Developing the Web100 Based Network Diagnostic Tool (NDT)

By
Richard A Carlson
RACarlson@anl.gov
Argonne National Laboratory[*] – Electronics and Telecommunications Division

Abstract

Finding and fixing a network performance or configuration problem is a difficult and complex task. It requires time, energy, and expertise to track down problems and it is difficult for users to know whom to contact when the source and destination computers are located in different administrative domains. This paper describes a novel new way of performing the basic diagnostic function needed to identify that a problem exists and if that problem is in the source computer, the destination computer, or the network infrastructure. Once a problem is identified, users will have detailed data that can be used by their network administrator to find a solution.

Introduction

The number one question network operators hear today is "my application is slow, what's wrong with the network?" To answer this question, the general solution is to install some specialized packet sniffing hardware/software, spend lots of time gathering data, and then find someone with the engineering expertise required to analyze the application's traffic. This complexity means that most performance problems, as opposed to connectivity problems, remain an unsolved mystery.

To address this problem, we have developed a novel new Network Diagnostic Tool (NDT) that uses a Web100 [WEB100] based approach to perform this basic investigative function. The basic premise is that by combining the Web100 data with measured TCP throughput data we can identify:

1. when normal network congestion limits throughput
2. when a network link is set to half duplex
3. what the bottleneck link is (Dial-up to 10 Gigabit Ethernet)
4. that a duplex mismatch condition exists
5. that a faulty cable or NIC is corrupting packets

This paper describes a set of experiments that test and validate this Web100 based approach to performance troubleshooting. We have successfully identified and documented 'network signatures' for the 5 conditions outlined above.

A command line version of the web based NDT has also been developed. The command line version is based on a modified version of the widely available NLANR Iperf [IPERF] tool. This command line version allows network administrators to repeatedly schedule tests to be run automatically to continuously monitor a specific network path. This paper will focus on the web based NDT, recognizing that the modified Iperf will perform the same diagnostic functions.

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

The initial work was done on a local network testbed. This testbed was created by connecting 13 PC's running RedHat 7.2 Linux with a web100 modified 2.4.9 kernel to a Cisco Catalyst 5500 switch. Each PC was equipped with an Intel 450 MHz PIII processor, 128 Mbytes of RAM, a 4 GB harddrive, and a 3Com 3c905B 10/100 Network Interface Card. The Catalyst 5500 switch was configured with 2 vlans with each PC connected to one of them.   A Cisco 7505 router was use to interconnect the multiple vlans and to provide access to the testbed from external machines.

One example of the how useful this approach is. During the base level testing it was noted that one of the PC's in the testbed had noticeably lower throughput than the other twelve PC's (84 Mbps vs. 94 Mbps). Taken in isolation this throughput value (84 Mbps) would seem to indicate that the network is operating properly, but taken as one value in the set of thirteen, it stands out as an obvious hidden problem. An investigation revealed that the Cat-5 drop cable that connects this PC to the switch was bad, causing enough errors to impact throughput. This fault would have been extremely difficult to find without the benefit of the additional diagnostic data provided by the Web100 KIS variables.

Following these initial tests, the web based NDT program was deployed on a globally accessible server located at the author's home institution. The initial 'network signatures' were refined to take into account the greater variety of hosts. This NDT has been operational for approximately 9 months. Figure 1 shows the utilization growth for this public server. Figure 2 shows the utilization by Internet top level domains (TLD's).

