www.internet2.edu

Network Diagnostic Tool (NDT): An Internet2 Cookbook

Disclosure/Disclaimer

This work was supported (in part) by the Office of Science, U.S. Department of Energy under Contract W-31-109-ENG-38.

Packet-Pair work was supported by the Cisco University Research Program Work-for-Others Contract P-03008.

Duplex mismatch work was supported by the National Library of Medicine under contract 467-MZ-401761.

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/.

NDT is the Network Diagnostic Tool, currently being modified for use with E2E piPEs and the Abilene Measurement Infrastructure. More information on this tool can be found at: http://e2epi.internet2.edu/ndt/.

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 NDT servers at your location.

An Overview of the Web100-Based NDT

Motivation

The major objectives for developing this tool are to:

- Measure performance directly to users desktop
- Develop a diagnostic tool that doesn't use historical data
- Combine numerous Web100 variables to analyze connection
- Develop network signatures for 'typical' network problems

It is difficult or impossible for a campus administrator to run repetitive tests to every desktop on site. Even if it were done, running enough tests to get a statistically valid baseline for every desktop or laptop computer is extremely difficult. A better approach is needed to allow this type of testing to occur on an as-needed basis. The NDT tester meets these objectives and allows users to self-test their desktop or laptop computer when they suspect a problem exists. The results include measured results, analysis of those results to point out common problems, some basic suggestions on how to fix them.

Methodology

The NDT is designed to identify both performance problems and configuration problems. Performance problems affect the user experience, usually causing data transfers to take longer than expected. These problems are usually solved by tuning various TCP (Transmission Control Protocol) network parameters on the end host. Configuration problems also affect the user experience; however, tuning will not improve the end-to-end performance. The configuration fault must be found and corrected to change the end host behavior. To accomplish these tasks, the tool will:

- Identify specific problem(s) that affect end users
- Analyze problem to determine 'Network Signature' for this problem
- Provide testing tool to automate detection process

Web100-Based Approach

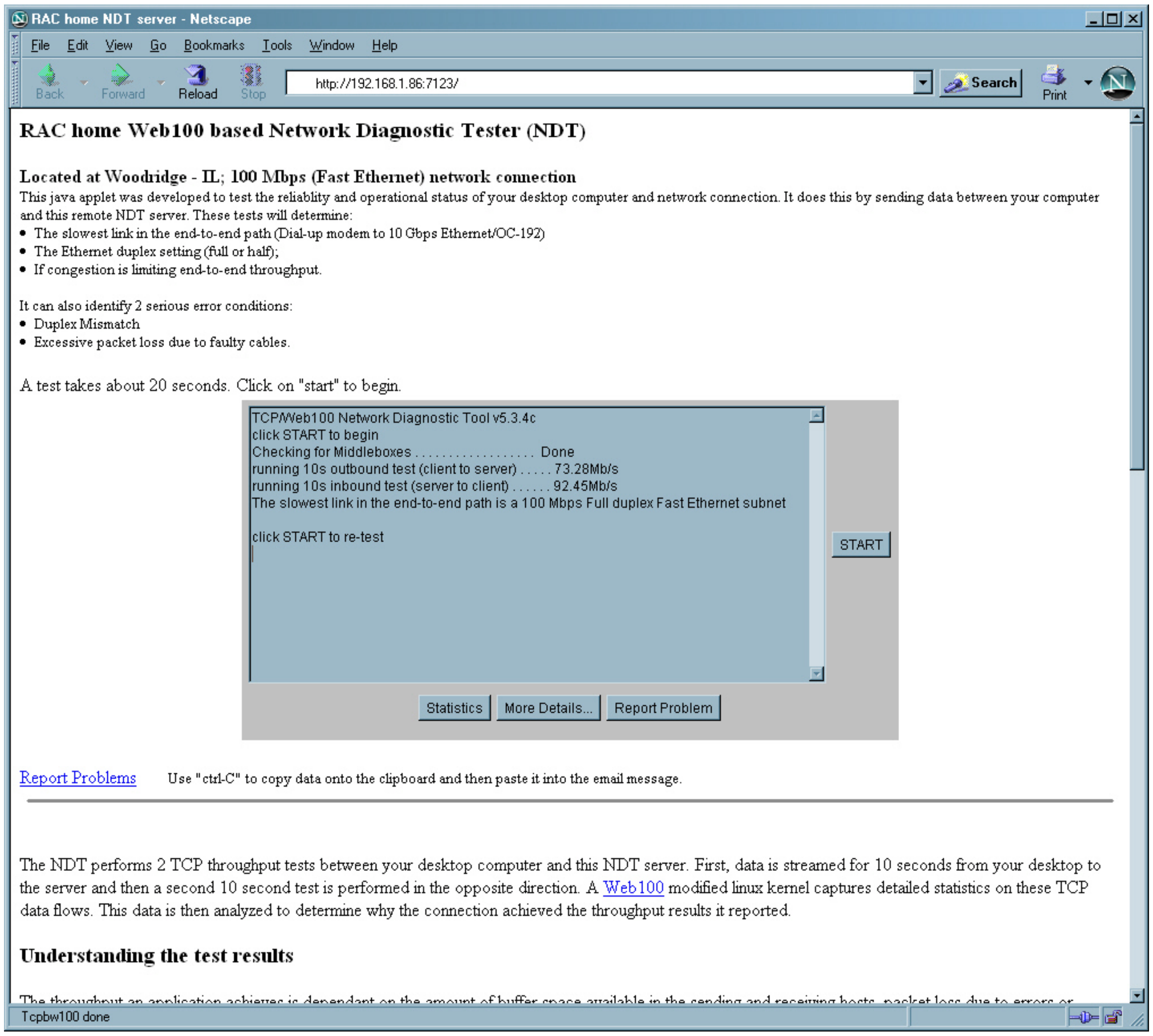
The majority of the traffic on the Internet uses the TCP to reliably deliver IP data packets between two communicating hosts. The major advantage of TCP is that it automatically recovers from faults and errors to ensure that the application data is successfully transferred from source to destination. The major disadvantage is that every problem TCP encounters results in a slow-down of the data transfer. This makes it difficult to determine why the connection slowed down.

The Web100 project was designed to address this problem, by exposing many of TCP's internal variables. Thus, users and applications are able to identify the reasons why each TCP connection receives the measured performance.

The NDT performs the following tasks:

- Simple bi-directional test to gather E2E data
- Gather multiple data variables from server
- Compare measured performance to analytical values
- Translate network values into plain text messages

An analogy is that repetitive tests build up an historical record that can point out when changes occur (a depth of measurement data). The NDT relies on multiple data variables (a breadth of measurement data) to achieve similar results.



RAC home Web100 based Network Diagnostic Tester (NDT)

Located at Woodridge - IL; 100 Mbps (Fast Ethernet) network connection

This java applet was developed to test the reliablity and operational status of your desktop computer and network connection. It does this by sending data between your computer and this remote NDT server. These tests will determine:

- The slowest link in the end-to-end path (Dial-up modem to 10 Gbps Ethernet/OC-192)
- The Ethernet duplex setting (full or half);
- If congestion is limiting end-to-end throughput.

It can also identify 2 serious error conditions:

- Duplex Mismatch
- Excessive packet loss due to faulty cables.

A test takes about 20 seconds. Click on "start" to begin.

```
TCP/Web100 Network Diagnostic Tool v5.3.4c
click START to begin
Checking for Middleboxes . . . . . . . . . . . . . . . . . .  Done
running 10s outbound test (client to server) . . . . . 73.28Mb/s
running 10s inbound test (server to client) . . . . . . 92.45Mb/s
The slowest link in the end-to-end path is a 100 Mbps Full duplex Fast Ethernet subnet

click START to re-test
```

Use "ctrl-C" to copy data onto the clipboard and then paste it into the email message.

The NDT performs 2 TCP throughput tests between your desktop computer and this NDT server. First, data is streamed for 10 seconds from your desktop to the server and then a second 10 second test is performed in the opposite direction. A Web100 modified linux kernel captures detailed statistics on these TCP data flows. This data is then analyzed to determine why the connection achieved the throughput results it reported.

