Pathdiag: Automated TCP Diagnosis*

Matt Mathis[1], John Heffner[1], Peter O'Neil[2,3], and Pete Siemsen[2]

[1] Pittsburgh Supercomputing Center
[2] National Center for Atmospheric Research
[3] Mid-Atlantic Crossroads

Abstract. This paper describes a tool to diagnose network performance problems commonly affecting TCP-based applications. The tool, *pathdiag*, runs under a web server framework to provide non-expert network users with one-click diagnostic testing, tuning support and repair instructions. It diagnoses many causes of poor network performance using Web100 statistics and TCP performance models to overcome the lack of otherwise identifiable symptoms.

1 Introduction

By design, the TCP/IP hourglass [4] hides the details of the network and the application from each other. This property is critical to the ongoing evolution of the Internet because it permits applications and the underlying network infrastructure to evolve independently. However, it also obscures all network flaws. Since TCP silently compensates for flaws, for example by retransmitting lost data, the only symptom of most problems is reduced performance. This "symptom hiding" property was the motivation behind the Web100 project [17], which developed the TCP extended statistics MIB [16] to expose TCP protocol events that are normally hidden from the application. A MIB is a formal specification of a set of management variables that can be accessed by SNMP or other lower overhead mechanisms. Experimental prototypes of the MIB have been implemented in a number of operating systems, including Linux [17] and Microsoft Windows Vista [23].

Diagnostic efforts are further complicated by another property of TCP: the symptoms of most flaws scale by the flow's round-trip time (RTT). Note that for window-based protocols, performance models generally have an RTT term in the denominator. For example, insufficient TCP buffer space in either the sender or receiver, or background (non-congested) packet loss all cause TCP to have a constant average window size and performance that is inversely proportional to the RTT.

This poorly understood property leads to faulty reasoning about diagnostic results. A simple throughput test on a short local section of a path with minor flaws is likely to yield good results. The same test run over a longer path containing the same local flaws is likely to yield poor results. The naïve conclusion

* This work was supported by the National Science Foundation, Grant ANI–0334061.

M. Claypool and S. Uhlig (Eds.): PAM 2008, LNCS 4979, pp. 152–161, 2008.

would be that the local section is flawless, and the problem must be present in the longer path section. This "symptom scaling" property of TCP leads to incorrect inductive reasoning about flaws, and significantly contributes to the difficulty of solving end-to-end Internet performance problems.

This paper describes a tool, *pathdiag*, that uses TCP performance modeling to extrapolate the impact of local host and network flaws on applications running over long paths. The tool analyzes a number of key metrics of the local host and path and uses TCP performance models to determine thresholds for these metrics based on the stated application performance goals. *Pathdiag* reliably detects flaws that have no user-noticeable symptoms over a short path. It reports the problems and suggests remedies.

1.1 Motivation

Network performance has increased by an order of magnitude roughly every four years over the last two decades. Networking experts are usually quick to demonstrate the full data rate on each new network technology [11]. However, typical users experience data rates much lower than those seen by experts, and the gap is widening.

Internet2 has measured the performance of TCP bulk flows over their backbone since the beginning of 2002 [12]. As of August 2007, the median performance across their 10 Gb/s network was only about 3.4 Mb/s. Historical data shows that this rate has taken six years to double.

A small number of flows get very good performance. About 0.1% are faster than 100 Mb/s, and of those about half are close to 1 Gb/s. Since the backbone carries a significant number of very high-rate, long-distance flows, we know that it has to be free from flaws that would otherwise affect these sensitive flows.

The design goal of *pathdiag* is to help non-expert users attain better performance by easily and accurately diagnosing common flaws. These flaws are generally near the edge of the network where debugging efforts are subject to faulty inductive reasoning due to symptom scaling.

2 The *Pathdiag* Tool

Suppose a user tries to get good performance from an application that relies on bulk TCP data transfers from a remote server, as shown in Figure 1. The user's application client C, needs data from the application server S across a long network path that includes both a short local section and a long-haul backbone. The local section is assumed to have an RTT that is no more than a few milliseconds. The long-haul backbone can be any length, transcontinental (100 ms RTT) or even global (300 ms RTT).

