information [S2C]: Packet queuing detected: 15.06% (local buffers)

Server 'ndt.chic.net.internet2.edu' is not behind a firewall. [Connection to the ephemeral port was successful]

Client is probably behind a firewall. [Connection to the ephemeral port failed]

Information: Network Middlebox is modifying MSS variable (changed to 1440)

Server IP addresses are preserved End-to-End

Client IP addresses are preserved End-to-End

To get additional data, try adding the **-ll** flag, it will produce a more in depth analysis. NDT is useful as the first step of debugging to gather information about the end host, as well as the basic network configuration.

* ***BWCTL***

BWCTL is invoked from the command line with a number of options. Of these the following are important:

* + ***-f***  - Sets the format, supply either a byte based metric (K, M, G) or a bit based metric (k, m, g).
  + ***–t*** – Sets the length of the test in seconds
  + ***–i*** – Specifies the reporting interval (e.g. how often instantaneous bandwidth results are available) in seconds
  + ***–c*** – Specifics the host that will receive the flow of data, e.g. the “catcher”
  + ***–s*** – Specifics the host that will send the flow of data, e.g. the “sender”

An example of invoking BWCTL can be seen below. In this example we are sending data from the host we are on to another located in Kansas City MO, on the Internet2 network:

[user@host ~]$ bwctl -f m -t 10 -i 1 -c nms-rthr.kans.net.internet2.edu

bwctl: Using tool: iperf

bwctl: 93 seconds until test results available

RECEIVER START

bwctl: exec\_line: /usr/bin/iperf -B 64.57.16.210 -s -f m -m -p 5011 -t 10 -i 1

bwctl: start\_tool: 3568979157.239050

------------------------------------------------------------

Server listening on TCP port 5011

Binding to local address 64.57.16.210

TCP window size: 0.08 MByte (default)

------------------------------------------------------------

[ 14] local 64.57.16.210 port 5011 connected with 64.57.17.18 port 5011

[ 14] 0.0- 1.0 sec 105 MBytes 879 Mbits/sec

[ 14] 1.0- 2.0 sec 118 MBytes 990 Mbits/sec

[ 14] 2.0- 3.0 sec 118 MBytes 990 Mbits/sec

[ 14] 3.0- 4.0 sec 118 MBytes 990 Mbits/sec

[ 14] 4.0- 5.0 sec 118 MBytes 990 Mbits/sec

[ 14] 5.0- 6.0 sec 118 MBytes 990 Mbits/sec

[ 14] 6.0- 7.0 sec 118 MBytes 990 Mbits/sec

[ 14] 7.0- 8.0 sec 118 MBytes 990 Mbits/sec

[ 14] 8.0- 9.0 sec 118 MBytes 990 Mbits/sec

[ 14] 9.0-10.0 sec 118 MBytes 990 Mbits/sec

[ 14] 0.0-10.1 sec 1178 MBytes 979 Mbits/sec

[ 14] MSS size 8948 bytes (MTU 8988 bytes, unknown interface)

bwctl: stop\_exec: 3568979172.016198

RECEIVER END

This test reveals that over the course of 10 seconds we achieved an average bandwidth of 979Mbps and sent a total of 1178MB of data. We can reverse the direction of this test in the next example:

[user@host ~]$ bwctl -f m -t 10 -i 1 -s nms-rthr.kans.net.internet2.edu

bwctl: Using tool: iperf

bwctl: 75 seconds until test results available

RECEIVER START

bwctl: exec\_line: /usr/bin/iperf -B 64.57.17.18 -s -f m -m -p 5011 -t 10 -i 1

bwctl: start\_tool: 3568979241.960327

------------------------------------------------------------

Server listening on TCP port 5011

Binding to local address 64.57.17.18

TCP window size: 16.0 MByte (default)

------------------------------------------------------------

[ 14] local 64.57.17.18 port 5011 connected with 64.57.16.210 port 5011

[ ID] Interval Transfer Bandwidth

[ 14] 0.0- 1.0 sec 111 MBytes 929 Mbits/sec

[ 14] 1.0- 2.0 sec 118 MBytes 990 Mbits/sec

[ 14] 2.0- 3.0 sec 118 MBytes 990 Mbits/sec

[ 14] 3.0- 4.0 sec 118 MBytes 990 Mbits/sec

[ 14] 4.0- 5.0 sec 118 MBytes 990 Mbits/sec

[ 14] 5.0- 6.0 sec 118 MBytes 990 Mbits/sec

[ 14] 6.0- 7.0 sec 118 MBytes 990 Mbits/sec

[ 14] 7.0- 8.0 sec 118 MBytes 990 Mbits/sec

[ 14] 8.0- 9.0 sec 118 MBytes 990 Mbits/sec

[ 14] 9.0-10.0 sec 118 MBytes 990 Mbits/sec

[ 14] 0.0-10.2 sec 1193 MBytes 984 Mbits/sec

[ 14] MSS size 8948 bytes (MTU 8988 bytes, unknown interface)

bwctl: stop\_exec: 3568979256.889493

RECEIVER END

A similar result is seen in that we achieve near 1Gbps bandwidth (e.g. the hosts are only connected at 1Gbps).

