Web100: Extended TCP Instrumentation for Research, Education and Diagnosis

Matt Mathis John Heffner Raghu Reddy
Pittsburgh Supercomputing Center
4400 Fifth Avenue
Pittsburgh, PA 15213
{mathis, jheffner, rreddy}@psc.edu

ABSTRACT

TCP has become the dominant protocol for all network data transport because it presents a simple uniform data delivery service which is sufficient for most applications over all types of lower network layers. By its very nature, TCP's adaption and retransmission strategies hide all of the details of the lower layers from the application. For example the only symptom of spurious packet loss (or nearly any other network problem) is longer elapsed time and lower performance.

This information hiding is fundamentally important to the growth of the Internet because it decouples the evolution of applications from the evolution of link layers. However it also hides valuable information from researchers, educators, network administrators, and other people who would benefit from insight into the inner workings of TCP and the lower layers.

In this paper, we present an architecture and infrastructure that provides for per-connection TCP instrumentation to expose otherwise hidden protocol events. We show examples how the infrastructure can be used in support of research, education and advanced network diagnostic tools.

Our work was motivated by the observation that since about 1985 network data rates for typical novice network users have fallen by about three orders of magnitude behind expert users (who have kept up with Moore's Law). We use the term "Wizard Gap" to describe this phenomenon. The Web100 and Net100 projects were formed as one step in closing the Wizard Gap.

General Terms

TCP Performance, instrumentation, Web100, Net100

1. INTRODUCTION

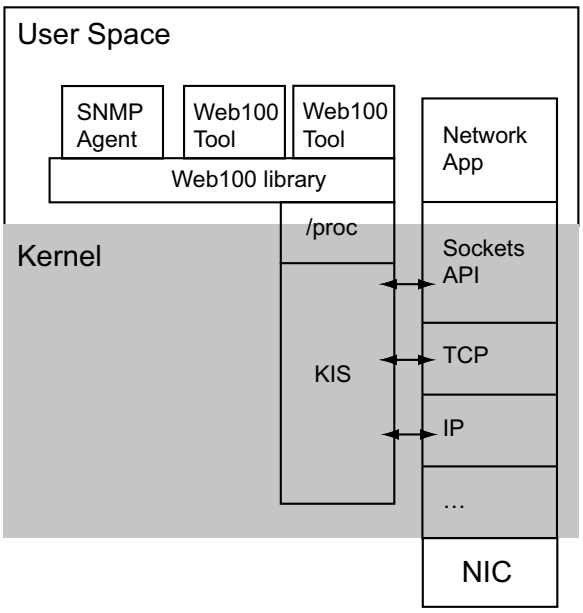
TCP is the dominant protocol for data transport. It carries the vast majority of all traffic over the Internet for a huge variety of applications including interactive sessions such as telnet or SSH, short flows such as Web traffic, and long-lived high bandwidth bulk data transfers. A significant reason for the success of the Internet is the TCP/IP "hourglass" design — TCP presents a simple uniform data delivery service suitable for the vast majority of applications while hiding all the details and complexity of lower layers.

It is fundamentally important to the growth of the Internet that TCP hides all of the details of the lower layers. It is for this reason that old applications always work over new networks, and that new applications work over old networks.

This information hiding is not without its cost: many important things happen in the lower layers that should be exposed but are not. For example, packet loss is hidden by TCP's retransmission machinery. If packet loss is due to a flawed network, the only symptom will be reduced performance. The inability to easily observe TCP's inner workings impairs our ability to conduct research in TCP behavior, test new TCP algorithms, educate future protocol researchers and detect bugs in TCP and the lower layers.

Many researchers expose network details by implementing ad-hoc TCP tools or specially instrumented alternate protocols. This can solve the information hiding problem, but it represents a significant duplication of effort to certify that the ad-hoc implementations are authentic.

The Web100 project was created specifically to develop an advanced management interface for TCP. We have instrumented TCP to collect per-connection statistics on all significant protocol events. There are instruments for capturing common events such as segments sent and received, as well as many more subtle instruments such as those characterizing the protocol events that cause TCP to reduce its transmission rate. The instruments are collectively referred to as the Kernel Instrument Set (KIS), which is the core of the Web100 project. The instruments are documented in a text database available online [43] and are summarized in Appendix A. Over the long run we expect this interface to become a standard, in the form of a TCP Extended Statistics Management Information Base (MIB) [24].



There is a separate KIS structure and instance of the protocol stack for each connection. On the right we see the network protocol stack, which exchanges information directly with the KIS, as kernel memory is shared. KIS data is moved between the kernel and user space through the /proc filesystem interface. Diagnostic and tuning applications use a library to access the filesystem data. It is possible to implement an SNMP agent on top of the library for remote access to the KIS.

Figure 1: The Web100 implementation structure

With the addition of some writable variables, the Web100 API and kernel data structures provide a convenient framework for controlling non-standard or experimental TCP algorithms on a per-connection basis. The Net100[31] project is using Web100 software in support of large high energy physics applications over the U.S. Department of Energy's ESnet.

Due to its standards-minded nature, the Web100 project must take a TCP purist and egalitarian point view — that is, features must be fair and acceptable in *all* parts of the Internet. The Net100 project takes the mission-oriented point of view — that high profile, data-intensive DOE applications are permitted to make use of unconventional and non-standard protocol features, as long as there is appropriate coordination with other people and projects sharing the same networks. This is a wonderful laboratory for field testing experimental algorithms, such as Sally Floyd's High-Speed TCP[12].

In an educational context, Web100 has been used to provide a framework for conducting experiments, providing students with hands-on experience. The real-time views of TCP variables, and the ability to modify key parameters helps students understand the various components that impact TCP flows.

Web100 is currently implemented as a Linux kernel patch plus user mode tools. All Web100 (with embedded Net100) code is available for download from `www.web100.org`.

This paper describes how TCP instruments can be used for Internet measurement (Section 3), testing experimental algorithms (Section 4) education (Section 5), and network diagnosis (Section 6).