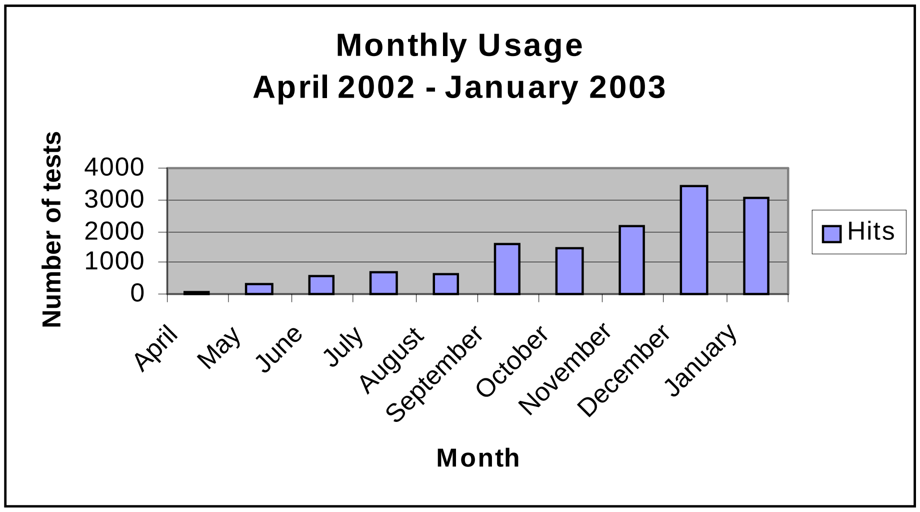

Figure 1 – NDT Utilization growth by external clients

The rest of this paper describes how the NDT operates and the specific detection algorithms used to identify the 5 major network conditions described above. This will be followed by a comparison between this Web100 diagnostic approach and other traditional measurement approaches. Finally the future direction for the NDT development will be briefly discussed.

NDT operation

The Web100 project provides a Linux kernel instrument set (KIS) patch to capture numerous TCP kernel variables (e.g., Round Trip Time (RTT), Congestion Window (CWND))[1]. These variables are then accessible from user applications via a set of user library functions. Thus the KIS patch captures TCP variables in real time and the user library allows retrieval of this information at a later period in time.

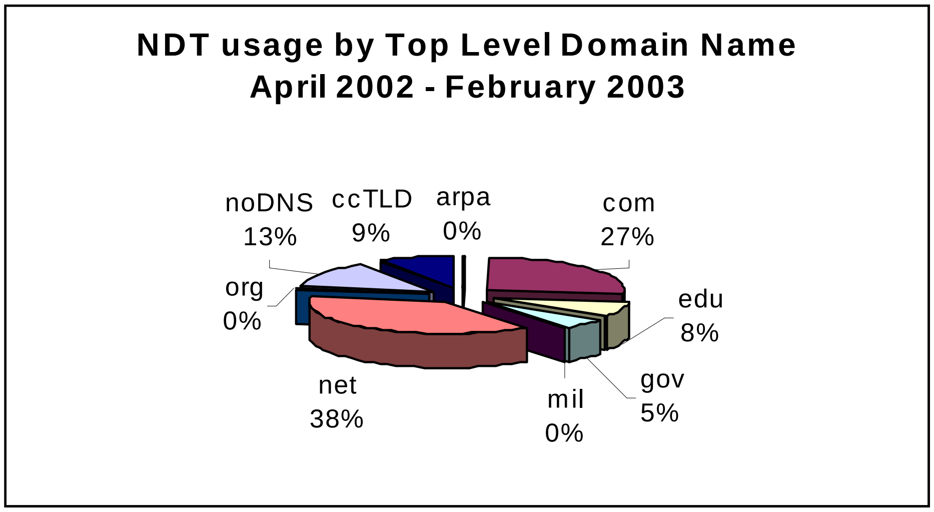

Figure 2 – NDT Utilization by Top Level Domains

The NDT is a Web-based client-server tool that was developed to quickly and easily identify desktop computer and network infrastructure problems. It uses 3 major components, described below, to generate and capture TCP traffic and the associated KIS variables. It then analyzes the captured data to determine what limited the connection throughput.

The first component is a simple web server application that accepts incoming connections and returns a simple web page with an imbedded Java applet. This application then returns to a listen state ready to receive new connection.

The second component is a server application that performs bi-directional throughput tests between the server and the client. The exact test routine is described below. This application accesses KIS variables using the user library routines and returns this data to the client for analysis.

The third component is a Java applet that is automatically downloaded to the client's desktop computer. This applet communicates with the server's throughput application to perform the bi-directional throughput tests. It then combines the test results with the Web100 data to print out a series of detailed diagnostic messages. It also provided the user with a simple point and click interface to their email client, allowing the results to be sent to the appropriate administrator.