BWCTL can (and should) be used to check available bandwidth between servers. Start first on the long path (e.g. end-to-end) then test to resources in the middle. Note that BWCTL supports a 3rd mode operation, wherein you can provide options for both the ‘-c’ and ‘-s’ and perform tests between these two hosts without being physically logged into either:

[user@host ~]$ bwctl -f m -t 10 -i 1 -s nms-rthr.kans.net.internet2.edu -c nms-rthr1.hous.net.internet2.edu

bwctl: Using tool: iperf

bwctl: 82 seconds until test results available

RECEIVER START

bwctl: exec\_line: /usr/bin/iperf -B 64.57.16.130 -s -f m -m -p 5001 -t 10 -i 1

bwctl: start\_tool: 3568979772.344387

------------------------------------------------------------

Server listening on TCP port 5001

Binding to local address 64.57.16.130

TCP window size: 0.08 MByte (default)

------------------------------------------------------------

[ 14] local 64.57.16.130 port 5001 connected with 64.57.16.210 port 5001

[ ID] Interval Transfer Bandwidth

[ 14] 0.0- 1.0 sec 103 MBytes 861 Mbits/sec

[ 14] 1.0- 2.0 sec 118 MBytes 990 Mbits/sec

[ 14] 2.0- 3.0 sec 118 MBytes 990 Mbits/sec

[ 14] 3.0- 4.0 sec 118 MBytes 990 Mbits/sec

[ 14] 4.0- 5.0 sec 118 MBytes 990 Mbits/sec

[ 14] 5.0- 6.0 sec 118 MBytes 990 Mbits/sec

[ 14] 6.0- 7.0 sec 118 MBytes 990 Mbits/sec

[ 14] 7.0- 8.0 sec 118 MBytes 990 Mbits/sec

[ 14] 8.0- 9.0 sec 118 MBytes 990 Mbits/sec

[ 14] 9.0-10.0 sec 118 MBytes 990 Mbits/sec

[ 14] 0.0-10.2 sec 1183 MBytes 977 Mbits/sec

[ 14] MSS size 8948 bytes (MTU 8988 bytes, unknown interface)

bwctl: stop\_exec: 3568979785.230833

RECEIVER END

BWCTL requires a stable NTP clock to work properly, be sure that NTP is configured before using this tool.

* ***OWAMP***

OWAMP is a tool that measures latency, loss, out of orderness, and duplication of packets between a source and a destination. Note that this test measures each direction ***independently*** versus that of the traditional round trip tool **ping**. Below is an example of a test:

[user@host ~]$ owping owamp.wash.net.internet2.edu

Approximately 12.6 seconds until results available

--- owping statistics from [eth-1.nms-rlat.newy32aoa.net.internet2.edu]:60455 to [owamp.wash.net.internet2.edu]:47148 ---

SID: 00160034d4ba4cad4e8c0485546b4ebf

first: 2013-02-04T15:05:18.240

last: 2013-02-04T15:05:27.254

100 sent, 0 lost (0.000%), 0 duplicates

one-way delay min/median/max = 2.02/2.1/2.06 ms, (err=0.218 ms)

one-way jitter = 0 ms (P95-P50)

Hops = 2 (consistently)

no reordering

--- owping statistics from [owamp.wash.net.internet2.edu]:47149 to [eth-1.nms-rlat.newy32aoa.net.internet2.edu]:33562 ---

SID: 00170098d4ba4cad8eb45282697d2cc2

first: 2013-02-04T15:05:18.259

last: 2013-02-04T15:05:27.175

100 sent, 0 lost (0.000%), 0 duplicates

one-way delay min/median/max = 3.19/3.3/3.27 ms, (err=0.218 ms)

one-way jitter = 0 ms (P95-P50)

Hops = 2 (consistently)

no reordering

The results clearly state each direction of operation, and any problems that were found. As in the BWCTL case the tool is highly reliant on stable NTP numbers, so be sure your server is synchronized against an NTP server.