2. IMPLEMENTATION OVERVIEW

The core of the Web100 project is the per-connection TCP instrument set. It is defined in standards language in an IETF MIB document [24], and implemented in what we call the Kernel Instrument Set (KIS).[1]

The architecture of our Linux implementation is depicted in Figure 1. Attached to each socket is a structure containing KIS variables and meta-data. Its fields are updated from key points in the protocol stack. An abstraction of this structure is exposed through a "proc" filesystem interface. To allow for future interfaces and other operating systems, we defined a portable API for accessing the instruments, implemented as a library. Since the instruments were designed to support a MIB, it is fairly natural to implement an SNMP agent on top of this library for exporting the instruments.[2]

It is a natural extension of this API to implement per-connection TCP controls. A "read" accesses an instrument, while a "write" accesses a control.

The library and kernel interfaces were designed with reasonable performance in mind. The filesystem exports binary data at fixed offsets rather than doing expensive formatting into human-readable ASCII data. On a 2 GHz Mobile Pentium 4, taking a snapshot of 140 variables takes about 30 μs, so polling on the order of hundreds of times per second puts little load on the CPU.

In addition to the library, the Web100 software suite also includes a number of application tools. There are some small command line-based tools which are particularly useful for scripting. Also, there are some graphical tools which aid in visualizing variables. One tool will display a running strip chart of a variable's value or its change per unit time. Another particularly useful part is the "triage" tool, which locates the performance bottleneck as the sender, receiver, or network. This tool is described in Section 6.1.

3. TCP BASED MEASUREMENT

TCP has an ideal vantage point to measure the network. It is situated at the neck of the Internet hourglass such that it processes the vast majority of all data moving through the Internet. It typically has direct interrupt level access to the IP layer and network devices. With proper instrumentation every TCP Acknowledgment is an opportunity to collect information about the network, the far end-system and even the application. Web100 strives to capture this information.

It is conventional to classify Internet measurement tools as

[1]The MIB and KIS are not currently in sync, but this is ultimately a goal for the KIS to implement a superset of the MIB.

[2]An SNMP module for the TCP Extended Statistics MIB on top of the Web100 library has been partially implemented by Glen Turner of AARNET [41].

active or passive, depending on whether or not they inject traffic into the network. [4] Strictly speaking web100 is passive, because it does not by itself inject any additional traffic into the network. In the usual case, the measurements are a side benefit of ordinary traffic passing through the network. However, Web100 has the flavor of active measurements because TCP is sending the packets and processing the Acknowledgments that it uses for measurement. Web100 is best characterized as passive instrumentation of active transport.

We believe that this approach has a number of profound advantages over other measurement techniques, which we will describe in the rest of this section. Some of these techniques are already work in progress, others are significant research opportunities that have yet to be explored.

3.1 Augmented Iperf

`Iperf` is a popular Internet performance tester that is frequently used to measure available Internet throughput [32]. Although `iperf` is the tool of choice for one–time tests, many people consider it to be far too expensive for on going repeated tests, because it normally requires several tens of seconds of full rate data transfers to get accurate measurements. If `iperf` is run a few times per day, 365 days per year, it will in total consume substantial network resources.

`Iperf` requires such long tests because it computes the average performance across the entire transfer, which includes the very time consuming TCP Slow Start at the beginning of the connection. Since Slow Start transfers relatively little data in a relative long time, it greatly depresses the observed average data rate. Ajay Tirumala has increased the accuracy of very short `iperf` measurements by using Web100 to detect the end of Slow Start, so that it can be excluded from the average data rate calculation. [40] He was able to obtain 1 second data rate measurements that use 94% less network traffic but are still within 10% of longer `iperf` measurements.

An alternative approach might use a `tcptrace` [33] style analysis tool to automatically inspect a packet trace and trim off the Slow Start. This approach would incur far more overhead, since every packet has to be processed by both `tcptrace` and TCP itself. Web100 has the advantage of providing the necessary instruments such that the end of Slow Start can be detected with sufficient precision with low rate polling.

3.2 Experimental link layers

We are aware of at least one equipment vendor who is using Web100 to investigate how their proprietary link layer technology interacts with standard TCP. Although they have not chosen to publish their particular application, they did offer specific comments on improvements to a couple of instruments, which we have included into Web100.

Without TCP instrumentation, link layer designers often resort to simple benchmarking to tune their products. This approach is overly sensitive to a large number of poorly controlled parameters, which makes it very difficult to effectively optimize the link layer. With detailed TCP instrumentation the engineer can tell exactly why the TCP performance changed, and use this to guide their design.

3.3 NTAF

The Network Tool Analysis Framework (NTAF) [21] was developed at Lawrence Berkeley National Laboratory (LBNL) for automatically monitoring changes in the"health" of multiple Internet paths. It periodically runs active measurement tools, including `iperf` and collects Web100 statistics. It converts them to to NetLogger events and archives them [39].

These statistics can be post-processed to support trend analysis. Lee et. al. [22] describe how they have used archived NTAF results to build a database of Grid network statistics. A significant motivation behind this effort is that it provides a way of correlating changes in infrastructure to changes in performance.

A good record of past network conditions can also be used to aid in pre-loading of certain network properties. For example, Linux has provisions to preset RTT, RTT Variance, Ssthresh, and other parameters in the route cache, which are be preloaded into TCP when a connection starts. Setting good values for these parameters can result in higher performance. However, incorrect values may significantly hurt performance. Careful analysis of logged data can aid in properly assigning these preloaded values. The Net100 Work Around Deamon, described in section 4.2 provides a more general framework for this sort of advanced tuning information.