The user can test the local section of the path and the client configuration by visiting a *pathdiag* server, PS, with a java-enabled browser. Ideally, PS would be located near the connection between the local network and the backbone. The *pathdiag* server tests the local path and client configuration and generates a report in the form of a new web page, displayed by the user's browser.

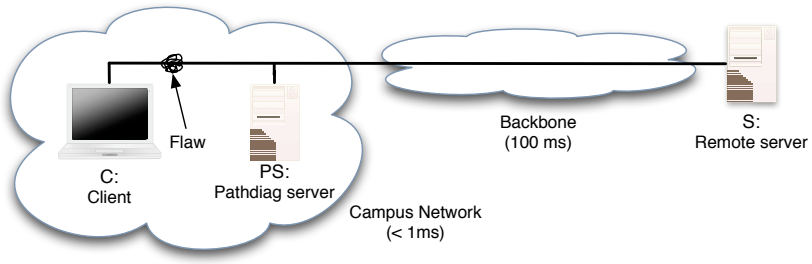


Fig. 1. Canonical *pathdiag* setup

Pathdiag estimates whether the local client and local path is sufficient to meet the target[1] data rate if the backbone were replaced by an ideal network with the same RTT. To do this, the user must provide two parameters: the target RTT from C to S and the target data rate for the application. If users do not know these parameters, the default values, 90 Mb/s over a 20 ms path, are appropriate for most university users. The report presents various metrics of the local client and local path, and indicates if they are within the thresholds of TCP performance models. It also suggests corrective action, if needed.

The components of the *pathdiag* server are shown in Figure 2. The browser loads the diagnostic client, which communicates with the server via a simple request-response control protocol. A TCP connection is established from the traffic receiver in the diagnostic client to the traffic generator. The measurement engine uses the Web100 prototype of the extended statistics MIB [16] to manipulate and instrument the TCP connection at the generator. An analysis engine evaluates the measurements and extrapolates the results to predict the impact of the local path on the user's application.[2]

2.1 The Measurement Engine

The measurement engine collects Web100 data in a series of sample intervals. For each interval, it adjusts the window size of the diagnostic TCP connection in discrete steps, and then captures the entire set of Web100 variables at the end of each sample. It computes several metrics during each test, the most important of which are $DataRate$, $LossRate$, RTT and $Power$ ($DataRate/RTT$). These are shown as functions of the window size for a typical link in Figure 3. These plots resemble those generated by "Windowed Ping" (*mping*) [14], a UDP-based tool that uses a similar measurement algorithm.

The measurement engine employs an adaptive scanner to select the window size for each sample interval. To minimize the total time required for the test,

[1] We use "target" when referring to components of the remote application and their parameters, such as end-to-end *target RTT*, desired *target data rate* and their product, the *target window size*.

[2] To support systems that cannot run a Java–enabled web browser, the "C" source for a portable command-line client is also published by the diagnostic server.