After the applet has been downloaded the program waits for the user to click the `start` button on the web page. At this point the applet contacts the server application and

[1] Appendix A contains a complete listing of the kernel variables retrieved and used by the NDT

requests a new test begin. The server application forks a child process to handle the test and returns to a listen state. If a test is already in progress the child process fails and the client is notified that a test cannot begin yet. The user must click the `start` button again to request a new test.

Assuming that no other test is in progress, the child process initializes successfully and the server puts two (2) new TCP sockets into the listen state. The server then notifies the client that testing can begin. The client starts by opening and closing a connection on one (1) of the listening sockets. This test performs a simple NAT detection test by having the server return the socket peer data associated with this connection. More work on middlebox detection will be incorporated in future releases.

After this test completes the server informs the client that it can begin a data streaming throughput test. The client opens a new connection to one (1) of the listening server sockets and streams data to the server for ten (10) seconds. At the completion of this test, the server and client save the test results. At this point it is possible to retrieve some KIS variable data from the server. Unfortunately, the data collected at this time has not proved to be very useful. The problem is not that the data is missing, but that a receiving TCP does not record interesting data. For example, a receiving TCP does not calculate an RTT value for the connection. The receiver side KIS variables are still being reviewed to determine if useful data can be extracted for diagnostic purposes.

After this throughput test completes, the server informs the client to begin a second throughput test. The client connects to the second listening socket and the server streams data to the client for ten (10) second. At the completion of this test, the KIS variables for this connection are retrieved and transmitted back to the client. The server process then runs through the detection algorithms, described below, and writes the results out to a log file. This log was used to generate the figures and tables presented in this paper.

Finally the client applet runs though the same detection algorithms and presents the results to the user. These results are broken down into 4 presentation screens that allow the user to pick and choose the amount of information they wish to view.

The first screen is the main applet page. This screen shows the basic throughput test results, and a summary statement of what the path's bottleneck link is. If a major configuration error, duplex-mismatch or hardware faults, is detected this information is reported instead of the bottleneck link type.

The second page appears when the user clicks the `Statistics` button. This action brings up a new window that contains additional connection information. The throughput values are repeated followed by a summary of what the 5 detection algorithms reported. This is followed by the packet size, RTT, and packet loss rate data. The connection (sender buffer, receiver buffer, or network congestion) state data is then presented to let the user know where the connection is spending its time. The bandwidth*delay product is also calculated at this time and the user is notified if an increase in buffer size would likely improve the throughput.

The third page appears when the user clicks the `More Details` button. This action brings up a new window that contains a listing of all the KIS variables captured by the server process. The variable name and its associated value are displayed. This is followed by the connection throughput limits based on the Mathis [MM97] formula and the bandwidth*delay product for the sender, receiver and congestion window buffer sizes.

The `Report Problem` button is designed to make it easy to forward test results to a network administrator. Clicking on this button will bring up the users email program. A new email message will be created and test data will automatically be input into the body of the message. The network administrator's email address is supplied by the server html page allowing this data to be customized by the NDT administrator.

Network administrators from Universities, National and international Research and Education, National Laboratories and corporations interested in understanding how their local networks are operating are encouraged to install their own NDT server. The first step in bringing up an NDT server is to build a Linux based PC with an appropriate, Fast or Gigabit Ethernet, network interface card (NIC). Next download and install the Web100 kernel and user library tar files from the Web100 web site[2]. Then download and install the NDT source files from the ANL anonymous FTP server[3]. Configure the NDT administrator's email address in the `tcpbw100.html` file and use the `start.www` shell script to bring up the server side applications.

We now describe the 5 detection algorithms used by the NDT.

Duplex Mismatch

Duplex mismatch conditions cause major problems for Internet users and network administrators. According to a NASA study [CL01] nearly 60% of the network trouble tickets were generated because of a duplex mismatch condition. Finding an easy and reliable way to identify this condition is an important part of this project.

At first glance, it would seem obvious that this condition could be detected by the low measured throughput. Our tests indicate that this is not the case. Low or asymmetric throughput results can be caused by many factors. A common example of asymmetric throughput is the common cable modem configuration. It is common to see transmit and receive throughput values differ by an order of magnitude when clients are connected to a normally operating cable modem.