3.4 Internet tomography

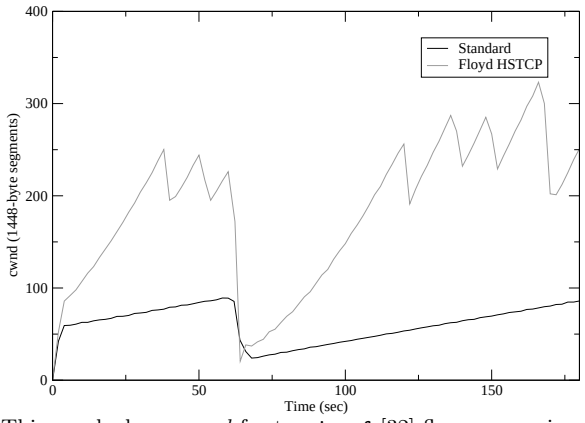
If a large content provider were to deploy Web100 with NetLogger or some other database in their server farms they could collect Internet performance statistics (e.g. packet loss rates and round trip times) to all of their customers. This performance data could be aligned with routing information to do large scale Internet tomography.

There is significant potential, too, for more wide-spread deployment. If, for example, a large number of campus network administrators used instrumented TCP to collect statistics across their campus or community to apply the techniques prototyped in the NTAF, the aggregation of the datasets across multiple campuses could be mined for performance and topological data about the global Internet.

Web100 makes this sort of measurement much more feasible because it is passive (does not necessarily require additional measurement traffic) and does not incur the huge processing overhead of inspecting packet traces.

4. TCP EXTENSIONS

Under the Net100 project we developed writable per connection controls for several TCP extensions. These controls greatly simplify the testing of experimental TCP algorithms because they enable researcher to configure and test risky new TCP features under controlled circumstances without interfering with normal system or network operation. Generally our TCP extensions are implemented such that they are off by default so that critical flows are not compromised by untested TCP algorithms. Experimental TCP algorithms can be enabled and configured on a per connection basis as needed by the researcher.



This graph shows *cwnd* for two `iperf` [32] flows, one using standard congestion avoidance, one using the HighSpeed TCP congestion avoidance for large windows. At about 60 seconds into each flow, there is a line-rate burst of UDP packets causing significant loss to the TCP flow (and consequently a large reduction in *cwnd*). The queue in the bottleneck router happens to be too small, so each flow exits slow start too early. The HighSpeed flow reaches full bandwidth several times faster than the standard flow would be able to (it never does). After the loss event, the HighSpeed flow recovers much faster.

Figure 2: Standard vs. HighSpeed AIMD

The combined resources of detailed TCP instrumentation and per connection controls help all phases of testing TCP extensions. New untested algorithms can easily be debugged in environments where the researcher has good visibility into the protocol behavior and control over its operations. More mature extensions can be easily compared with other algorithms or parameter choices in concurrent or sequential side-by-side tests on the same end systems. In the final stages of development, algorithms can be widely deployed to collect real operational field experience in such a way that researchers can manage the risks associated with experimental protocols in production environments.

In the following sections we illustrate the per connection TCP controls with several examples of experimental algorithms.

4.1 Testing possible standard algorithms

Sally Floyd's High Speed TCP for large windows[12] and Tom Kelly's Scalable TCP[20] have been reimplemented in the Web100 kernel by Tom Dunigan and Florence Flower at ORNL under the Net100 project. These algorithms can be enabled and configured per flow with Web100 control variables, permitting comparison tests [7, 8] between the various algorithms and standard congestion control.

The ORNL team is currently testing a reimplementation of Vegas Congestion Management [3]. We would like to include additional congestion control algorithms, such as FAST[19] and in Web100.

4.2 Workarounds

Some experimental TCP algorithms are workarounds for specific problems in the network environment, such as insufficient buffering in routers or switches.

It is well documented that TCP needs a full bandwidth-delay product of buffering at every potential bottleneck in the network to reach full data rate[42]. If the network does not have sufficient buffering, one of the reasons that TCP performance suffers is because TCP's Slow-Start will overflow the short queue, and the connection will prematurely enter Congestion Avoidance at a small window. Floyd's Limited Slow-Start[13] implements a workaround for this problem by clamping Slow-Start as the window gets large. The algorithm has a parameter, *max_ssthresh*, which limits the amount of queue space which will be used by the connection during Slow Start by slowing the increase of *cwnd* when it is larger than this value. If *max_ssthresh* is set smaller than the shortest queue in the network, the short queue is less likely to cause premature losses. TCP has a chance to reach a large enough window to fill the network pipe before experiencing its first loss.

The down side to Limited Slow-Start is that TCP Slow-Start reverts from exponential to steep linear window growth above *max_ssthresh*. While this may help on a particular under-buffered link, it greatly slows TCP on links with proper buffering and doesn't help at all for links with even smaller queues. Properly setting *max_ssthresh* requires advance out-of-band information about the network path.

This is one example of a class of algorithms that we refer to a workarounds. They use some outside information, such as bottleneck queue size, to increase TCP performance by limiting TCP in some way to prevent it from overwhelming the network. Workarounds can not be standardized because they generally do more harm than good if applied to arbitrary paths without advance information.