Understanding the test results

Figure 1.1. Web-based Java client

Benefits

The NDT has the following benefits:

- End user-based view of network
- Can identify configuration problems
- Can identify performance bottlenecks
- Provides some ‘hard evidence’ to users and network administrators to reduce finger pointing
- Doesn’t rely on historical data

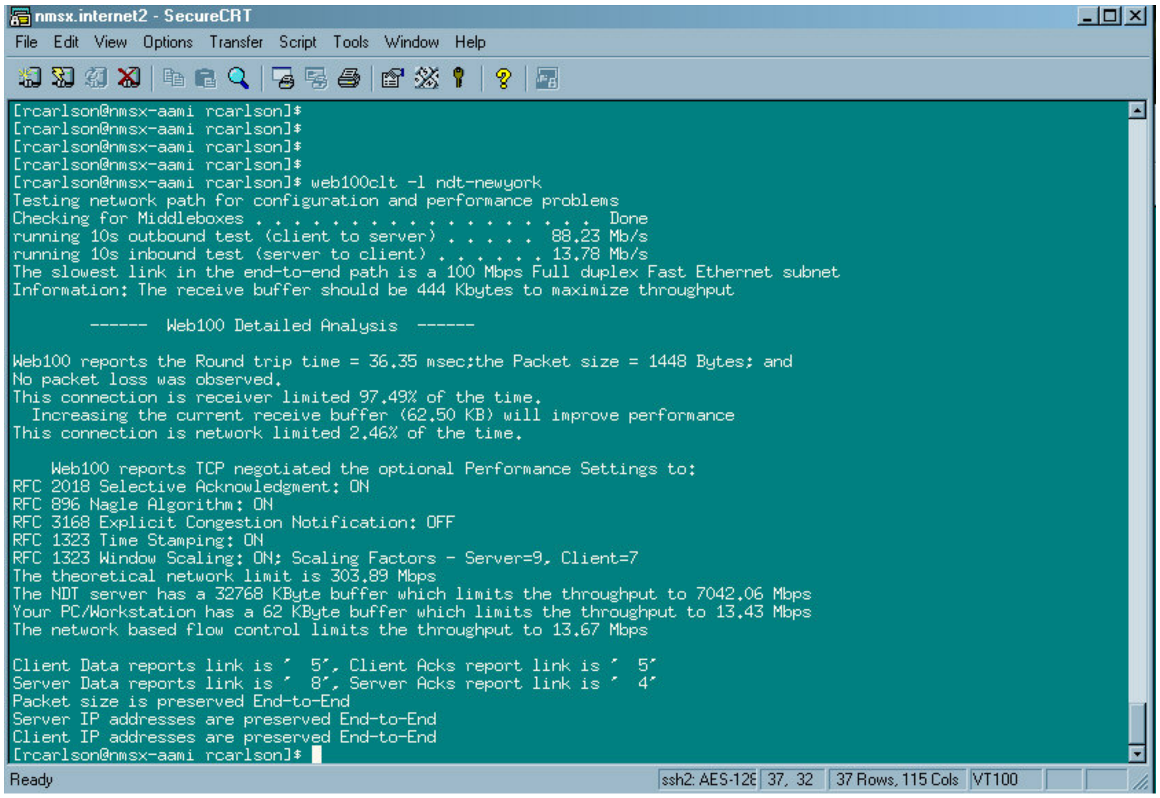
The biggest benefit is that the tool may be run by any user on an as-needed basis. This allows the user to self-test their network connection and their desktop or laptop computer in a real environment. Other tests that replace the host with a well -known test machine may measure the network infrastructure, but they do not measure the host stack or look for transient problems. This can lead to confusion and finger pointing when the network staff says “no problem found” but the user is still in the dark about what problems exist and, more importantly, how they can be fixed.

In addition, providing hard evidence is necessary to make the user feel that something can be done to improve their situation. The NDT operates on any client with a Java-enabled Web browser; further:

- What it *can* do:
    - Positively state if Sender, Receiver, or Network is operating properly
    - Provide accurate application tuning info
    - Suggest changes to improve performance
- What it *can’t* do:
    - Tell you where in the network the problem is
    - Tell you how other servers perform
    - Tell you how other clients will perform

Figure 1.1 shows the web-based Java client. Using this client means the applet automatically downloads into the client, eliminating the need to pre-install the software on the client machine. This is a plus when a new user wants to test; nothing must be pre-loaded before a test can begin.

It is also important to define the NDT strengths and deficiencies (see below). Note that the performance tuning information is based on getting to the NDT server, not the real application host. Thus, the tuning information may be suspect but it should provide the right trends in setting buffer sizes.


nmsx.internet2 - SecureCRT
File Edit View Options Transfer Script Tools Window Help
Ready
ssh2: AES-128 37, 32 37 Rows, 115 Cols VT100

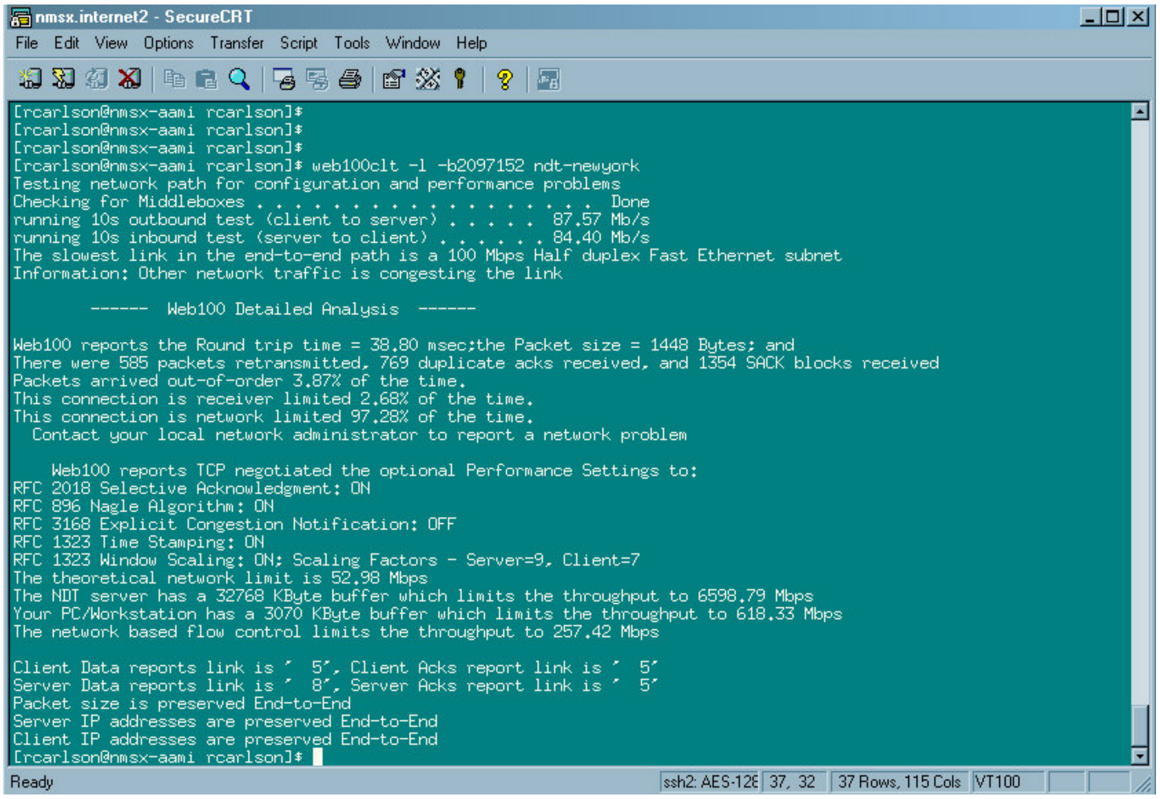
```
[rcarlson@nmsx-aami rcarlson]$
[rcarlson@nmsx-aami rcarlson]$
[rcarlson@nmsx-aami rcarlson]$
[rcarlson@nmsx-aami rcarlson]$
[rcarlson@nmsx-aami rcarlson]$ web100clt -l ndt-newyork
Testing network path for configuration and performance problems
Checking for Middleboxes . . . . . . . . . . . . . . . . . .  Done
running 10s outbound test (client to server) . . . . .  88.23 Mb/s
running 10s inbound test (server to client) . . . . . . 13.78 Mb/s
The slowest link in the end-to-end path is a 100 Mbps Full duplex Fast Ethernet subnet
Information: The receive buffer should be 444 Kbytes to maximize throughput

        ------  Web100 Detailed Analysis  ------

Web100 reports the Round trip time = 36.35 msec;the Packet size = 1448 Bytes; and
No packet loss was observed.
This connection is receiver limited 97.49% of the time.
  Increasing the current receive buffer (62.50 KB) will improve performance
This connection is network limited 2.46% of the time.

    Web100 reports TCP negotiated the optional Performance Settings to:
RFC 2018 Selective Acknowledgment: ON
RFC 896 Nagle Algorithm: ON
RFC 3168 Explicit Congestion Notification: OFF
RFC 1323 Time Stamping: ON
RFC 1323 Window Scaling: ON; Scaling Factors - Server=9, Client=7
The theoretical network limit is 303.89 Mbps
The NDT server has a 32768 KByte buffer which limits the throughput to 7042.06 Mbps
Your PC/Workstation has a 62 KByte buffer which limits the throughput to 13.43 Mbps
The network based flow control limits the throughput to 13.67 Mbps

Client Data reports link is '  5', Client Acks report link is '  5'
Server Data reports link is '  8', Server Acks report link is '  4'
Packet size is preserved End-to-End
Server IP addresses are preserved End-to-End
Client IP addresses are preserved End-to-End
[rcarlson@nmsx-aami rcarlson]$
```

Figure 1.2. Problem: TCP buffer size too small

As noted above, it’s important to define what the NDT can’t do. There is enough variation is the Internet and in individual hosts that running a test to one desktop will not provide any help in determining how another computer will operate. Neither does it help tell you how your desktop will operate when talking to a different server (system load, file system constraints) all play a role in the wall clock time required to complete a specific task.