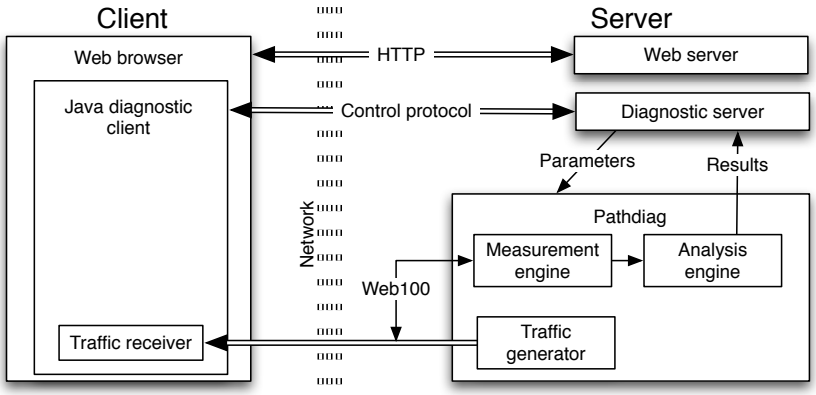


Fig. 2. Block diagram of the *pathdiag* client-server framework

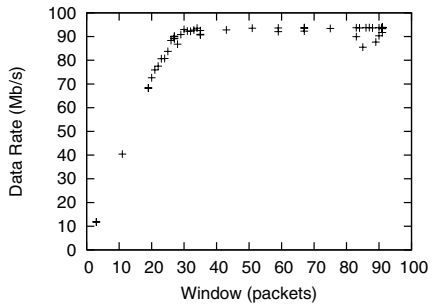
data is collected in multiple phases that emphasize specific properties of the network. A coarse scan across the entire window range is used to approximately locate two important window sizes: the onset of queuing and the maximum window size. Ranges around these values are then rescanned at progressively higher resolutions. In Figure 3, the fine scans can be seen clearly around window sizes of 30 and 80 packets, respectively. The maximum window sizes for scans are determined when TCP congestion control or an end-host limitation prevents the window from rising for three consecutive sample intervals.

Several network path metrics are calculated directly from the raw data as it is collected. $MaxDataRate$ and $MinRTT$ yield a measurement of the test path's bandwidth-delay product. $MaxPowerWindow$ is the window size with the maximum $Power$, indicating the onset of queuing. The $MaxWindow$ is the maximum amount of unacknowledged data that the network held. The difference between the $MaxWindow$ and $MaxPowerWindow$ is an estimate of the queue buffer space at the bottleneck.

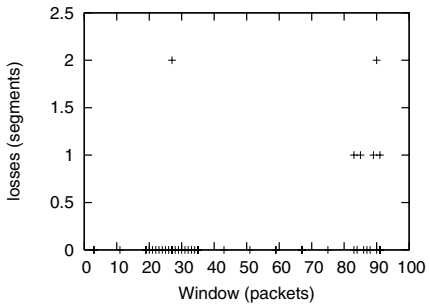
$BackgroundLossRate$ is calculated from the total packet losses from all sample intervals below the onset of queuing, as indicated by the $MaxPowerWindow$. It reflects bit errors and other losses that are not related to network congestion. If the adaptive scans do not provide sufficient loss data for the test described in the next section, additional loss data is collected at a fixed window size just below the onset of queuing. In general, the measurement engine collects enough data to observe the loss rate at the scale needed by AIMD congestion control to reach the target window size.

2.2 The Analysis Engine

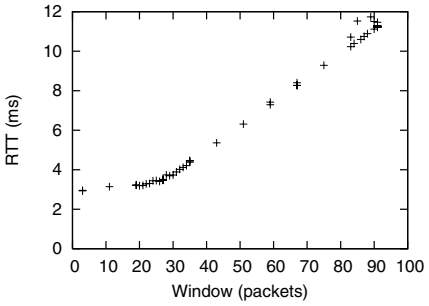
The analysis engine uses the two user-supplied parameters, end-to-end RTT and desired application data rate to evaluate the results from the measurement engine and produce a diagnostic report, as shown in Figure 4.



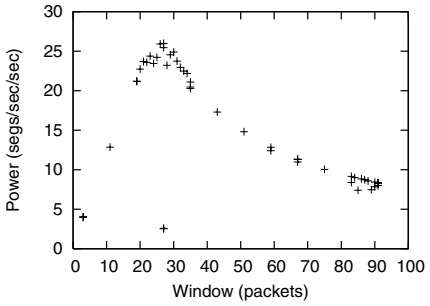
(a) Data rate vs. window size. Window sizes less than 30 were too small to fill the path, so the data rate was proportional to window size. Window sizes between 30 and 80 packets show data rates that were near the bottleneck rate, about 94 Mb/s.



(b) Loss rate vs. window size. Above 80 packets, the link started to exhibit persistent loss. Given the small RTT (about 2.5 ms), TCP can recover from these losses with only a slight reduction in throughput.



(c) RTT vs. window size. RTT was essentially constant at small window sizes. Above a window of 30, each additional packet in TCP's window was added to a standing queue at the bottleneck, and the RTT increased linearly with window size.



(d) Power vs. window size. Power reached a maximum at the point where the bottleneck crossed over from under-full (the link had idle time) to over-full (there was standing data in the queue), in some sense the ideal TCP operating point.

Fig. 3. Plots of scan results

The results in the generated report are grouped hierarchically. The base of the report shows test parameters and conditions. Test results are grouped into the following categories: local host (client) configuration, path measurements, and tester (server) consistency checks. Path measurements are further divided into data rate, loss rate, network buffering, duplex mismatch tests, and suggestions for alternate test parameters.