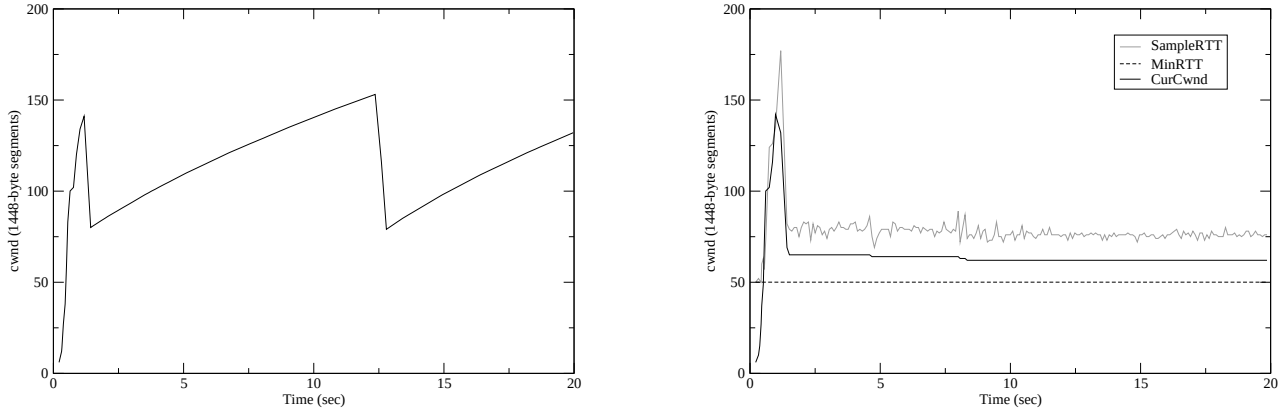
The Work Around Daemon (WAD) component of the "Network Aware Operating System" under the Net100 project[10] implements a framework to systematically collect and manage configuration data for workarounds. It uses policy and prior measurements as collected by the NTAF (See section 3.3, above) to set per-connection controls such as *max_ssthresh*. In this manner it can help to systematically deploy workarounds.

Note that the WAD mechanism cannot be globally deployed because the configuration data does not scale. It would be better to eliminate the underlying problems than to widely deploy workarounds. We hope that in the long run Web100 will have this effect: better observability will expose the underlying network problems and the WAD can be used to demonstrate the potential of the network by masking the problems for specific host pairs. This combination might motivate network administrators to correct the underlying problems for all users.

4.3 User mode congestion control

Before we look at examples of using Web100 to control more aggressive TCP algorithms, we need to consider issues of fairness and appropriate experimental etiquette.

Some networks such as Department of Energy's ESnet and many corporate intranets are mission oriented, meaning the



(a) Reno (b) Vegas

In the "Reno" mode, we see after about a one second slow start the traditional "saw tooth" AIMD congestion avoidance pattern in *cwnd*. The "Vegas" mode plot shows *cwnd* being controlled as a function of the perceived bottleneck queue as measured by the difference in sampled and minimum RTT (also plotted). These measurements were taken using a 10 Mbps bottleneck and an approximately 50 ms delay.

Figure 3: User mode congestion control behavior

network is designed specifically to support the primary mission of its host organization, and is properly sized to do so. In these networks the notion of fairness is replaced by the notion that mission-critical data is more important to the organization than all other data. In these networks it is possible to conduct experiments to measure the interaction between more aggressive algorithms and legacy algorithms in an authentic setting[14, 35], without addressing all of the political issues associated with fairness. As long as the experimental algorithms are used in coordination with mission-critical applications and the Network Operation Center, they are sheltered from controversy. This situation is especially valuable for conducting experiments to measure the actual fairness of algorithms that are alleged to be unfair.

In this vein, Web100 includes some controls to support user mode congestion control, where the majority of the congestion control algorithm is implemented in a user-space daemon. This facilitates faster prototyping because it is vastly easier to debug code in user-space than in the kernel. To do this we implemented two TCP control variables:

NoAI A value of **true** disables the "additive increase" part of the TCP's AIMD congestion control algorithm. When set, *cwnd* remains constant unless reduced by some congestion indication, or adjusted by:

CwndAdjust Cwnd will be adjusted up or down by the signed value written to CwndAdjust, and CwndAdjust is cleared.

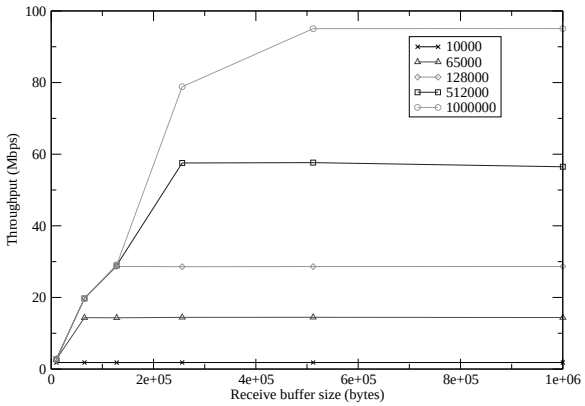
To implement congestion control in user space, set the *NoAI* variable to true. The kernel will still multiplicatively decrease *cwnd* when congestion is detected. The daemon adjusts *cwnd* in steps by writing to *CwndAdjust*, for example, by writing +1 exactly once per round trip time.

To validate these variables, we wrote a simple user mode congestion control daemon which approximately implements standard Reno style Congestion Avoidance [17] and Vegas [3]. The following pseudo-code suggests the implementation, although the actual implementation took advantage of timers, etc, for efficiency.[3]

```
while (State == ESTABLISHED) {
  switch (cc_mode) {
  Reno:
    snd_max_sav = SndMax;
    cwnd_sav = CurCwnd;
    while (SndMax < snd_max_sav+cwnd_sav)
      ;
    CwndAdjust = 1;
    break;
  Vegas:
    snd_max_sav = SndMax;
    cwnd_sav = CurCwnd;
    while (SndMax < snd_max_sav+cwnd_sav)
      ;
    queue = cwnd_sav-(cwnd_sav*MinRTT/SampleRTT);
    if (queue < ALPHA)
      CwndAdjust = 1;
    else if (queue > BETA)
      CwndAdjust = -1*max((queue-BETA)/8, 1);
    break;
  }
}
```

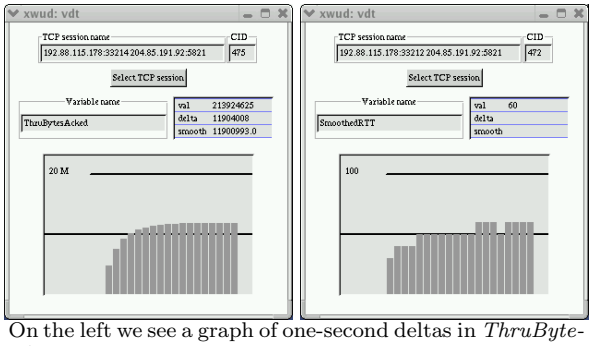
With the Web100 infrastructure in place, the kernel part of this implementation is simple. It only requires 8 lines of

[3]On our networks, Vegas also requires higher resolution timers than normally provided by Linux 2.4. The system clock was reconfigured to provide the necessary resolution.



