Network Working Group
Request for Comments: 2398
FYI: 33
Category: Informational

S. Parker
C. Schmechel
Sun Microsystems, Inc.
August 1998

Some Testing Tools for TCP Implementors

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

1. Introduction

Available tools for testing TCP implementations are catalogued by this memo. Hopefully disseminating this information will encourage those responsible for building and maintaining TCP to make the best use of available tests. The type of testing the tool provides, the type of tests it is capable of doing, and its availability is enumerated. This document lists only tools which can evaluate one or more TCP implementations, or which can privde some specific results which describe or evaluate the TCP being tested. A number of these tools produce time-sequence plots, see

Tim Shepard's thesis [She91] for a general discussion of these plots.

Each tools is defined as follows:

Name

The name associated with the testing tool.

Category

One or more categories of tests which the tools are capable of providing. Categories used are: functional correctness, performance, stress. Functional correctness tests how stringent a TCP implementation is to the RFC specifications. Performance tests how

quickly a TCP implementation can send and receive data, etc. Stress tests how a TCP implementation is effected under high load conditions.

Description

A description of the tools construction, and the implementation methodology of the tests.

Automation

What steps are required to complete the test? What human intervention is required?

Availability

How do you retrieve this tool and get more information about it?

Required Environment

Compilers, OS version, etc. required to build and/or run the associated tool.

References

A list of publications relating to the tool, if any.

2. Tools

2.1. Dbs

Author
Yukio Murayama

Category
Performance / Stress

Description
Dbs is a tool which allows multiple data transfers to be coordinated, and the resulting TCP behavior to be reviewed. Results are presented as ASCII log files.

Automation
Command of execution is driven by a script file.

Availability
See http://www.ai3.net/products/dbs for details of precise OS versions supported, and for download of the source code. Current implementation supports BSDI BSD/OS, Linux, mkLinux, SunOS, IRIX, Ultrix, NEWS OS, HP-UX. Other environments are likely easy to add.

Required Environment
C language compiler, UNIX-style socket API support.

2.2. Dummynet

Author
Luigi Rizzo

Category
Functional Correctness / Performance

Description
Dummynet is a tool which simulates the presence of finite size queues, bandwidth limitations, and communication delays. Dummynet inserts between two layers of the protocol stack (in the current implementation between TCP and IP), simulating the above effects in an operational system. This way experiments can be done using real protocol implementations and real applications, even running on the same host (dummynet also intercepts communications on the loopback interface). Reconfiguration of dummynet parameters (delay, queue size, bandwidth) can be done on the fly by using a sysctl call. The overhead of dummynet is extremely low.

Automation
Requires merging diff files with kernel source code. Command-line driven through the sysctl command to modify kernel variables.

Availability
See http://www.iet.unipi.it/~luigi/research.html or e-mail Luigi Rizzo (l.rizzo@iet.unipi.it). Source code is available for FreeBSD 2.1 and FreeBSD 2.2 (easily adaptable to other BSD-derived systems).

Required Environment
C language compiler, BSD-derived system, kernel source code.

References
[Riz97]

2.3. Netperf

Author
Rick Jones

Category
Performance

Description
Single connection bandwidth or latency tests for TCP, UDP, and DLPI. Includes provisions for CPU utilization measurement.

Automation
Requires compilation (K&R C sufficient for all but-DHISTOGRAM, may require ANSI C in the future) if starting from source. Execution as child of inetd requires editing of /etc/services and /etc/inetd.conf. Scripts are provided for a quick look (snapshot_script), bulk throughput of TCP and UDP, and latency for TCP and UDP. It is command-line driven.

Availability
See http://www.cup.hp.com/netperf/NetperfPage.html or e-mail Rick Jones (raj@cup.hp.com). Binaries are available here for HP/UX Irix, Solaris, and Win32.

Required Environment
C language compiler, POSIX.1, sockets.

2.4. NIST Net

Author
Mark Carson

Category
Functional Correctness / Performance

Description
NIST Net is a network emulator. The tool is packaged as a Linux kernel patch, a kernel module, a set of programming APIs, and command-line and X-based user interfaces.