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.


nmsx.internet2 - SecureCRT
File Edit View Options Transfer Script Tools Window Help
Ready
ssh2: AES-128
37, 32
37 Rows, 115 Cols
VT100

```
[rcarlson@nmsx-aami rcarlson]$
[rcarlson@nmsx-aami rcarlson]$
[rcarlson@nmsx-aami rcarlson]$
[rcarlson@nmsx-aami rcarlson]$ web100clt -l -b2097152 ndt-newyork
Testing network path for configuration and performance problems
Checking for Middleboxes . . . . . . . . . . . . . . . . . .  Done
running 10s outbound test (client to server) . . . . .  87.57 Mb/s
running 10s inbound test (server to client) . . . . . . 84.40 Mb/s
The slowest link in the end-to-end path is a 100 Mbps Half duplex Fast Ethernet subnet
Information: Other network traffic is congesting the link

        ------  Web100 Detailed Analysis  ------

Web100 reports the Round trip time = 38.80 msec;the Packet size = 1448 Bytes; and
There were 585 packets retransmitted, 769 duplicate acks received, and 1354 SACK blocks received
Packets arrived out-of-order 3.87% of the time.
This connection is receiver limited 2.68% of the time.
This connection is network limited 97.28% of the time.
  Contact your local network administrator to report a network problem

    Web100 reports TCP negotiated the optional Performance Settings to:
RFC 2018 Selective Acknowledgment: ON
RFC 896 Nagle Algorithm: ON
RFC 3168 Explicit Congestion Notification: OFF
RFC 1323 Time Stamping: ON
RFC 1323 Window Scaling: ON; Scaling Factors - Server=9, Client=7
The theoretical network limit is 52.98 Mbps
The NDT server has a 32768 KByte buffer which limits the throughput to 6598.79 Mbps
Your PC/Workstation has a 3070 KByte buffer which limits the throughput to 618.33 Mbps
The network based flow control limits the throughput to 257.42 Mbps

Client Data reports link is '  5', Client Acks report link is '  5'
Server Data reports link is '  8', Server Acks report link is '  5'
Packet size is preserved End-to-End
Server IP addresses are preserved End-to-End
Client IP addresses are preserved End-to-End
[rcarlson@nmsx-aami rcarlson]$
```

Figure 1.3. Small TCP buffer size problem solved

Bottleneck Link Detection

What is the slowest link in the end-2-end path? NDT:

- Monitors packet arrival times using libpcap routine
- Uses TCP dynamics to create packet pairs
- Quantizes results into link type bins (no fractional or bonded links)

This is the first major task: find the bottleneck link speed. For example, suppose you have a 10/100/1000 interface card and the intra-building network is GigE-based. In this case, you might reasonably expect that your desktop could transfer files to/from a Gig-E attached server at 1000 Mbps. However, assume that the switch feeding your wall jack was only a FastE network port. Your throughput would be limited to 100 Mbps – far less than you expected. To determine if a problem really exists, you would need to know the true end-to-end path capacity. The NDT easily will supply this information by telling you that the bottleneck is a FastE link somewhere in the path. You could then talk to your network administrator to find and replace this slow link to obtain the expected performance.

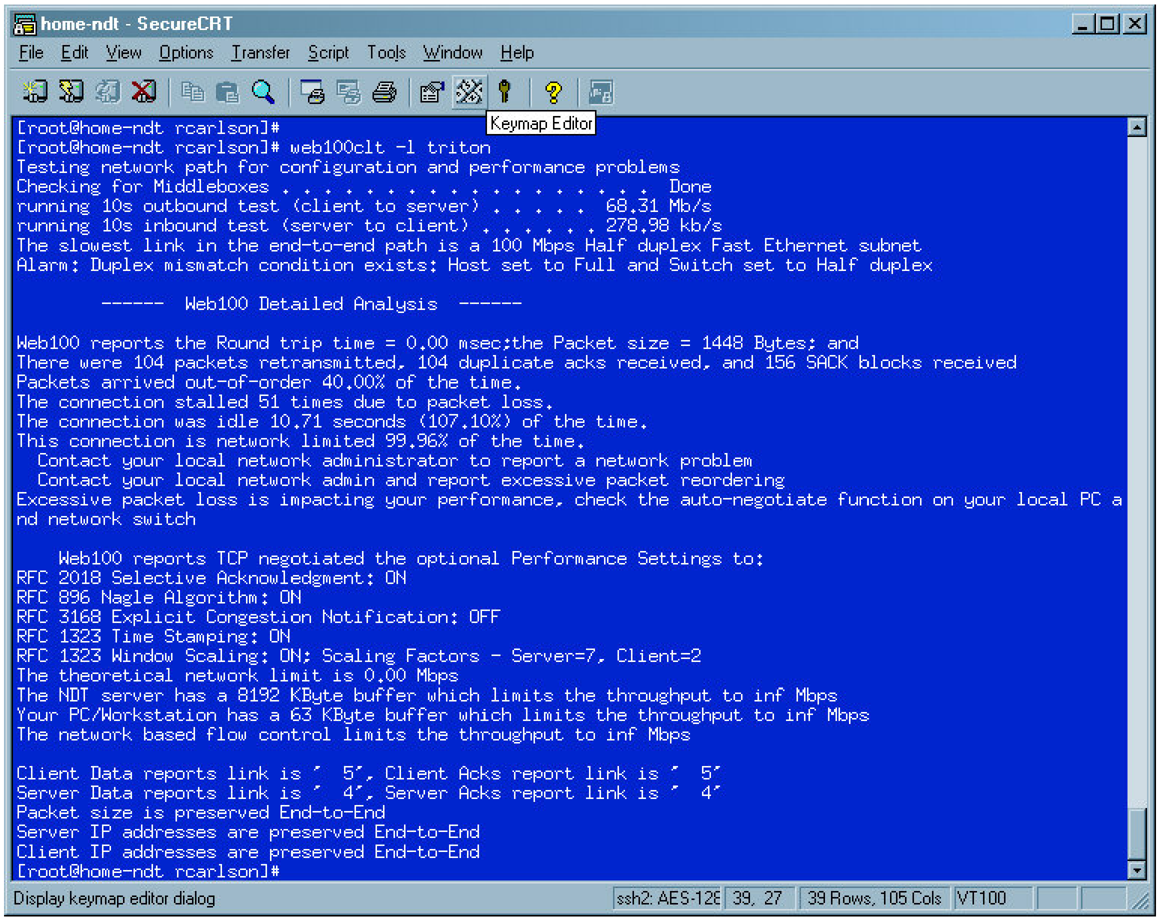
Another example: suppose the path takes you through a slow exchange point and there is a backup Ethernet link being used while the normal FastE link is down for some reason. The NDT will report that a bottleneck Ethernet links exists. Again, this message can quickly point out that a configuration problem exists instead of a performance problem.

This will prevent users from spending numerous hours trying to tune the host to achieve better performance.

Figures 1.2 and 1.3 illustrate the benefit of this approach. In Figure 1.2, we see that a host achieves almost 90 Mbps while receiving only 13 Mbps during transmission. Just looking at those results, it is not possible to determine if this is the expected speed or why the difference occurs. If you look closely at the messages that appear below the speeds, you see that the NDT determined that the slowest link in the end-to-end path is a 100 Mbps Fast Ethernet link. With this information, we can determine that the client-to-server direction is achieving over 88% of the maximum network capacity. This means that little could be done to improve things.