The plot contains one line for each send buffer size; X axis represents the various receive buffer sizes, and Y axis is the achieved throughput.

Figure 4: Student Data



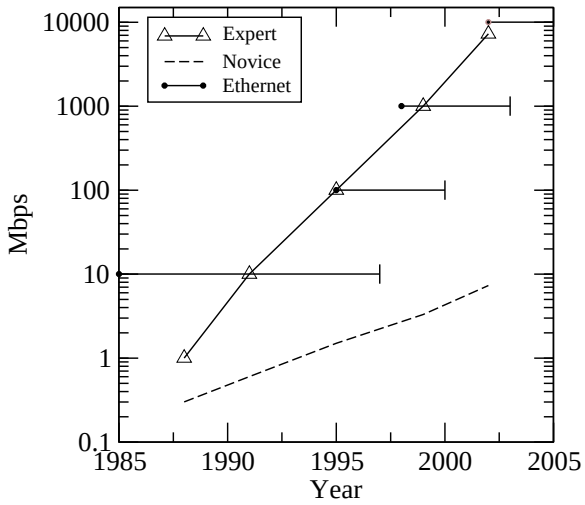
On the left we see a graph of one-second deltas in *ThruBytesAcked*, equivalent to the throughput of the connection. We see this ramp up as the connection goes through Slow-Start and tapers off as it reaches its 100 Mbps interface line rate (the local interface is this path's bottleneck). On the right, we see a plot of *SmoothedRTT* which increases from about 35 ms to 65 ms, indicating that there is a queue building up at the bottleneck. This is a good path for this flow.

Figure 5: Web100 strip chart tool

clear, easy to validate code. All of the interesting complexity is in the daemon, which is far easier to debug and adapt as the experimental algorithm evolves. Furthermore, changes can be tested without rebooting.

With Web100 it is very easy for any researcher to add additional simple controls (and even instruments) to TCP for rapid prototyping of experimental algorithms in user space. Having a framework that easily supports specialized research instruments and additional per connection controls greatly facilitates rapid prototyping.

This is not without a downside: exposing this degree of control can also facilitate abuse. It is not terribly difficult to imagine aggressively tuned peer-to-peer applications causing congestive collapse of a campus network. However, this risk is not new or uniquely caused by Web100 software. And, unlike Denial-of-Services tools already used today, a Web100 TCP will not lie about its endpoint IP addresses, and is still constrained by the requirements of reliable delivery.



The horizontal bars indicate the lifecycle of Ethernet at its available speeds from introduction to maturity. The triangles are actual data points of Internet performance milestones. The dashed line reflects the general trend in average throughput based on workstations' default buffer sizes, which rose from 4 KB in 1985 to 64 KB today, and North American continental round trip times, which have dropped about 50% due to higher clock speeds and shorter packet copy times.

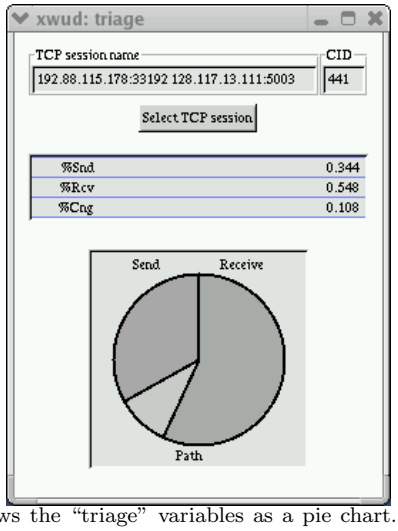
Figure 6: The Wizard Gap

5. EDUCATION

Web100 has also proved useful in a teaching environment. Larry Dunn uses it in a networking class at the University of Minnesota to help students explore concepts relevant to TCP, such as the bandwidth-delay product and host network buffers.

In this class, students first gain a working knowledge of the Web100 graphical tools in a LAN environment. Then, they use these tools to systematically vary both send and receive buffer space for a TCP flow running over a wide area path, and examine the impact on performance. Typical results are shown in Figure 4. From this experiment, they learn that either a send or receive buffer smaller than the bandwidth-delay product will proportionally limit throughput. They also see the impact of various default buffer sizes – a 64 KB buffer size limits even the modest path used by the class to 15 Mbps versus the achievable 90+ Mbps. Students can also empirically determine the bandwidth-delay product of the path used for the experiment.

The moving bar graph diplayed by the Web100 tools is helpful in connecting network events and TCP behavior. Starting with adequately sized buffers, displaying the throughput per second shows a gentle TCP sawtooth. Looking simultaneously at "CongestionSignals" reported at the sender shows a direct correspondence between congestion signals and rate reduction (the downward edge of the sawtooth). This serves to nicely illustrate the TCP congestion controller's behavior. It is also possible to see the correlation between round-trip time and and window size as the bottleneck queue grows, as shown in Figure 5.



This shows the "triage" variables as a pie chart. The receiver on this flow sleeps between reads, so the receive window is the majority limit for this flow.

Figure 7: Triage tool.

Students have uniformly reported that this type of hands-on experience with Web100 has deepened their understanding of bandwidth-delay product issues specifically, and TCP behavior generally.

6. DIAGNOSTICS

The Web100 project was initially motivated by the recognition that ordinary network users in such diverse fields as high energy physics, computational biology, astronomy and meteorology [2] were very unhappy with the performance of their network based applications. Their dissatisfaction was exacerbated by the highly publicized first network Land Speed Record [44]. We coined the term "Wizard Gap" [23] to describe the situation where networking experts could get orders of magnitude better performance than ordinary users, and the difference is increasing exponentially. See Figure 6.