NIST Net works by turning the system into a "selectively bad" router - incoming packets may be delayed, dropped, duplicated, bandwidth-constrained, etc. Packet delays may be fixed or randomly distributed, with loadable probability distributions. Packet loss may be uniformly distributed (constant loss probability) or congestion-dependent (probability of loss increases with packet queue lengths). Explicit congestion notifications may optionally be sent

in place of congestion-dependent loss.

Automation
To control the operation of the emulator, there is an interactive user interface, a non-interactive command-line interface, and a set of APIs. Any or all of these may be used in concert. The interactive interface is suitable for simple, spur-of-the-moment testing, while the command-line or APIs may be used to create scripted, non-interactive tests.

Availability
NIST Net is available for public download from the NIST Net web site, http://www.antd.nist.gov/itg/nistnet/. The web site also has installation instructions and documentation.

Required Environment
NIST Net requires a Linux installtion, with kernel version 2.0.27 - 2.0.33. A kernel source tree and build tools are required to build and install the NIST Net components. Building the X interface requires a version of XFree86 (Current Version is 3.3.2). An Athena-replacement widget set such as neXtaw (http://www.inf.ufrgs.br/~kojima/nextaw/) is also desirable for an improved user interface.

NIST Net should run on any i386-compatible machine capable of running Linux, with one or more interfaces.

2.5. Orchestra

Author
Scott Dawson, Farnam Jahanian, and Todd Mitton

Category
Functional Correctness / Performance

Description
This tool is a library which provides the user with an ability to build a protocol layer capable of performing fault injection on protocols. Several fault injection layers have been built using this library, one of which has been used to test different vendor implementations of TCP. This is accomplished by probing the vendor implementation from one machine containing a protocol stack that has been instrumented with Orchestra. A connection is opened from the vendor TCP implementation to the machine which has been instrumented. Faults may then be injected at the Orchestra side of the connection and the vendor TCP's response may be monitored. The most recent version of Orchestra runs inside the X-kernel protocol stack on the OSF MK operating system.

When using Orchestra to test a protocol, the fault injection layer is placed below the target protocol in the protocol stack. This can either be done on one machine on the network, if protocol stacks on the other machines cannot be modified (as in the case of testing TCP), or can be done on all machines on the network (as in the case of testing a protocol under development). Once the fault injection layer is in the protocol stack, all messages sent by and destined for the target protocol pass through it on their way to/from the network. The Orchestra fault injection layer can manipulate these messages. In particular, it can drop, delay, re-order, duplicate, or modify messages. It can also introduce new messages into the system if desired.

The actions of the Orchestra fault injection layer on each message are determined by a script, written in Tcl. This script is interpreted by the fault injection layer when the message enters the layer. The script has access to the header information about the message, and can make decisions based on header values. It can also keep information about previous messages, counters, or any other data which the script writer deems useful. Users of Orchestra may also define their own actions to be taken on messages, written in C, that may be called from the fault injection scripts.

Automation
Scripts can be specified either using a graphical user interface which generates Tcl, or by writing Tcl directly. At this time, post-analysis of the results of the test must also be performed by the user. Essentially this consists of looking at a packet trace that Orchestra generates for (in)correct behavior. Must compile and link fault generated layer with the protocol stack.

Availability
See http://www.eecs.umich.edu/RTCL/projects/orchestra/ or e-mail Scott Dawson (sdawson@eecs.umich.edu).

Required Environment OSF MK operating system, or X-kernel like network architecture, or adapted to network stack.

References
[DJ94], [DJM96a], [DJM96b]

2.6. Packet Shell

Author
Steve Parker and Chris Schmechel

Category
Functional Correctness / Performance

Description
An extensible Tcl/Tk based software toolset for protocol development and testing. Tcl (Tool Command Language) is an embeddable scripting language and Tk is a graphical user interface toolkit based on Tcl. The Packet Shell creates Tcl commands that allow you to create, modify, send, and receive packets on networks. The operations for each protocol are supplied by a dynamic linked library called a protocol library. These libraries are silently linked in from a special directory when the Packet Shell begins execution. The current protocol libraries are: IP, IPv6, IPv6 extensions, ICMP, ICMPv6, Ethernet layer, data layer, file layer (snoop and tcpdump support), socket layer, TCP, TLI.