However, this does not answer the question of why the server-to-client speed is so slow. Again, if you look down at the messages, you see that the speed is limited due to the TCP buffer setting (62 Kbytes) on the desktop client. Re-running the test, but with a larger TCP buffer (2 Mbytes) shows a jump in the server-to-client speed. The user now has two choices:

1) Manually specify a larger TCP buffer size every time an application runs, or
2) Have the system administrator increase the default TCP buffer size.


home-ndt - SecureCRT
File Edit View Options Transfer Script Tools Window Help
Keymap Editor
Display keymap editor dialog
ssh2: AES-128
39, 27
39 Rows, 105 Cols
VT100

```
[root@home-ndt rcarlson]#
[root@home-ndt rcarlson]# web100clt -l triton
Testing network path for configuration and performance problems
Checking for Middleboxes . . . . . . . . . . . . . . . . . .  Done
running 10s outbound test (client to server) . . . . .  68.31 Mb/s
running 10s inbound test (server to client) . . . . . . 278.98 kb/s
The slowest link in the end-to-end path is a 100 Mbps Half duplex Fast Ethernet subnet
Alarm: Duplex mismatch condition exists: Host set to Full and Switch set to Half duplex

        ------  Web100 Detailed Analysis  ------

Web100 reports the Round trip time = 0.00 msec;the Packet size = 1448 Bytes; and
There were 104 packets retransmitted, 104 duplicate acks received, and 156 SACK blocks received
Packets arrived out-of-order 40.00% of the time.
The connection stalled 51 times due to packet loss.
The connection was idle 10.71 seconds (107.10%) of the time.
This connection is network limited 99.96% of the time.
  Contact your local network administrator to report a network problem
  Contact your local network admin and report excessive packet reordering
Excessive packet loss is impacting your performance, check the auto-negotiate function on your local PC a
nd network switch

    Web100 reports TCP negotiated the optional Performance Settings to:
RFC 2018 Selective Acknowledgment: ON
RFC 896 Nagle Algorithm: ON
RFC 3168 Explicit Congestion Notification: OFF
RFC 1323 Time Stamping: ON
RFC 1323 Window Scaling: ON; Scaling Factors - Server=7, Client=2
The theoretical network limit is 0.00 Mbps
The NDT server has a 8192 KByte buffer which limits the throughput to inf Mbps
Your PC/Workstation has a 63 KByte buffer which limits the throughput to inf Mbps
The network based flow control limits the throughput to inf Mbps

Client Data reports link is '  5', Client Acks report link is '  5'
Server Data reports link is '  4', Server Acks report link is '  4'
Packet size is preserved End-to-End
Server IP addresses are preserved End-to-End
Client IP addresses are preserved End-to-End
[root@home-ndt rcarlson]#
```

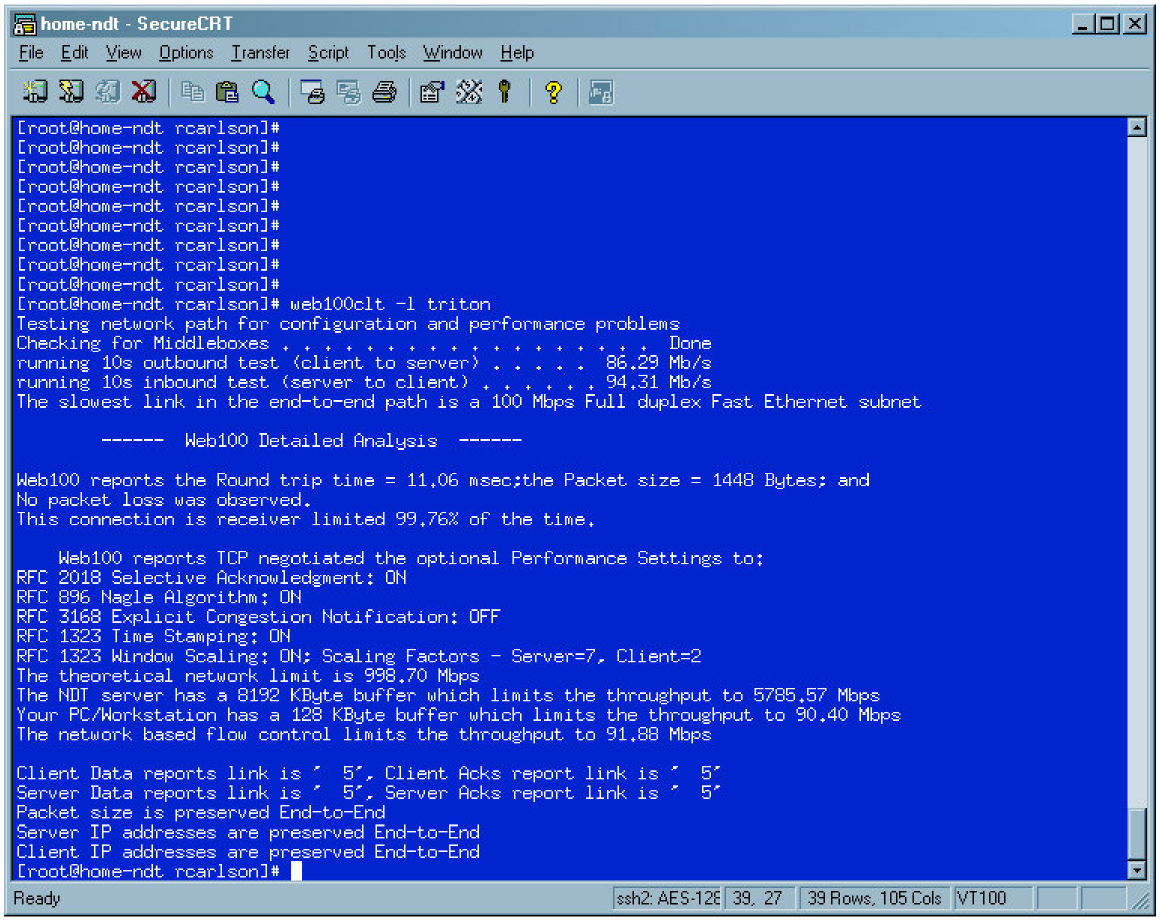
Figure 1.4. Duplex mismatch detected

Either option will allow the host to achieve better performance over the long network path used in this test. Also, notice that the NDT client suggests a buffer size to maximize the application speed.

The NDT uses packet dispersion techniques; e.g., it measures the inter-packet arrival times for all data and ACK packets sent or received. It also knows the packet size, so it can calculate the speed for each pair of packets sent or received. The results are then quantized, meaning that the NDT doesn't recognize fractional link speed (Ethernet, T3, or FastE). It also wouldn't detect bonded Etherchannel interfaces.

Duplex Mismatch Detection

Duplex mismatch is a condition whereby the host Network Interface Card (NIC) and building switch port fail to agree on this basic network operating parameter. While this failure will have a large impact on application performance, basic network connectivity still exists. This means that normal testing procedures (e.g., ping, traceroute) will report that no problem exists while real applications will run extremely slowly.



```
[root@home-ndt rcarlson]#
[root@home-ndt rcarlson]#
[root@home-ndt rcarlson]#
[root@home-ndt rcarlson]#
[root@home-ndt rcarlson]#
[root@home-ndt rcarlson]#
[root@home-ndt rcarlson]#
[root@home-ndt rcarlson]#
[root@home-ndt rcarlson]#
[root@home-ndt rcarlson]# web100clt -l triton
Testing network path for configuration and performance problems
Checking for Middleboxes . . . . . . . . . . . . . . . . . .  Done
running 10s outbound test (client to server) . . . . .  86.29 Mb/s
running 10s inbound test (server to client) . . . . . . 94.31 Mb/s
The slowest link in the end-to-end path is a 100 Mbps Full duplex Fast Ethernet subnet

        ------  Web100 Detailed Analysis  ------

Web100 reports the Round trip time = 11.06 msec;the Packet size = 1448 Bytes; and
No packet loss was observed.
This connection is receiver limited 99.76% of the time.

    Web100 reports TCP negotiated the optional Performance Settings to:
RFC 2018 Selective Acknowledgment: ON
RFC 896 Nagle Algorithm: ON
RFC 3168 Explicit Congestion Notification: OFF
RFC 1323 Time Stamping: ON
RFC 1323 Window Scaling: ON; Scaling Factors - Server=7, Client=2
The theoretical network limit is 998.70 Mbps
The NDT server has a 8192 KByte buffer which limits the throughput to 5785.57 Mbps
Your PC/Workstation has a 128 KByte buffer which limits the throughput to 90.40 Mbps
The network based flow control limits the throughput to 91.88 Mbps

Client Data reports link is '  5', Client Acks report link is '  5'
Server Data reports link is '  5', Server Acks report link is '  5'
Packet size is preserved End-to-End
Server IP addresses are preserved End-to-End
Client IP addresses are preserved End-to-End
[root@home-ndt rcarlson]#
```

Figure 1.5. Duplex mismatch problem resolved

The NDT is designed to identify when this problem exists. To accomplish this, we:

- Developed an analytical model to describe how Ethernet responds,
- Expanded the model to describe UDP and TCP flows,
- Developed a practical detection algorithm, and
- Tested models in LAN, MAN, and WAN environments.