OWAMP is a lightweight test and can be used to show minor amounts of packet loss between hosts. Perform the test on the full end-to-end path, and then bisect the path by testing to points in the middle. Often low throughput observed via BWTL will show up as packet loss in OWAMP.

## End-to-end Testing

The concept of end-to-end testing is required as a first step in debugging network problems. Via the OSG tools it is possible to use “client” tools as discussed in Section 7.2 to gauge the total end-to-end path. These client tools can be pointed at a pS Performance Toolkit instance installed on the remote end, or via stand-alone daemon applications started on OSG systems. In either case, a daemon and client will be needed.

The following procedure should be followed:

* Notify local networking staff at each end that are noticing problems, and would like to investigate them. Note that you can run tests end-to-end, and share them when you are complete.
* Identify Servers on both ends (e.g. standalone pS Performance Toolkit instances or starting daemons on OSG servers)
* Identify clients on both ends, normally the compute or storage nodes. Avoid using machines that are not involved in the OSG software process such as laptops.
* Perform end-to-end testing with:
  + NDT
  + BWCTL
  + OWAMP
* Perform several tests and always record the results. It’s a good idea to run at different times during the day, and note when you ran the tests
* Share results with local network staff, and open a ticket with the GOC if you feel it is something they can help investigate.

After end-to-end testing, and examining the results with local and GOC based professionals, it may be time to embark on a larger debugging exercise with partial path decomposition.

## Partial Path Decomposition

As we saw in Section 7.4, it is necessary to use all tools in a structured and scripted manner. Deciding to divide the path is no different. The following steps should be followed:

* Using a tool like traceroute or tracepath, find the paths between you and the remote host. If possible validate the path for the reverse direction as well. It may be possible that these are different.
* For one of the networks on the path, usually one in the direct middle (often a backbone network like Internet2, ESnet, or NLR), find a testing host. If these are not posted on public we pages, send an email to the support teams (e.g. [rs@internet2.edu](mailto:rs@internet2.edu), or [trouble@es.net](mailto:trouble@es.net)).
* Perform end-to-middle testing from the source and desgination with:
  + NDT
  + BWCTL
  + OWAMP
* Perform several tests and always record the results. It’s a good idea to run at different times during the day, and note when you ran the tests
* Share results with local network staff, and open a ticket with the GOC if you feel it is something they can help investigate.
* If the tests show one ‘side’ as being better than the other, you can repeat this process by further bisecting the path on the side with the problem.

## Interpreting Results

Interpretation of results can be tricky due to the nature of protocols on the network, including TCP. In general the only symptom that is given off to a problem with TCP is “low throughput”. The following are some tips on interpreting results:

* NDT will denote if your host does not have the proper amount of tuning. If it doesn’t, please considering following the host tuning steps discussed in Section 5.1
* NDT will give the first indication of network problems as well, and may indicate the presence of packet loss, link bottlenecks, or congestion. Since NDT is based on heuristics these results can turn out to be false positives, but are often worthy of following up on. In the event of congestion, ask local networking staff to see if there are any heavily utilized links. If packet loss is an issue, ask to see if any interface errors or discards are present.
* BWCTL, when used in TCP mode, is only useful at nothing high or low throughput. This is normally good from a capability standpoint, but it cannot tell you anything else about a serious problem
* OWAMP is useful for detecting loss. The theory is that if you notice the loss of small UDP packets produced by OWAMP, the same behavior will be seen in the form of low throughput from a tool like BWCTL.
* OWAMP can also be used to show asymmetric routing (with the aide of tools like traceroute) or if queuing and congestion are becoming a factor in one direction vs. another.
* Bisecting a path, and being patient, are normally the only ways to narrow down problems.

In addition to these adages, please consider asking your local staff for assistance when you first notice a problem. If they are unable to help, consult the resources listed in Section 8.

# Additional Help

The following locations can be consulted for more help in debugging network problems:

* Internet2 Research Support Center – [rs@internet2.edu](mailto:rs@internet2.edu)
* ESnet Trouble Reporting – [trouble@es.net](mailto:trouble@es.net)
* NLR NOC - [noc@nlr.net](mailto:noc@nlr.net)
* OSG GOC - [goc@opensciencegrid.org](mailto:goc@opensciencegrid.org)

# Acknowledgements

The authors would like to acknowledge and thank the OSG community for their support and feedback into network performance problems and tools that would be useful for end users.

The perfSONAR-PS community has been invaluable, and the authors would like to thank them for their generous contributions of software, expertise, and time.

# References

TBD