Test results are labeled and color-coded for easy reading. All failing tests (red) include a "corrective action" (starting with ">") indicating what needs to be fixed and how to fix it. In general, failing tests are guaranteed to be performance show-stoppers - the application will fail to meet the target data rate over the full end-to-end path as long as there are failing tests. The help for

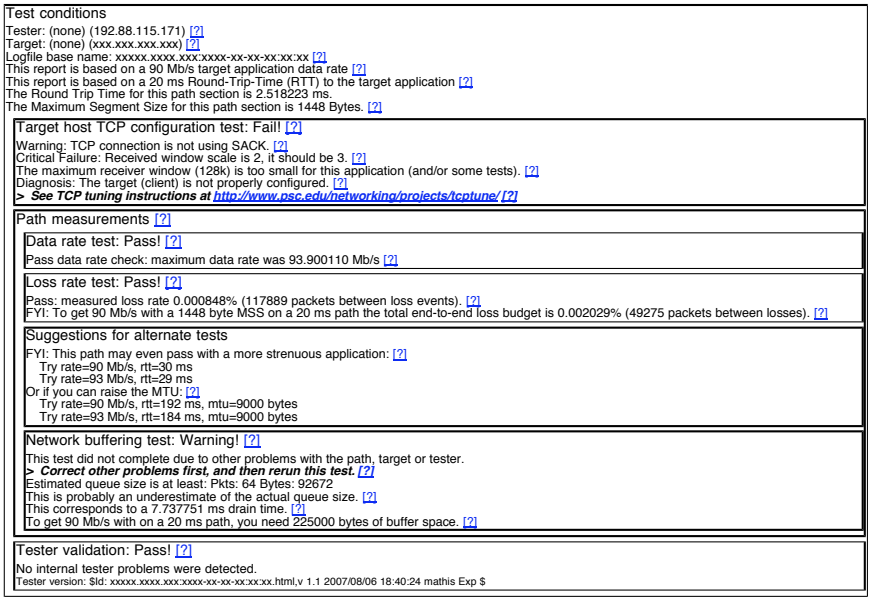


Fig. 4. Sample report from the same data as Figure 3

passing tests (green) indicates any caveats about limitations of the tests. Tests that are inconclusive for some reason yield orange warning messages. These include flaws that might not cause performance problems and tests that did not complete due to other failing tests. Messages in black are informational and are of most value to expert users. The analysis engine can detect 21 different failure conditions and 16 possible warnings.

Host Configuration. The host configuration tests confirm that TCP settings on the client are appropriate for target parameters. *Pathdiag* checks the options negotiated on the SYN and SYN-ACK. The Window Scale option [13] must have negotiated an appropriate value or it is flagged as a critical failure. It also checks if TCP Selective Acknowledgments (SACK) [18] or TCP Timestamps [13] are enabled.

A key test is whether the TCP receive buffer is larger than the target window. Since many modern operating systems adaptively size their receive buffers [9, 5, 23], it is necessary to check the announced receive window at extreme points of the measurements (peak data rate or window size). There are several corner cases that the analysis engine needs to consider. If the flow is limited by the receive window, and the maximum observed receive window for the entire run is less than the target window then the receiver never announced enough buffer space for the path, and the buffer is too small. If the receiver reduced its window at the extreme points, then receiver is not fast enough, which is a different problem. Also, since the section of the path under test is normally shorter than

the target path, it might be correct for an adaptive receive window to have a maximum size that is smaller than the target window. In this case, *pathdiag* cannot make a strong conclusion about the receive buffer size. However, in most default configurations, hosts that announce sufficient receiver window for the queue space test and pass the window scale check will also have sufficient receive buffer space.

Path Measurements. *Pathdiag* tests three parameters of the local path: maximum data rate, background loss rate, and bottleneck queue size. It also has a special-case test for Ethernet duplex mismatch [21], which is only invoked if the signature is detected.

Normally, the data rate test fails only if the tested path is not short enough, the user is mistaken about the properties of the path, or there is a serious problem such as a media-type negotiation failure. With a short RTT, TCP can overcome most flaws with only a minor performance reduction. As a consequence, most flaws do not mask other flaws when the path is short enough, so a single test run can detect multiple flaws.