Improving the detection of this problem has been the focus of recent work. We have an analytical model and a detection algorithm was created based on this model.

Figures 1.4 and 1.5 show how this problem manifests itself and how the NDT reports the problem. In Figure 1.4, a test was run with a duplex mismatch condition in the network path. The asymmetric speeds clearly show that something serious is wrong, but is it a performance or configuration problem? In this case, the user may not notice a problem while uploading files to a server, but she notices that downloading takes a very long time. Also, notice that basic connectivity does exist so the network administrator would be hard pressed to determine what exact problem exists.

However, the NDT clearly states that a duplex mismatch condition exists to aid the network administrator in understanding that a real problem is occurring. The administrator can look at the various switches and routers in the path to determine which link has the fault. In most cases, this fault will exist in the link connecting the user's desktop to the rest of the network. Figure 1.5 shows how performance is improved when the mismatch condition is resolved.

Future Enhancements

In addition to detecting the previously mentioned conditions, development work is continuing. Future versions will perform:

- WiFi detection
- Faulty Hardware detection
- Congestion modification
- Full/Half duplex detection

IEEE 802.11 (WiFi) Detection

The main goal for this is to detect when a host is connected via wireless (WiFi) link:

- Radio signal changes strength
- NICs implement power saving features
- Multiple standards (a/b/g/n)

Faulty Hardware Link Detection

The main goal for this is to detect non-congestive loss due to:

- Faulty NIC/switch interface
- Bad Cat-5 cable
- Dirty optical connector

Preliminary works shows that it is possible to distinguish between congestive and non-congestive loss.

Full/Half Link Duplex Setting

The main goal for this is to detect a half-duplex link in end-to-end path; NDT will identify when throughput is limited by half-duplex operations. Preliminary work shows detection is possible when link transitions between blocking states. The issue is maximizing performance; a half-duplex link will not achieve as high a speed as a full duplex link. Note: Old Ethernet hubs *require* half-duplex operation.

Normal Congestion Detection

Shared network infrastructures will cause periodic congestion episodes; the goal is for NDT to:

- Detect/report when TCP throughput is limited by cross-traffic
- Detect/report when TCP throughput is limited by own traffic

The issue is to detect when your traffic is sharing the network infrastructure with other users. In this case, you should get 1/Nth of the bottleneck link speed. It would also be nice to know when TCP is entering the congestion avoidance phase.

Functions and Features

The NDT tester is a client-server based tool that consists of two major components. Two server -side programs handle the web-based interface and testing/analysis of the network path. In addition, a client program, Java-based and command line –based, allow various clients to run tests.

The server programs contain these basic features:

- Basic configuration file (the admin can store run time options in a configuration file)
- FIFO scheduling of tests (NDT will handle multiple request in a First-In, First-Out manner; others wait in a queue for service)
- Simple server discovery protocol (allows multiple servers to find each other when operating in Federated mode)
- Federation mode support (allows multiple servers to redirect clients to the ‘closest’ NDT server)
- Command line client support (allows administrators to run test remotely without access to web browser)

Availability

This project was created as an open source product; see the sourceforge.net project page (http://www.sourceforge.net/projects/ndt) for more information. The tool, as well as the source code, is available at: http://e2epi.internet2.edu/ndt/download.html. Email-based discussion lists are available; go to http://e2epi.internet2.edu/ndt web site and choose:

- ndt-users – General discussion on NDT tool
- ndt-announce – Announcements on new features

Flow Chart

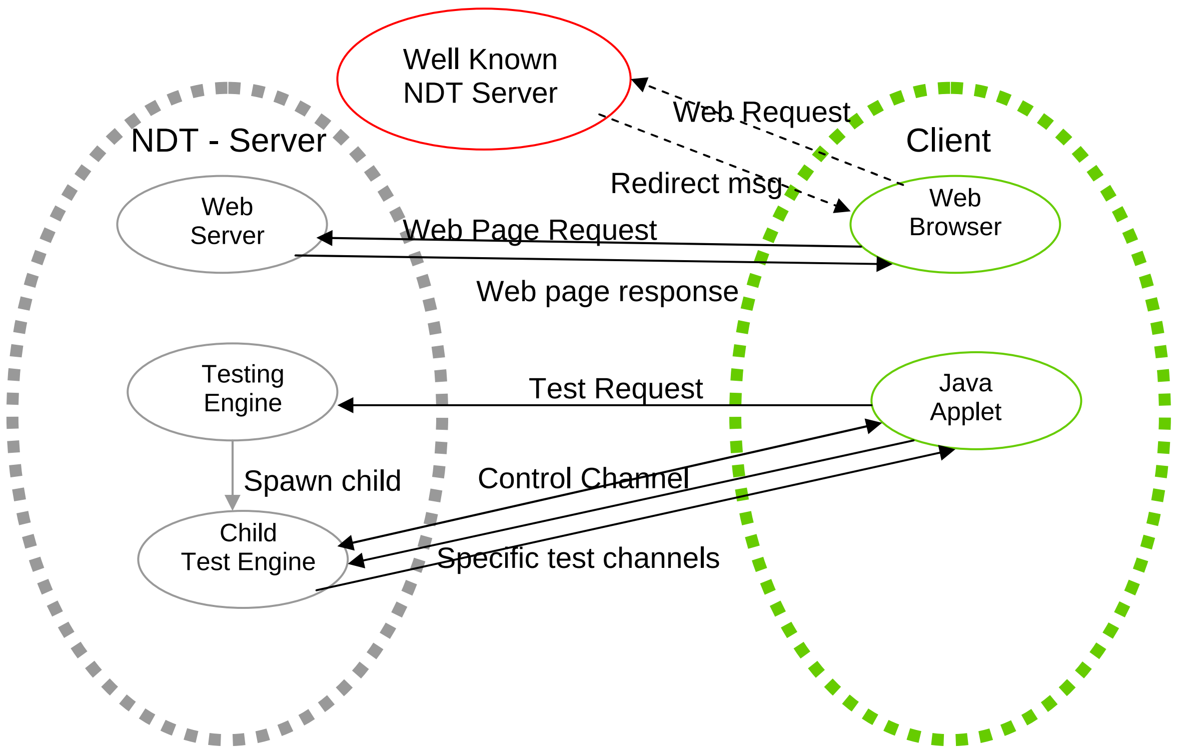
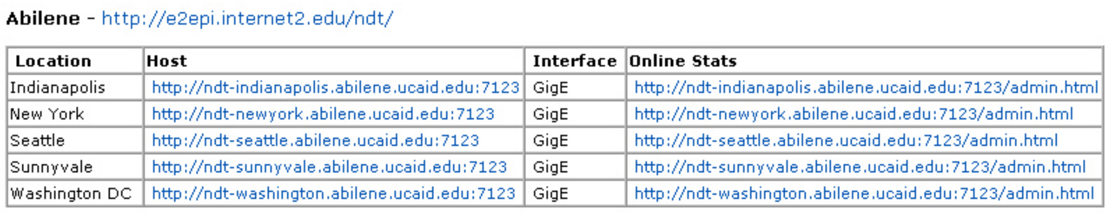


Figure 1.6. Basic flow chart for the NDT program

1) The process starts with the user opening a browser and entering the NDT server URL
   a) An optional step is to point to a well known server and accept a redirect message (Federated mode)
   b) Otherwise the URL points to the NDT server itself (either an apache web server or the fakewww process answer the request)
2) The web server responds by returning the page, with an embedded java applet (class or jar file)
3) The user must manually request a test be performed by clicking the “start” button
4) The applet opens a connection back to the server’s testing engine (web100srv process)
5) A child process is created to handle the test and the parent goes back to listening for more test requests. The parent keeps a FIFO queue to process multiple requests.
6) A control channel is created between the server and client to control the client’s actions and synchronize the start of the various tests.
7) The client opens two new data channels back to the client for testing purposes. Allowing the client to open connections makes it easy to get past client-side firewall boxes.
8) The client opens and closes a connection to perform the middle-box test
9) The client streams data back to the server to measure the clients upload speed.
10) The client opens another connection and the server streams data back to the client measuring the clients download speed
11) The server extracts the Web100 data and analyzes the connection for faults.
12) The results are recorded in the servers’ log file and the results are returned to the client for display to the user.