In addition our test results showed that desktop clients that were connected to a standard 10 Mbps Ethernet link were unable to measure abnormal throughput values when the server is connected to a 100 Mbps Fast Ethernet link. As shown in Table 1, the client measured nearly 7 Mbps independent of the server's duplex settings. Thus in most cases clients connected to Ethernet links would not see or report a problem, while clients or other servers connected to Fast Ethernet links would see poor performance. Without a lot of detailed knowledge of what all the network paths looked like the network administrator would be hard pressed to understand why some users were complaining and others weren't.

Table 1 shows why duplex mismatch faults are so difficult to find. In case 1, the client was attached to the network via a 10 Mbps half-duplex connection while the NDT link

[2] http://www.web100.org
[3] ftp://achilles.ctd.anl.gov:/pub/web100/web100-tools.tar.gz

had a duplex mismatch with the server NIC set to 100 Mbps full-duplex and the switch port set to 100 Mbps half-duplex. In case 2 the client's connection is identical but the duplex mismatch condition is reversed with the server NIC now at 100 Mbps half-duplex and the switch full-duplex. In case 3 and 4, the client is still on a 10 Mbps half-duplex link while the NDT NIC and switch port agree on speed and duplex settings 100 Mbps full-duplex or half-duplex respectively. In all cases the measured throughput remained constant.

MaxSsthresh	TransRwin	TransCwnd	TransSender	Throughput	Description
31856	1	11	11	6.97	10/HD --> H:100/FD; S:100/HD
31856	1	12	12	6.96	10/HD --> H:100/HD; S:100/FD
0	1	6	6	6.98	10/HD --> 100/FD
0	1	7	7	6.98	10/HD --> 100/HD

Table 1 - Duplex mismatch condition

An examination of the data collected by the NDT produced a reliable indicator for this mismatch condition. The logical AND of the following 4 conditions will identify this fault condition:

1. The connection spends over 90% of its time in the congestion window limited state. The KIS identifies 3 unique connection limited states. These are sender buffer full, receiver buffer full, and congestion window limit. The KIS records the amount of time the connection spends in each of these states.
2. The estimated bandwidth over this link is less than 2 Mbps. The estimated bandwidth is proportional to RTT and inversely proportional to packet loss [MM97]. This value is calculated using the measured RTT, MSS, and packet loss rates.
3. There are more than 2 packets being retransmitted every second of the test. This value is calculated using the measured number of retransmissions and the total test time.
4. The connection experienced a transition into the TCP slow-start state.

.

The above detection algorithm will determine when a desktop client link contains a duplex mismatch condition. However there is another case where a client is attached to an office hub/switch and this office hub/switch is then attached to a building switch. This building switch is then attached to a router port for connection to the campus network. Testing showed that the above algorithm failed to detect a duplex mismatch condition in the office hub/switch to building switch up-link. To detect this fault condition logically AND the results of these 4 tests:

1. The measured client to server throughput rate exceeded 50 Mbps. The measured rate is the total number of bits divided by the test time.
2. The measured server to client throughput rate is less than 5 Mbps.
3. The connection spent more than 90% of the time in the receiver window limited state.
4. There is less that 1% packet loss over the life of the connection.

Hardware Fault

Once it is determined that a duplex mismatch condition does not exist, it is possible to determine if a hardware fault was impacting application throughput. Most Fast Ethernet networks utilize Cat-5 class copper wires to interconnect hosts, switches, and routers. A damaged cable or connector can cause intermittent problems that can be difficult to identify. In addition though rare, host network interface adapters fail and packets are corrupted.

Since throughput is not a good indicator that a problem exists, the KIS variables were examined to determine which ones could be used to identify this problem. An analysis showed that this condition can be identified by logically AND the results of these 4 tests:

1. The connection is loosing more than 15 packets per second. This value is calculated by dividing the number of packet loss events by the total test time.
2. The connection spent over 60% of the time in the congestion window limited state.
3. The packet loss rate is less than 1% of the packets transmitted. This value is calculated by dividing the total number of packets transmitted by the number of retransmitted packets. While the connection is loosing a large number of packets per second (test 1) the total number of packets transferred during the test is extremely small so the percentage of retransmitted packets is also small value of packet loss rate.
4. The connection entered the TCP slow-start state.