Pathdiag measures the background, non-congested loss rate at a window that is slightly smaller than the window necessary to cause congestion on the link. A failure is reported if the measured loss rate is greater than the rate calculated by a TCP performance model [19] applied to the specified RTT and data rate for the target application.

A warning is issued if the measured bottleneck queue buffer space is less than the bandwidth-delay product of the target path. It is only a warning because *pathdiag* cannot determine if the small buffer will cause significantly reduced performance for the target path. The test reflects the traditional full-BDP rule for sizing router buffers for TCP [22]. Recent results show that smaller buffers are adequate for aggregated flows [1,8], but for single flows there are some situations that might cause full window-sized bursts. In particular, TCP slow-start naturally requires $cwnd/2$ buffer space at the bottleneck to avoid prematurely transitioning to congestion avoidance [6]. Furthermore, if the bottleneck employs active queue management (AQM) [2] such as RED [7], *pathdiag* is likely to measure the threshold for dropping packets rather than the actual buffer space used to absorb bursts. We are planning future work in this area.

Tester Consistency Checks. Occasionally, either the traffic generator or *pathdiag* itself might be a bottleneck. For example, there may be unanticipated users on the server. *Pathdiag* checks for this and other exceptional events, and reports them as problems with the tester.

2.3 The Server Framework

The reports generated by *pathdiag* are ordinary web pages. They can be bookmarked and the URL forwarded to experts for additional analysis. The on-line documentation stresses this feature [15]. Even relatively naïve users can generate diagnostic reports that clearly identify a problematic subsystem, and then forward them to people with the resources and authority to take corrective action.

Web archival of *pathdiag* reports is also critical to our ongoing improvement of the tool. We periodically scan various deployed diagnostic servers and retrieve the reports from the analysis engine and the raw data from the measurement engine. We inspect selected reports to confirm they agree with our manual analysis of the raw data. If we discover flaws that are not reported clearly, we make improvements to the analysis engine. We test the improved analysis engine by reapplying it to our collection of measurement data, and inspect the re-generated reports that differ from the original reports. In this manner every user contributes to our pool of test data, and to refinement of the tool. Our archive currently holds more than 7000 diagnostic reports.

The sever does not expose any private information about the user except the name and IP address of the client machine. No user information or system version information is explicitly exposed, though some operating systems may be deduced by their performance properties.

3 Strengths and Weaknesses

Symptom scaling makes traditional tools currently used for network diagnostics, such as *ttcp* and *iperf*, completely insensitive to flaws on short paths. Network experts use these tools over long paths to test for the existence of flaws, but actually locating the flaws is often a difficult trial and error process. As described above, *Pathdiag*'s defining characteristic is its ability to compensate for symptom scaling. As such, it works best when run on short path sections, and is most useful for debugging problems close to end systems.

Pathdiag fundamentally relies on active measurement[3] and must send a significant amount of bulk data to measure the loss rate at the scale of the target application. In this way it is different than many bandwidth-estimation tools that obtain results by measuring dispersion of short bursts of traffic without sending sufficient traffic to measure the loss rate at a scale relevant to AIMD congestion control.

The measurement algorithms used by *pathdiag* assume that other traffic across the tested path is relatively unvarying. Though it will not result in a false pass, highly variable levels of cross traffic may yield inconsistent results, especially in measurements of bottleneck buffer size and measured throughput. This can be largely mitigated by testing a shorter section of the path.

One fairly basic limitation is that the underlying diagnostic TCP stream is unidirectional, and TCP is intrinsically difficult to instrument from the receiving end. There are a number of potential solutions to this, which we hope to address in future work. Users *can* test the reverse path by running *pathdiag* at the other end of the test path. This can be done most easily by using the Internet2 Network Performance Toolkit [3] live boot CD to run a temporary server on almost any PC.

[3] For specialized uses, *pathdiag* can be run from the command line as a standalone tool without the web server server framework. One special use is to manually attach it to a bulk TCP stream belonging to another application.

Pathdiag cannot diagnose application problems, since the target application does not participate in the testing process. It is often very difficult to write applications that can attain high data rates even on ideal long networks. Some application problems are addressed in related work [10, 20].