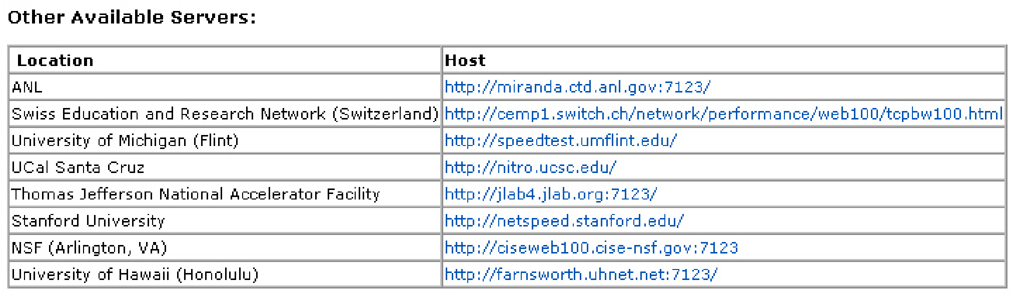
Publicly-Accessible Servers

Below is a list of servers on the Abilene network, and other public servers. Note: This is not a complete list; more are being added as they become available. In addition, several institutions run private servers, notably DoD and possibly DoE NNSA. There are no restrictions on the use, other than the University of Chicago public license requirement.



Abilene - http://e2epi.internet2.edu/ndt/

Location	Host	Interface	Online Stats
Indianapolis	http://ndt-indianapolis.abilene.ucaid.edu:7123	GigE	http://ndt-indianapolis.abilene.ucaid.edu:7123/admin.html
New York	http://ndt-newyork.abilene.ucaid.edu:7123	GigE	http://ndt-newyork.abilene.ucaid.edu:7123/admin.html
Seattle	http://ndt-seattle.abilene.ucaid.edu:7123	GigE	http://ndt-seattle.abilene.ucaid.edu:7123/admin.html
Sunnyvale	http://ndt-sunnyvale.abilene.ucaid.edu:7123	GigE	http://ndt-sunnyvale.abilene.ucaid.edu:7123/admin.html
Washington DC	http://ndt-washington.abilene.ucaid.edu:7123	GigE	http://ndt-washington.abilene.ucaid.edu:7123/admin.html



Other Available Servers:

Location	Host
ANL	http://miranda.ctd.anl.gov:7123/
Swiss Education and Research Network (Switzerland)	http://cemp1.switch.ch/network/performance/web100/tcpbw100.html
University of Michigan (Flint)	http://speedtest.umflint.edu/
UCal Santa Cruz	http://nitro.ucsc.edu/
Thomas Jefferson National Accelerator Facility	http://jlab4.jlab.org:7123/
Stanford University	http://netspeed.stanford.edu/
NSF (Arlington, VA)	http://ciseweb100.cise-nsf.gov:7123
University of Hawaii (Honolulu)	http://farnsworth.uhnet.net:7123/

Figure 1.7. Publicly-accessible NDT servers

Examples and Observations

After running a public server for several years, here are several observations:

- Changing desktop effects performance
- Identifying faulty hardware is key to many performance problems
- Mathis et.al formula occasionally fails

First, it is important to test to the users desktop; having the network staff show up with a ‘good’ laptop doesn’t help much. As an example, during early testing one laptop client with a 10 Mbps Ethernet NIC saw 7 Mpbs (70% utilization), which is good for a half-duplex connection. Some timeouts and retransmission occurred (probably due to the half-duplex nature of the link). When informed of this loss, the network admin arrived with a ‘tuned’ laptop and ran a test with a 100 Mbps NIC, finding good throughput (85% utilization) and no loss. His conclusion was that there was no network problem to report. What is the typical user experience? “I see a problem but the network staff says no problem was found.” This situation leads to finger pointing and bad feelings because the user doesn’t get a solution and is told to live with the perceived problems.

Figure 1.8 shows another example of transient problems in a lab setup where 12 desktop computers were connected to the same Cisco switch, with two vLANs and a Cisco router between them. (These tests are vLAN to vLAN, e.g., through the router, and, in the first four cases, everything is 100 Mbps full duplex.) There is an order of magnitude difference in RTT (Round Trip Time), a factor of four in speed differences, and no correlation between speed and RTT. Also, loss never reaches 1%. In the last two cases, one of the hosts was changed to a 10 Mbps link. Again, we see an order of magnitude change in RTT but speed remains constant and loss is below 1%.

100 Mbps FD

Ave Rtt	%loss	loss/sec	Speed	
5.41	0.00	0.03	94.09	Good
1.38	0.78	15.11	22.50	Bad NIC
6.16	0.00	0.03	82.66	Bad reverse
14.82	0.00	0.10	33.61	Congestion
10 Mbps				
72.80	0.01	0.03	6.99	Good
8.84	0.75	4.65	7.15	Bad NIC

Figure 1.8. LAN Testing Results

The specific conditions for each case are:

Case 1: Everything is operating normally, with 100 Mbps full duplex links.

Case 2: The router had a bad interface module, and it was reporting these errors in the router logs, note loss/sec rate. However, a typical end-user would not be allowed to examine the routers statistics so would be unaware that this problem existed.

Case 3: In this case, the TCP traffic is flowing in the opposite direction but the bad router interface is still present. (Who would report a problem?)

Case 4: Three pairs of hosts are testing at the same moment, causing congestion on the shared router links (should be reported as normal).

Case 5: One of the hosts is set to 10 Mbps (normal operation).

Case 6: The faulty router interface is again in the path. Note the increased loss/second rate, but speed is still good.

Imagine what happens with GigE-attached servers and FastE attached clients. Would anyone complain?

Occasionally the Mathis (et al) formula fails. This formula describes the relationship between throughput, packet size, RTT, and loss rate. It is expressed as:

```
Estimate = (K * MSS) / (RTT * sqrt(loss))
```

- old-loss = (Retrans - FastRetran) / (DataPktsOut - AckPktsOut)
- new-loss = CongestionSignals / PktsOut

This formula describes the normal operating mode for a Reno TCP connection. In a normally operating network, the following condition should hold:

```
Estimate < Measured  (K = 1)
```

- old-loss 91/443 (20.54%)
- new-loss 35/443 (7.90%)

As noted, the NDT server is reporting that some connections don’t conform to this model. It isn’t clear why this discrepancy exists.

Installation Guide: How-To Setup your Own NDT Server

Components

NDT has two major components:

- Web100-based server programs
  - Testing and analysis engine
  - Optional 'lite' web server
- Two different client programs
  - Java Applet-based client that requires JVM browser plugin installed in client
  - Command line-based client that requires executable for specific client operating system

These are the NDT components needed to run a server. Everything is contained in a single downloadable tar file. Files are stored on the Internet2 End-to-End Performance Initiative (E2Epi) web site at: http://e2epi.internet2.edu/ndt/download.html.

Hardware Requirements

The NDT system does not place a tremendous set of demands on a host. The basic question is: can it operate in the environment specified by the network administrator? If the primary client is located on a campus with an Ethernet, WiFi, or Fast Ethernet network connection, then a low-powered server would suffice. If the primary goal is to serve Gigabit Ethernet-connected hosts, transcontinental, or international clients, then a more powerful host will be required.

Minimum requirements for a campus only server are:

- 500 MHz Intel or AMD CPU
- 64 MB of RAM
- Fast Ethernet

If you can purchase a new or better machine, an optimal server for the future would be:

- 2 GHz or better processor
- 256 MB of RAM
- Gigabit Ethernet

Disk space is needed for executables and log files, but there is no disk I/O involved during testing. Thus, any disk capable of holding the basic Linux operating system would suffice for a basic NDT server.

Software Requirements

Web100 enhancements that are needed are:

- Linux kernel
- User library

Other third-party software needed to compile source includes:

- Java SDK
- pcap library
- Client uses Java JRE (beware of versioning issues)

And, of course, the NDT source file – test engine (web100srv) requires root authority.