Under the Web100 project we focused on two aspects of closing the Wizard Gap: first greatly increase the network and system administrators ability to inspect the network and protocol stack to help diagnose hidden bugs, and second, automate end-system tuning to reduce the level of expertise needed by true end users. This latter aspect is an extension of prior autotuning work[38, 11, 16] and will be covered in a future paper.

Although network diagnostics are central to the project as a whole, in this section we will describe only a couple of the most important examples.

6.1 The triage instruments

The triage instruments are most useful for the very first investigation of a performance problem. These instruments provide a direct indication of the extent to which a bulk data transfer is limited by the sending host, receiving host or the path in between, by tracking the events that cause TCP to stop sending data. The Web100 user tool kit includes a GUI tool (shown in Figure 7) for displaying the relative amounts of time that TCP spends waiting in each of these three states:

Receiver: TCP stopped sending data because it ran out of receiver window. This indicates that the receiver is applying flow control back pressure to the sender (because the application is not keeping up with the data) or that the receiver is mis-tuned (insufficient buffer space).

Path: TCP stopped sending data because it ran out of congestion window, *cwnd*, or experienced a retransmission timeout. This generally indicates insufficient network capacity or a problem with the network path.

Sender: TCP stopped sending data even though it has sufficient congestion and receiver window. This indicates that the Sender ran out of data (application limited) or some other sender algorithm such as the Nagle[30] or SWS [6] stopped the sender.

In most situations the Triage Tool identifies one subsystem as the primary bottleneck, eliminating effort that might have been wasted debugging the other non-bottlenecked subsystems.

The triage GUI tool is built on top of a Web100 library that only assumes simple semantics from the underlying interface to the kernel instruments. This library was specifically designed to protect Web100 diagnostic tools from changes in the details of the kernel implementation. In most cases we were successful: substantial kernel changes usually did not even require recompiling the tools. Furthermore, most of the GUI tools are implemented as"GTK widgets", so that other tool developers can reuse existing code rather than reimplement their own real time graphical displays for Web100 variables.

6.2 Path diagnostics

We are often asked to help debug long paths that fail to support high data rate applications, even though each section of the path seems to have sufficient unused capacity and can individually support a high data rate diagnostic test. Mathis and Reddy have developed a Web100 based high intensity network diagnostic [27] that addresses this situation.

The `pathdiag` tool is based on the following simplified macroscopic congestion model[28]:

$$Data\_Rate \approx \frac{MSS}{RTT}\frac{0.7}{\sqrt{p}} \quad (1)$$

For a specific target data rate on a particular path, the value of each variable except p, the loss rate, is known *a priori*. The minimum round trip time (RTT) is determined by the length of the network path, and the Maximum Segment Size (MSS) is determined by switches and routers along the path. The objective of the test is to determine if any section of the path contributes more packet loss than the budgeted maximum p as computed from equation 1. If any section

has too much packet loss it will prevent the entire end-to-end path from sustaining the target data rate.

Web100 enables us to directly monitor all the variables in equation 1, as we test each section of the path.

6.3 Diagnostic Servers

Web100 makes it convenient to implement diagnostic web servers that accept connections from any client and perform diagnostic tests on the connection back to the client. This approach yields a "one click" diagnostic that can provide novice and expert users with very good information about the client configuration and the network path. The big advantage of diagnostic web servers is that a small number of well connected servers can provide this service for a huge number of clients, without any special support on the client.

For example, the SYNTEST server [37], which can be invoked at `http://syntest.psc.edu:7961/` or by `telnet syntest.psc.edu 7960`, confirms correct SYN option negotiation as required for high performance TCP operation over long paths. The checked options include Window Scaling, Timestamps [18] and SACK[25]. This server is useful for confirming the correct manual host tuning [26] and detecting middleboxes that inappropriately strip some options.

Richard Carlson developed a particularly useful web site `http://miranda.ctd.anl.gov:7123/` that uses a java applet in the browser and Web100 in the server to measure and diagnose Internet paths. [5] This diagnostic server is based in large part on the ORNL "bandwidth tester" [9] developed by Tom Dunigan. Carlson has developed some additional heuristics for diagnosing common Internet mis-configurations, including detecting the number one fast Ethernet deployment problem: duplex mismatches between nodes. Since this server is well connected to ESnet, Abilene (Internet2) and StarTAP/StarLight it can provide one click diagnosis of many problems at most well connected universities.

7. ANTI-CONCLUSION

No part of the Web100 project is competed. We claim that extended per-connection TCP instrumentation and controls will prove to be extremely valuable for research as well as debugging the operational Internet and application base.

The extended instrumentation has opened a great new vista into TCP and network dynamics; researchers have already used Web100 to observe previously unexplored TCP phenomena. We are actively seeking additional users and researchers to experiment with Web100.

The KIS data structures can easily be used as a framework for controlling and testing experimental TCP algorithms in ways beyond the initial scope of the Web100 project. This has already proven to be a valuable lubricant for TCP congestion research.

The WAD controls provide a flexible and general mechanism to adjust TCP in non-general and non-standard ways to overcome specific problems in the network, such as insufficient buffering. It is likely that the WAD controls will foster discussion about the relative costs of fixing the network vs. standardizing these algorithms.

Intranets and agency networks provide an excellent opportunity for experimenting with alternate TCP algorithms if they already have polices in place that give high profile mission-oriented activities precedence over other traffic. As long as the experimentation is coordinated with the network operation center and mission-critical activities, then any potential repercussion of the experimentation can be resolved. In these environments we have the opportunity to actually field test how new and more aggressive congestion control algorithms interact with todays standard TCP.

The Web100 team is addressing key issues in support of bringing extended TCP statistics into wide adoption in the the commercial operating system market. This is a multi-pronged effort, including a new task to re-implement the KIS and API in FreeBSD, such that we can release source under both GPL and BSD licenses. We are also submitting patches into Linux 2.6.