4 Closing

Pathdiag is designed to improve TCP performance for the Research and Education masses—those with a need for high performance but without the time or expertise to individually diagnose network problems. It is particularly well suited for testing at the edges of the network, which is usually where the majority of performance-reducing flaws occur. Since it compensates for symptom scaling, *pathdiag* is able to isolate these near-edge flaws that are very difficult to diagnose using conventional local diagnostics.

In its most common form, deployment of *pathdiag* is fairly straightforward. A single well-connected test server is in a position to provide coverage for an entire campus or metropolitan network. It is our intent that *pathdiag* test servers will ultimately yield significant benefits to both the users and administrators of high-performance networks.

References

1. Appenzeller, G., Keslassy, I., McKeown, N.: Sizing router buffers. In: Proc. of ACM SIGCOMM 2004, October 2004, pp. 281–292 (2004)
2. Braden, B., et al.: Recommendations on queue management and congestion avoidance in the internet. In: RFC 2309 (April 1998)
3. Carlson, R.: Network performance toolkit, `http://e2epi.internet2.edu/network-performance-toolkit.html`
4. Carpenter, B., Brim, S.: Middleboxes: Taxonomy and issues. In: RFC 3234 (February 2002)
5. Fisk, M., Feng, W.: Dynamic right-sizing is TCP. In: 2nd Annual Los Alamos Computer Science Institute Symposium (LACSI 2001) (October 2001)
6. Floyd, S.: Limited slow-start for TCP with large congestion windows. In: RFC 3742 (March 2004)
7. Floyd, S., Jacobson, V.: Random early detection gateways for congestion avoidance. IEEE ACM Transactions on Networking 1(4), 397–413 (1993)
8. Ganjali, Y., McKeown, N.: Update on buffer sizing in internet routers. ACM CCR 36(4), 67–70 (2006)
9. Heffner, J.: High bandwidth TCP queuing, `http://www.psc.edu/~jheffner/papers/senior_thesis.pdf`
10. Heffner, J., Mathis, M.: Applications and the speed of light: How well do applications perform on long perfect networks (2007), Web paper: `http://www.psc.edu/networking/projects/applight/`
11. Internet2 Land Speed Record, `http://www.internet2.edu/lsr/`
12. Internet2 NetFlow Weekly Reports, `http://netflow.internet2.edu/weekly/`
13. Jacobson, V., Braden, B., Borman, D.: TCP extensions for high performance. In: RFC 1323 (May 1992)

14. Mathis, M.: Windowed ping: an IP layer performance diagnostic. Computer Networks and ISDN Systems 27(3), 449–459 (1994)
15. Mathis, M., et al.: NPAD diagnostics servers: Automatic diagnostic server for troubleshooting end-systems and last-mile network problems (2007), Web paper: http://www.psc.edu/networking/projects/pathdiag/
16. Mathis, M., Heffner, J., Raghunarayan, R.: TCP extended statistics MIB. In: RFC 4898 (May 2007)
17. Mathis, M., Heffner, J., Reddy, R.: Web100: Extended TCP instrumentation for research, education and diagnosis. Computer Communications Review 33(3), 69–79 (2003)
18. Mathis, M., Mahdavi, J., Floyd, S., Romanow, A.: TCP selective acknowledgement options. In: RFC 2018 (October 1996)
19. Mathis, M., Semke, J., Mahdavi, J.: The macroscopic behavior of the TCP congestion avoidance algorithm. Computer Communications Review 27(3), 67–82 (1997)
20. Rapier, C., Stevens, M.: High performance SSH/SCP - HPN-SSH (2007), http://www.psc.edu/networking/projects/hpn-ssh/
21. Shalunov, S., Carlson, R.: Detecting duplex mismatch on ethernet. In: Dovrolis, C. (ed.) PAM 2005. LNCS, vol. 3431, pp. 135–148. Springer, Heidelberg (2005)
22. Villamizar, C., Song, C.: High performance TCP in ANSNET. Computer Communications Review 24(5), 45–60 (1994)
23. New networking features in Windows Server 2008 and Windows Vista (2008), http://technet.microsoft.com/en-us/library/bb726965.aspx