Recommended Settings

There are no settings or options for the Web-based java applet. It allows the user to run a fixed set of tests for a limited time period. The command line client does support the setting of TCP buffer size, allowing the user to experiment with various settings.

The test engine has numerous options that allow the administrator to control many functions. The basic options that will improve the usability of the system include:

- Turn on admin view (-a option)
- If multiple network interfaces exist, use the –i option to specify correct interface to monitor (ethx)

And, for the simple Web server (fakewww), it is recommended that you use the –l fn option to create a log file.

Potential Risks and Alternatives

Note that a non-standard kernel is required and the GUI tools can be used to monitor other ports. Also, public servers generate trouble reports from remote users. The owner of an NDT server must decide whether to respond to trouble reports or simply ignore emails. Another potential risk is that test streams can trigger IDS alarms – the solution to this is to configure IDS to ignore the NDT server.

Alternatives include other tools that can perform client testing, such as:

- Several web sites offer the ability for a user to check PC upload/download speed.
- Internet2/Surfnet Detective
- NCSA Advisor

Building the Server

Build basic Linux system using your favorite Linux distribution. Then, obtain the Web100 files from the Web100.org web site:

- http://www.web100.org/download
- Pre-built kernel in rpm format
- Patch file in tar format
- System Library and Utilities package in tar format

Download the latest version of the pre-built kernel or patch file *and* userland library files.

Choosing a Kernel

When choosing a kernel, the user has two basic choices: using a pre-built kernel or building a custom kernel.

Using a Pre-Built Kernel:

Install the pre-built kernel rpm (RedHat package Mmanager):

- rpm –i kernel-web100-*version*.rpm

Modify the boot loader config file:

- /etc/grub.conf or /etc/lilo.conf

Reboot the system and test the new kernel.

Build a Custom Kernel:

Download the base kernel from the kernel.org ftp site:

- ftp ftp.kernel.org and change directory to the proper Linux kernel directory

Unpack the kernel tar file in the /usr/src directory, unpack the Web100 kernel patch, and apply the patch:

- patch –p3 < *path-to-kernel-patch-file*

It is important to note that kernel patch files are keyed to a specific version of the Linux kernel. You must look at the Web100 kernel readme file to determine which kernel version you need to download/patch.

Configure and build the new kernel; note: you must enable "*Prompt for development code …*" so that Web100 options appear under "*Networking Options.*" Modify the boot loader configuration file, reboot, and test the new system.

[Note that the Linux configuration procedures are beyond the scope of this paper. If the administrator isn't familiar with them, then the pre-built kernel is a better choice. Also, the kernel configuration menu and build procedures for 2.4.x kernels are different from 2.6.x kernels.]

Building the Web100 Library

Unpack the Web100 user library tar file. Use the standard GNU automake commands:

- Change directory to package directory
- Create local make files (./configure {--*prefix=dir*})
- Build library and utility files (make)
- Install library and utility files (make install)

Once the kernel is built and running, it will automatically begin collecting data on every TCP connection to/from the server. The user library file contains the routines needed to extract that kernel data from the system. The administrator can verify that the kernel is recording properly by using the /usr/local/bin/gutil program. This X-windows based GUI allows the administrator to monitor any TCP connection going to/from the server.

Obtaining the Java Software Development Kit (SDK)

The administrator must download and install a Java SDK, which is required to build the applet client code. The SDK contains the Java compiler (javac) and archive builder (jar) programs. These executable programs need to be in the path or the NDT build process will fail. Since the administrator can put the SDK files anywhere on the system (there is no default path), the administrator must do this manually.

Download SDK from the Sun Microsystems web site (http://java.sun.com). Versions 1.2.2, 1.3.1, and 1.4.2 have been tested with NDT. Note: version 1.4.2 may cause old JRE clients to fail. Follow the package instructions to install the SDK file. Once installed, add the SDK *bin* directory to the path:

- export PATH=$PATH:/sdk/path/bin

Obtaining Libcap Library

The pcap library provides raw access to the network interface. The NDT uses packet pair dispersion techniques to determine the bottleneck link speed/type. This means that the libpcap.so runtime libraries must be installed on the NDT server. This library is a standard part of all Linux distributions. The administrator should check the NDT server to ensure they were installed when the system was built. The command:

```
ls /usr/lib/libpcap*
```

– will return several files if the libraries are installed. If they are not installed, obtain the rpm from your favorite mirror site or download and install the source from the http://www.tcpdump.org web site.

Building the NDT Programs

Once all the pre-requisites are installed, the admin is now ready to create the NDT system. Simply grab the latest tarball at http://e2epi.internet2.edu/ndt/download.html, unpack the tar file, and use the standard GNU automake tools to create and install the executables:

- Change directory to package directory
- Create make files (./configure {--*prefix=/some/dir*})
- Build executables (make)
- Install executables (make install)

This process creates both server programs (web100srv and fakewww), the client tools (analyze, web100clt), and it builds the java class and jar programs. The make install process puts all the executables in the proper place.

Customizing the Installation

Once the executables are installed, the base NDT web page needs to be built. The tarball contains a template and a simple shell script to aid in the creation of a customized web page. To create a custom NDT web page:

- Run the script ./conf/create-html.sh and follow the prompts
- Script will prompt for installation directory, use *prefix*= value and append /ndt

Use the conf/start.ndt script as sample for starting server processes. The conf/ndt script can be copied into the /etc/init.d directory for boot time startup (see chkconfig man page on RedHat systems for more details). Once the base web page is created, you are ready to start the server processes. The scripts provided will simplify the start process.

Customizing the Server

The admin has to make a choice: run the included web server (fakewww) or a full blown web server (apache). The start scripts assume you will start the included (fakewww) server. This server contains a list of file names that it can return to the client. This reduces the vulnerability of the system and prevents the server from processing requests from malicious users. The log file is useful to help monitor what is being requested.

Some useful options for the included web server (fakewww) are:

- Set an alternate port number (-p80)
- Run in Federated mode (-F)
- Log web requests (-l logfile)

The test and analysis engine (web100srv) must be operating for the system to work. The user will get an error message if this process is not running. Note that the server uses three TCP ports to run these tests. The administrator is responsible for ensuring that any site firewall or router filter is configured to pass these ports. The defaults are 3001, 3002, & 3003. Some useful options to use with the testing engine (web100srv) are:

- Generate basic usage information (-a)
- Use fixed configuration file (-c)

Note that both server program and the command line client accept debugging (-d) flags. Multiple levels of debug messages are available and levels increase when multiple –d flags are specified (e.g., -ddd specifies levels 0 to 2). In addition, the server programs will respond to the help (-h) flag by displaying basic usage information.

Creating a Federated Server

In some cases, it is beneficial to run multiple NDT servers in a collaborative manner. As noted above, the NDT server can identify both configuration and performance problems. The assumption is that configuration problems are more serious than performance problems and they usually occur near the client host. This means that testing to the 'closest' NDT server makes finding configuration problems easier. Operating a collection of NDT servers in Federated mode accomplishes this task.