Faulty Bandwidth Estimation

As was noted above, we calculate an estimated bandwidth using the Mathis formula described in [MM97]. Our link detection algorithms rely on this formula being correct. If this assumption is false then we will be unable to correctly identify what type of network link technology was used to build this path. Thus if the estimated bandwidth is less than the measured throughput or assumption is false and we disable the link detection algorithms.

Network Link Speed

After determining that no major faults exist, we are left with the difficult task of determining what type of networks links (e.g., DSL/Cable, Fast Ethernet) are used to build the network path between the client and NDT server. At the current time we examine the entire end-to-end path to determine the bottleneck link type. Future enhancements will determine where in the path this bottleneck link is, but for now we simply report what the bottleneck link type is.

This task is difficult because we have no history on which to base our decisions. A low throughput result could be the result of a slow link type (i.e., cable modem), a highly congested shared high speed link (i.e., Fast Ethernet), or a large bandwidth*delay product due to a long RTT between the client and NDT server. It is important that the end user and the network administrator can differentiate between these different conditions. It helps the user by setting realistic expectations, a cable modem will not allow high throughput rates. It also helps network administrators by identifying slow or congested links in the network infrastructure.

To accomplish this task, the NDT starts with the assumption that the end-to-end path is a series of 100 Mbps full-duplex Fast Ethernet links. The NDT then uses several link detection algorithms to validate or disprove this assumption. This approach has show good results when the clients are located near the NDT server (i.e., small RTT values)[4]. Three separate link detection tests (described below) are run to disprove the base assumption.

[4] As note below, new link detection algorithms are being developed to overcome this deficiency.

Our first task is to determine if the path contains a 10 Mbps Ethernet bottleneck link. A normally operating Ethernet link can be identified by the logical AND of the following 6 tests:

1. The measured client to server throughput is less than 9.5 Mbps. This value is calculated by dividing the number of bits transferred over the total test time.
2. The measured client to server throughput is greater than 3 Mbps.
3. The measured server to client throughput is less than 9.5 Mbps.
4. The packet loss rate is less that 1%.
5. Less than 35% of the packets are reordered by the network. Packet reordering is calculated by subtracting the number of retransmission from the number of duplicate acks received. The total number of packets sent is then divided by this result[5].
6. The estimated bandwidth is greater than the measured throughput.

Our next task is to determine if the path contains an IEEE 802.11b wireless link. A normally operating wireless link can be identified by the logical AND of the following 6 tests:

1. The connection never spends any time in the send buffer limited state. This is one of the connection limited states recorded by the KIS variables.
2. The measured client to server throughput is less than 5 Mbps.
3. The estimated bandwidth is greater than 50 Mbps.
4. The connection transitions between the receiver buffer limited state and the network congestion limited state an even number of times. In addition to tracking how much time a connection spends in the 3 limiting states, it also tracks how many times a connection transitions between these states.
5. The connection is receiver buffer limited over 90% of the time. Even though the connection transitions between the 2 stated noted in test 4, it spends the majority of it's time in the receive buffer limited state.
6. The estimated bandwidth is greater than the measured throughput.

Our last task is to determine if the path is limited by a Digital Subscriber Line (DSL) or cable model link. A normally operating DSL/cable modem link can be identified by the logical AND of the following 3 tests:

1. The connection never enters the Send buffer limited state.
2. The measured client to server throughput is less than 2 Mbps.
3. The estimated bandwidth is greater than the measured throughput.

Full or Half Duplex link

The next task is to determine if a half-duplex link exists in the end-to-end path. Half-duplex links decrease throughput by 20 – 30% due to their need to enforce the link idle time described in the IEEE 802.3 CSMA/CD protocol. As noted above, the KIS variables track three state conditions for the TCP connection. The NDT uses the state transition counters to identify a half-duplex condition.