We are also actively working on a TCP Extended Statistics MIB[24] to make a large portion of the KIS available through standard network management protocols. One of our collaborators already has a SNMP agent partially implemented. We see advancing the MIB in the IETF standards track to be an absolutely critical precursor to widespread commercial adoptions.

It is our intent that the Web100 project will progressively close the Wizard Gap by exposing bugs that are otherwise hidden from all but the best experts. It greatly reduces the effort required by programmers and system and network administrators to fix pervasive minor bugs. Diagnostics will uncover problems in local networks that impede higher performance on the wide area networks.

Diagnostic servers make it very easy to detect some common types of path and system configuration bugs. By making it far easier to discover and correct the "easy" problems, these serve to free more precious networking expertise to work on the more difficult problems.

The triage tool make it easier to provide the first level diagnosis and more effectively request expert help.

Improved automatic controls in the operating systems will completely free novice users from having to learn about arcane TCP adjustments such as socket buffer tuning.

Automatic controls that make sense in all parts of the Internet can be standardized and built into operating systems. Other controls may come to be widely implemented through Work Around Daemons that can combine out-of-band measurement with policy controls to enable situation-specific algorithms to help performance. Although not general, these techniques are extremely effective in some situations, and can be configured by experts for automatic use by novices.

We hope that the improved diagnosis will cause novice users to appreciate new scales for "large files" and "fast network."

The Wizard Gap implies that current notions of "large" and "fast" have fallen almost 3 orders of magnitude behind Moore's Law when applied to network applications. Closing the Wizard Gap has the potential to increase typical application performance by orders of magnitude. This has the potential to raise presented network traffic everywhere.

See `www.web100.org` for more information.

8. ACKNOWLEDGMENTS

This paper is the product of a cast of hundreds. It would not have been possible without the effort of so many people in so many different organizations. Please forgive us if we fail to mention you and your contribution to the project.

Thanks to the whole Web100 team: John Estabrook and Steven Engelhardt who wrote the Web100 graphical tools; Tanya Brethour who supports all of our collaborators and keeps the web pages up-to-date; Peter O'Neil who pursues vendors and other outside contacts; Wendy Huntoon, Janet Brown, Marla Meehl and Jim Ferguson who provide adult supervision; and a very special thanks to George Brett and Basil Irwin, whose initial vision and persistence were crucial to getting the project launched. Everyone on the Web100 team could have been listed as authors.

Thanks also to the whole Net100 team: Tom Dunigan, Florence Fowler and Nagi Rao at ORNL; Brian Tierney, Martin Stoufer and Jason R. Lee at LBNL. Their experience and experiments have brought much much wisdom and clarity to the Web100 project.

Thanks to Larry Dunn and University of Minnesota (and Cisco) for contributing the material in the Education section.

Thanks to all the Web100 early adopters and other collaborators who have provided so much feedback and good advice.

Web100 is funded by the National Science Foundation under Grant No. 0083285 and a Cisco URP grant.

Net100 is funded through the Mathematical, Information and Computational Sciences Division in the Office of Science at the U.S. Department of Energy.

9. REFERENCES

[1] M. Allman, V. Paxson, and W. Stevens. Tcp congestion control, RFC2581, April 1999.

[2] ANINEAR. Advanced networking infrastructure needs in the atmospheric and related sciences (aninars) workshop report. http://www.scd.ucar.edu/nets/projects/completed/1999.complete.projects/nlanr/final.report.htm.

[3] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP vegas: New techniques for congestion detection and avoidance. In *ACM SIGCOMM*, pages 24–35, 1994.

[4] CAIDA. Internet tools taxonomy, 2003. http://www.caida.org/tools/taxonomy/.

[5] R. Carlson. Developing the Web100 based network diagnostic tool (NDT). *PAM*, April 2003.

[6] D. D. Clark. Window and acknowledgement strategy in TCP, RFC813, July 1982.

[7] T. Dunigan. Floyd's TCP slow-start and AIMD mods, 2003. http://www.csm.ornl.gov/~dunigan/netperf/floyd.html.

[8] T. Dunigan. Kelly's scalable TCP AIMD mods, 2003. http://www.csm.ornl.gov/~dunigan/netperf/kelly.html.

[9] T. Dunigan. ORNL TCP Web100 bandwidth tester, 2003. http://firebird.ccs.ornl.gov:7123/.

[10] T. Dunigan, M. Mathis, and B. Tierney. A TCP Tuning Daemon. *Supercomputing 2002*, November 2002.

[11] W. Feng, M. Fisk, M. Gardner, and E. Weigle. Dynamic Right-Sizing: An Automated, Lightweight, and Scalable Technique for Enhancing Grid Performance. *7th PfHSN*, page 16, April 2002.

[12] S. Floyd. HighSpeed TCP for Large Congestion Windows. *Work-in-Progress: IETF Internet-Draft*, August 2003. http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-highspeed-01.txt.

[13] S. Floyd. Limited Slow-Start for TCP with Large Congestion Windows. *Work in progress: IETF Internet-Draft*, July 2003. http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-slowstart-00.txt.

[14] S. Floyd and V. Paxson. Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking*, 9(4):392–403, August 2001.

[15] M. Handley, J. Padhye, and S. Floyd. TCP congestion window validation, RFC2861, June 2000.

[16] J. Heffner. High bandwidth TCP queuing, July 2002. http://www.psc.edu/~jheffner/papers/senior_thesis.ps.

[17] V. Jacobson. Modified TCP congestion avoidance algorithm. Message to end2end-interest list, April 1990. ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt.

[18] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance, RFC1323, May 1992.

[19] C. Jin et al. FAST kernel: Background theory and experimental results. In *First International Workshop on Protocols for Fast Long-Distance Networks*, February 2003.

[20] T. Kelly. Scalable TCP: Improving performance in highspeed wide area networks. In *First International Workshop on Protocols for Fast Long-Distance Networks*, February 2003.