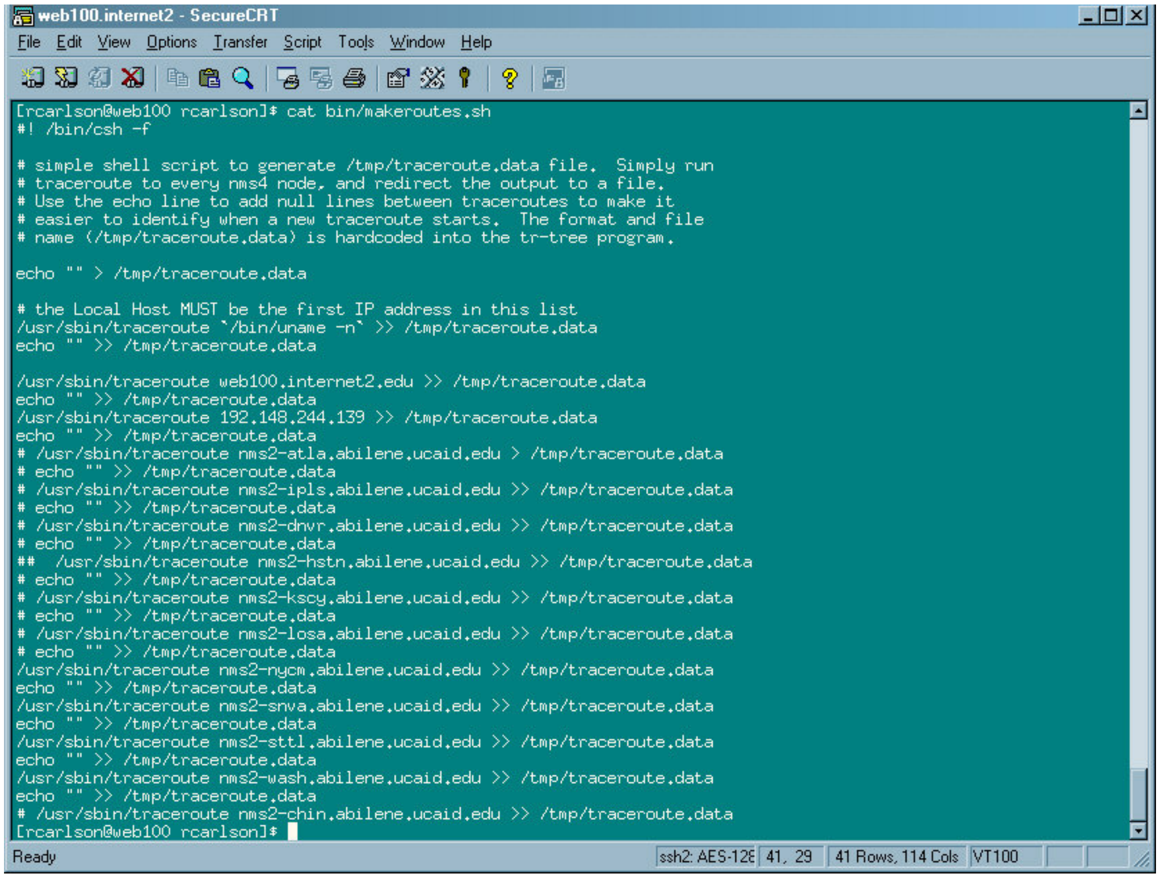
Simply run traceroute form each NDT server to every server in your federation:

- Save output in file (/tmp/traceroute.data)
- Figure 2.1 shows a sample script
- Note: blank line is required between traces

Once this data file is created it can be turned into a binary tree file:

- /usr/local/bin/tr-mkmap –b {-f fn}

Note: You may create a cron job to automate this process!


web100.internet2 - SecureCRT
File Edit View Options Transfer Script Tools Window Help
Ready
ssh2: AES-128 41, 29 41 Rows, 114 Cols VT100

```
[rcarlson@web100 rcarlson]$ cat bin/makeroutes.sh
#! /bin/csh -f

# simple shell script to generate /tmp/traceroute.data file.  Simply run
# traceroute to every nms4 node, and redirect the output to a file.
# Use the echo line to add null lines between traceroutes to make it
# easier to identify when a new traceroute starts.  The format and file
# name (/tmp/traceroute.data) is hardcoded into the tr-tree program.

echo "" > /tmp/traceroute.data

# the Local Host MUST be the first IP address in this list
/usr/sbin/traceroute `/bin/uname -n` >> /tmp/traceroute.data
echo "" >> /tmp/traceroute.data

/usr/sbin/traceroute web100.internet2.edu >> /tmp/traceroute.data
echo "" >> /tmp/traceroute.data
/usr/sbin/traceroute 192.148.244.139 >> /tmp/traceroute.data
echo "" >> /tmp/traceroute.data
# /usr/sbin/traceroute nms2-atla.abilene.ucaid.edu > /tmp/traceroute.data
# echo "" >> /tmp/traceroute.data
# /usr/sbin/traceroute nms2-ipls.abilene.ucaid.edu >> /tmp/traceroute.data
# echo "" >> /tmp/traceroute.data
# /usr/sbin/traceroute nms2-dnvr.abilene.ucaid.edu >> /tmp/traceroute.data
# echo "" >> /tmp/traceroute.data
##  /usr/sbin/traceroute nms2-hstn.abilene.ucaid.edu >> /tmp/traceroute.data
# echo "" >> /tmp/traceroute.data
# /usr/sbin/traceroute nms2-kscy.abilene.ucaid.edu >> /tmp/traceroute.data
# echo "" >> /tmp/traceroute.data
# /usr/sbin/traceroute nms2-losa.abilene.ucaid.edu >> /tmp/traceroute.data
# echo "" >> /tmp/traceroute.data
/usr/sbin/traceroute nms2-nycm.abilene.ucaid.edu >> /tmp/traceroute.data
echo "" >> /tmp/traceroute.data
/usr/sbin/traceroute nms2-snva.abilene.ucaid.edu >> /tmp/traceroute.data
echo "" >> /tmp/traceroute.data
/usr/sbin/traceroute nms2-sttl.abilene.ucaid.edu >> /tmp/traceroute.data
echo "" >> /tmp/traceroute.data
/usr/sbin/traceroute nms2-wash.abilene.ucaid.edu >> /tmp/traceroute.data
echo "" >> /tmp/traceroute.data
# /usr/sbin/traceroute nms2-chin.abilene.ucaid.edu >> /tmp/traceroute.data
[rcarlson@web100 rcarlson]$
```

Figure 2.1. Simple Traceroute Script

Verifying the Operation

The basic checks an administrator can perform to ensure that everything is operating properly include:

1) Check the process table
   - ps auxw | grep fakewww
   - ps auxw | grep web100srv
2) Check the TCP port status
   - Fakewww = netstat –nat | grep 7123
   - Web100srv = netstat –nat | grep 300
     - ➢ Note: ports 7123 & 3001 in listen state
3) Make sure the server is monitoring the correct network interface. Note that at the current time the web100srv process needs to know which interface to monitor. If the server has multiple interfaces, then the admin needs to specify the primary interface and all test traffic must go through this interface. Check interface for link detection:
   - netstat –nr

Building a Custom Command Line Client

If desired, you can build a custom command line client for multiple operating systems:

- Download and unpack the latest NDT package
- Run the automake ./configure command
- Change directory to the *src* subdirectory and build the client (make web100clt). [Note: Web100 lib or kernel not required.]

This is an optional step. The administrator only needs to perform this step if they want to run the command line client on another host. You need to change to the src sub-directory and invoke make with the web100clt argument. If you don't then the make will fail because it wouldn't find the Web100 libs needed for the server program. NDT has been built on several different hosts (FreeBSD, IRIX, Mac OS-10, and Cygwin), so it should work on others.

Additional Features

There are man pages for all the server programs. The analyze program can read/parse the log file and report what's been happening. Finally, the server programs will log all activity so the administrator can monitor it.

Obtaining the Test Results

Two 10-second tests, one in each direction, are run between the Client and Server. No diagnostic data is collected. Then, it runs a 10-second test from Server to Client; Web100 diagnostic data is collected at the end of test. Last, it prints out a summary status message:

- Link speed and duplex
- Informational or Warning messages

Note that you don't get a lot of good data when the server is receiving the data stream. After the tests have finished, a brief summary is printed stating what the bottleneck link is and a guess as to what duplex condition (full or half) exists. Finally any major problems, such as duplex mismatch, are reported.

Analyzing the Test Results

When the user clicks the "statistics" button a pop-up window is created. This window holds some additional details:

- Send and Receive throughput achieved
- Details for five configuration tests (link type, duplex mode, congestion, excessive errors, and duplex mismatch condition)
- Throughput limits section (%S-R-N limited, RTT, %loss, %out-of-order)
- Negotiated settings (TCP modifications to improve performance)

Such information as the percentage of time the connection spend in one of the Web100 triage states (sender limited, receiver limited, or network limited) is reported. It also reports loss, out-of-order, RTT, and MSS values. Finally, the TCP options that can be enabled/disabled are listed. Note that these are the negotiated values, not the configured values. A middlebox might disable one or more of these options and, right now, NDT can’t determine if that happened.

The “more details” button generates another pop-up window. This window contains a listing of all the Web100 and derived variables used by the server to perform the analysis. The client program simply prints out the pre-computed answers. It also provides some performance tuning Information:

- Individual TCP counters collected by Web100
- Conditional test parameters
- Throughput analysis section including theoretical limits, bandwidth*delay products, loss rate, and buffer sizes

The “report problem” button provides a simple way to send in a trouble report. When this button is pressed, the applet creates an email message, via the clients default email program. The content of this email is automatically filled in with the test results. Hopefully, the user will annotate this email to say exactly what the trouble is. Lastly, the admin can look at the server log files to ensure that no operational problems exist.

The server logs all counter variables used for condition tests.