A Fast Ethernet half-duplex link can be identified by connection that toggles rapidly between the sender buffer limited and receiver buffer limited states. In tests on an un-congested path, these values exceeded 100 transitions per second. However, even

[5] In the authors' experience, a large number of reordered packets indicate a 100 Mbps PCMCIA NIC in a laptop computer.

though the connection toggled into and out of the sender buffer limited state numerous times, it did not remain in this state for long periods of time. Over 95% of the time was spent in the receiver buffer limited state. The NDT identifies half-duplex conditions by a logical AND of the following 4 tests:

1. The connection is receiver buffer limited over 95% of the time.
2. The connection transitioned into the receiver buffer limited state over 30 times per second.
3. The connection transitioned into the send buffer limited state over 30 times per second.
4. The link detection algorithm reported the bottleneck link is Fast Ethernet.

Table 2 shows the results of the NDT half-duplex detection algorithm. In case 1, the path between the client and NDT server is build from full-duplex Fast Ethernet links. Note that throughput value reaches 94% of the link capacity, and the connection is receiver buffer limited. In case 2, the client was changed to a half-duplex condition. Note that the throughput drops to 70% of the link capacity and the connection rapidly transitions between the receiver buffer limited and send buffer limited states, while the connection spends the majority of the time in the receiver buffer limited state. In case 3 the NDT server was changed to a half-duplex condition and the client was returned to a full-duplex condition. The same half-duplex signature remains.

TransRwin	TransCwnd	TransSender	Rwin %time	Throughput	Description
1	6	6	99.77	94.06	100/FD --> 100/FD
3909	7	3915	98.49	68.66	100/HD --> 100/FD
3945	6	3950	98.48	69.80	100/FD --> 100/HD

Table 2 - Half Duplex link detection

Congestion

Finally the NDT looks for signs of congestion. It is obvious that a Fast Ethernet link shared by ten TCP streams will provide each stream with approximately 10 Mbps of throughput. However, a single TCP stream on an un-congested link may also report a 10 Mbps throughput due to a TCP sender/receiver buffer limit. The NDT congestion detection algorithm identifies when network congestion is the limiting factor. The NDT reports congestion by a logical AND of the following 4 tests:

1. The packet loss rate is less than 1%. A properly operating TCP stream will see very little packet loss even under congestion. This is due to the congestion control algorithms that cause TCP to lower the congesting window which limits the amount of data the sender can inject into the network.
2. The connection is network congestion limited more than 20% of the time.
3. A duplex-mismatch condition has not been detected.
4. The connection entered the TCP slow-start state.

As can be seen from the above descriptions, there is some dependency between the various NDT detection algorithms. Thus, a single test will not be able to tell if more than one (1) fault condition exists in the network path. Repetitive tests must be done to discover when multiple misconfiguration conditions exist. This strategy means that serious problems (e.g., duplex mismatch) are identified and reported before less serious ones (e.g., normal congestion). It is important to realize that these are stateless tests, in that current data is not compared to data gathered in previous tests. This condition makes the analysis a little harder, but it covers the general case where previous data is unavailable.

Comparison to other work

This work differs from other measurement projects Pinger [WM98], Surveyor [MZ99], NIMI [VP98]. Those projects seek to gather and save network measurement data that can be used by network engineers and administrators to discover network operating conditions and long term trends. This work seeks to allow end users to identify network performance that effect their individual applications.

While this work can identify if the network infrastructure is operating properly, it can not identify where in the path a faulty or slow link exists. To discover this type of information a path measuring tools like pathchar [VJ97], pchar, or clink [AD99] must be used. The analysis performed by this tool can be used to identify paths that need this detailed investigation.

Conclusion

This paper shows that many important network configuration and performance issues can be discovered through the examination of some important TCP kernel variables. The Web100 project has provided KIS patches to the Linux 2.4.9, 2.4.16, and 2.4.19 kernels that captures these variables for later analysis. It has also provided a user library to export these KIS variables to user applications.

A novel new Network Diagnostic Tool (NDT) was developed to exploit these KIS variables. The NDT runs a series of short, 10 second, throughput tests between a Java applet loaded onto a client and the NDT server. The measured results are combined with calculated values derived from the KIS variables to determine if the throughput bottleneck is in the NDT server, the client computer or the network infrastructure.

The NDT has meet with great success. Several U.S. universities have notified the author that they have installed and are using the NDT to diagnose campus network problems. The author has also been informed that national and international research and educational network administrators have found the tool useful in their environments. The author is currently working to deploy more NDT servers at ‘interesting’ locations throughout the Internet.