[21] LBNL. Network tools analysis framework (ntaf), 2003. http://www-didc.lbl.gov/NTAF/.

[22] J. Lee, M. Stoufer, and B. Tierney. Monitoring data archives for Grid environments. *Supercomputing 2002*, November 2002.

[23] M. Mathis. Pushing up performance for everyone, December 1999. Presentation to Joint Techs workshop (first use of wizard gap).

[24] M. Mathis, J. Heffner, R. Reddy, and J. Saperia. TCP Extended Statistics MIB. *Work in progress: IETF Internet-Draft*, November 2002. Status page: http://www.web100.org/mib.

[25] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options, RFC2018, October 1996.

[26] M. Mathis and R. Reddy. Enabling High Performance Data Transfers, 2002. http://www.psc.edu/networking/perf_tune.html.

[27] M. Mathis and R. Reddy. Pathprobe: Network Path Diagnostic Tools, 2002. http://www.psc.edu/~web100/pathprobe/.

[28] M. Mathis, J. Semke, and J. Mahdavi. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communications Review*, 27(3), 1997.

[29] J. Mogul and S. Deering. Path MTU discovery, RFC1191, November 1990.

[30] J. Nagle. Congestion control in IP/TCP internetworks, RFC896, January 1984.

[31] Net100. Home page, 2003. http://www.net100.org/.

[32] NLANR. Iperf—the TCP/UDP bandwidth measurement tool, 2002. http://dast.nlanr.net/Projects/Iperf/.

[33] S. Ostermann. TCPtrace, 2003. http://www.tcptrace.org/.

[34] V. Paxson and M. Allman. Computing TCP's retransmission timer, RFC2988, November 2000.

[35] V. Paxson and S. Floyd. Why we don't know how to simulate the internet. In *Winter Simulation Conference*, pages 1037–1044, 1997.

[36] K. Ramakrishnan, S. Floyd, and D. Black. A proposal to add explicit congestion notification (ECN) to IP, RFC3168, September 2001.

[37] R. Reddy. SYN option check server, 2003. http://syntest.psc.edu:7961/.

[38] J. Semke, J. Mahdavi, and M. Mathis. Automatic TCP Buffer Tuning. In *ACM SIGCOMM*, pages 315–323, 1998.

[39] B. Tierney. Using NetLogger and Web100 for TCP analysis. *Protocols for High Speed Networks*. http://www-didc.lbl.gov/papers/PFDL.tierney.pdf.

[40] A. Tirumala, L. Cottrell, and T. Dunigan. Measuring end-toend bandwidth with Iperf using Web100. *PAM*, April 2003.

[41] G. Turner. tcpestats: A Net-SNMP AgentX agent implementing the Web100 Project's TCP Extended Statistics MIB, 2002. http://www.aarnet.edu.au/network/software/web100/.

[42] C. Villamizar and C. Song. High performance TCP in ANSNET. *Computer Communications Review*, 24(5), 1995.

[43] Web100. Kernel Instrument Set, 2002. http://www.web100.org/download/kernel/alpha2.0/tcp-kis.txt.

[44] G. Wood. I2-NEWS: Internet2 Land Speed Winners Set New Transcontinental Internet Performance Records, 2002. http://mail.internet2.edu:8080/guest/archives/i2-news/log200003/msg00011.html.

APPENDIX

A. INSTRUMENTATION OVERVIEW

The vast majority of the Web100 kernel instruments (about 120) are suitable for standardization and have already been published as the TCP Extended Statistics MIB [24].

The instruments are naturally grouped into several categories:

Connection state, including the state of the TCP state machine and flags indicating negotiated protocol features such as RFC1323 window scaling, timestamps[18] and RFC2018 SACK[25].

IP traffic, including the number of bytes and segments sent or received. These reflect the network resources consumed by the connection.

Throughput, including total elapsed time and elapsed sequence space. These reflect the performance of the service TCP provides to the application.

Triage, instruments that characterize the protocol events that limit TCP sending rate. See section 7.

Congestion events, that determine the evolution of the congestion window. These are *cwnd* adjustments instrumented in abstract categories. Upward adjustments include Slow-Start and Congestion Avoidance[1]. Downward adjustments include multiplicative decreases, reflecting ECN[36] or packet losses, and non-multiplicative decreases, such as those used by Vegas[3] or Cwnd Validation[15]. Captured value of *cwnd*, *ssthresh*, and their extrema are also included.

Network path properties, as measured by TCP. These are fairly concrete measures of the properties of the underlying IP network, subject to the intrinsic limitation of the algorithms in TCP. The path properties are further sub-divided into:

- Loss and recovery properties, including timeouts, duplicate data and spurious retransmissions.
- Other congestion signals, including ECN, ICMP Source Quench, and back-pressure from the NIC.
- Segment re-ordering
- Timers, including Round Trip Time (RTT) and Retransmission Time Out (RTO) [34].
- Path MTU [29] and Maximum Segment Size.

API usage, including buffer occupancy and tuning[4].

TCP controls and Work-arounds, as introduced by the Net100[31] project. These are non-general TCP controls to improve TCP performance under specific circumstances, but for the most part are not included in the draft MIB. Section 4 describes several of these instruments.

We have been worried about the completeness of our TCP instrumentation. Ideally we would like to be able to claim that our instrumentation has sufficient coverage to diagnose all possible future network and application failures. However, it is really rather unlikely that we haven't missed something. The Web100 project has instead taken an empirical "parallel debugging" approach, by supporting outside users, collaborators and co-developers starting as soon as we had running code and relying on their feedback. While we have received much excellent input, there will always be the possibility that some time in the future we will discover that we overlooked some important instrument.

For more detail on individual instruments please refer to either the TCP Extended Statistics MIB[24] or the KIS database[43].

[4]This is one area where the instrument set is still a little sparse. We encourage researchers interested in API performance to consider additional instruments in this area.