Efforts are also underway to improve the NDT detection algorithms. The author is currently experimenting with packet pair arrival timings to improve the link detection algorithms. These timing rates will be combined with KIS variables to determine what type of bottleneck link exists in the network path. In addition the number of detected link types is being increased to detect higher speed, OC-12 and higher, link technologies.

Appendix A. List of the current KIS variables collected and used by the NDT.

Variable Name	Description
AckPktsIn	Ack Packets In
AckPktsOut	Ack Packets Out
BytesRetrans	Retransmitted Bytes
CongestionSignals	Total Congestion Signals.

CountRTT	Count of RTT Samples included in SumRTT
CurrentCwnd	Current Congestion Window
CurrentMSS	Current Maximum Segment Size
CurrentRTO	Current Retransmission Timer
CurrentRwinRcvd	Current receiver window
CurrentRwinSent	Current receiver window out
CurrentSsthresh	Current Slowstart Threshold
DSACKDups	Duplicate Data DSACK Reports
DataBytesIn	Data Bytes received
DataBytesOut	Data Bytes sent
DataPktsIn	Total Data Packets received
DataPktsOut	Total Data Packets sent
DupAcksIn	Number of Duplicate Acks Received
FastRetran	Fast Retransmits
MaxCwnd	Maximum Congestion Window
MaxMSS	Maximum MSS
MaxRTO	Maximum Retransmission Timer
MaxRTT	Maximum RTT
MaxRwinRcvd	maximum receiver window
MaxRwinSent	maximum receiver window out
MaxSsthresh	Maximum Slowstart Threshold
MinMSS	Minimum MSS
MinRTO	Minimum Retransmission Timer
MinRTT	Minimum RTT
MinRwinRcvd	minimum receiver window
MinRwinSent	minimum receiver window out
PktsIn	Total Packets received
PktsOut	Total Packets sent
PktsRetrans	Packets With Retransmitted Data
Rcvbuf	Available receiver buffer memory
SACKEnabled	SACK Enabled
SACKsRcvd	Number of SACKs Options Received
SmoothedRTT	Smoothed RTT
Sndbuf	Available sender buffer memory
SndLimTimeRwin	Receiver Limited Time
SndLimTimeCwnd	Congestion Limited Time
SndLimTimeSender	Sender Limited Time
SndLimTransRwin	Receiver Limited Transitions
SndLimTransCwnd	Congestion Limited Transitions
SndLimTransSender	Sender Limited Transitions
SndLimBytesRwin	Receiver Limited Bytes
SndLimBytesCwnd	Congestion Limited Bytes
SndLimBytesSender	Sender Limited Bytes
SumRTT	Cumulative RTT
Timeouts	Number of Initial Timeouts
TimestampsEnabled	Timestamps Enabled
WinScaleRcvd	Requested window scale
WinScaleSent	Requested Window Scale Out

References

WEB100 W. Huntoon; Web100 concept paper; http://www.web100.org/docs/concept_paper.php; September 1999
IPERF A. Tirumala; J. Ferguson, Iperf, http://dast.nlanr.net/Projects/Iperf/;
MM97 M. Mathis, J. Semke, J. Mahdavi, T. Ott; "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm"; Computer Communication Review, volume 27, number 3, pp. 67-82; July 1997.
CL02 C. de Luna; personal communication; June 2002
WM98 W. Matthews, L. Cottrell; Internet Monitoring in the HEP Community; International Conferene on Computing in High Energy Physics 1998 (CHEP 98); September 1998
MZ99 M. Zekauskas, S. Kalidindi; Surveyor: An Infrastructure of Internet Performance Measrements; INET99; http://www.advanced.org/surveyor/; June 1999
VP98 V. Paxson, J. Mahdavi, A. Adams, M. Mathis; An Architecture for Large-Scale Internet Measurement; IEEE Communications 1998
VJ97 V. Jacobson; pathchar – a tool to infer characteristics of Internet paths; MSIR; ftp://ftp.ee.lbl.gov/pathchar/msri-talk.pdf; April 1997
AD99 A. Downey; Using pathchar to Estimate Internet Link Characteristics; SIGCOMM’99; http://rocky.wellesley.edu/downey/clink/; August 1999