It includes harness, which is a Tk based graphical user interface for creating test scripts within the Packet Shell. It includes tests for no initial slow start, and retain out of sequence data as TCP test cases mentioned in [PADHV98].

It includes tcpgraph, which is used with a snoop or tcpdump capture file to produce a TCP time-sequence plot using xplot.

Automation
Command-line driven through Tcl commands, or graphical user interface models are available through the harness format.

Availability
See http://playground.sun.com/psh/ or e-mail owner-packet-shell@sunroof.eng.sun.com.

Required Environment

Solaris 2.4 or higher. Porting required for other operating systems.

2.7. Tcpanaly

Author
Vern Paxson

Category
Functional Correctness / Performance

Description
This is a tool for automatically analyzing a TCP implementation's behavior by inspecting packet traces of the TCP's activity. It does so through packet filter traces produced by tcpdump. It has coded within it knowledge of a large number of TCP implementations. Using this, it can determine whether a given trace appears consistent with a given implementation, and, if so, exactly why the TCP chose to transmit each packet at the time it did. If a trace is found inconsistent with a TCP, tcpanaly either diagnoses a likely measurement error present in the trace, or indicates exactly whether the activity in the trace deviates from that of the TCP, which can greatly aid in determining how the traced implementation behaves.

Tcpanaly's category is somewhat difficult to classify, since it attempts to profile the behavior of an implementation, rather than to explicitly test specific correctness or performance issues. However, this profile identifies correctness and performance problems.

Adding new implementations of TCP behavior is possible with tcpanaly through the use of C++ classes.

Automation
Command-line driven and only the traces of the TCP sending and receiving bulk data transfers are needed as input.

Availability
Contact Vern Paxson (vern@ee.lbl.gov).

Required Environment
C++ compiler.

References
[Pax97a]

2.8. Tcptrace

Author
Shawn Ostermann

Category
Functional Correctness / Performance

Description
This is a TCP trace file analysis tool. It reads output trace files in the formats of : tcpdump, snoop, etherpeek, and netm.

For each connection, it keeps track of elapsed time, bytes/segments sent and received, retransmissions, round trip times, window advertisements, throughput, etc from simple to very detailed output.

It can also produce three different types of graphs:

Time Sequence Graph (shows the segments sent and ACKs returned as a function of time)

Instantaneous Throughput (shows the instantaneous, averaged over a few segments, throughput of the connection as a function of time).

Round Trip Times (shows the round trip times for the ACKs as a function of time)

Automation
Command-line driven, and uses the xplot program to view the graphs.

Availability
Source code is available, and Solaris binary along with sample traces. See http://jarok.cs.ohiou.edu/software/tcptrace/tcptrace.html or e-mail Shawn Ostermann (ostermann@cs.ohiou.edu).

Required Environment
C compiler, Solaris, FreeBSD, NetBSD, HPUX, Linux.

2.9. Tracelook

Author
Greg Minshall

Category
Functional Correctness / Performance

Description
This is a Tcl/Tk program for graphically viewing the contents of tcpdump trace files. When plotting a connection, a user can select various variables to be plotted. In each direction of the connection, the user can plot the advertised window in each packet, the highest sequence number in each packet, the lowest sequence number in each packet, and the acknowledgement number in each packet.

Automation
Command-line driven with a graphical user interface for the graph.

Availability
See http://www.ipsilon.com/~minshall/sw/tracelook/tracelook.html or e-mail Greg Minshall (minshall@ipsilon.com).

Required Environment
A modern version of awk, and Tcl/Tk (Tk version 3.6 or higher). The program xgraph is required to view the graphs under X11.

2.10. TReno

Author
Matt Mathis and Jamshid Mahdavi

Category
Performance

Description
This is a TCP throughput measurement tool based on sending UDP or ICMP packets in patterns that are controlled at the user-level so that their timing reflects what would be sent by a TCP that observes proper congestion control (and implements SACK). This allows it to measure throughput independent of the TCP implementation of end hosts and serve as a useful platform for prototyping TCP changes.

Automation
Command-line driven. No "server" is required, and it only requires a single argument of the machine to run the test to.

Availability
See http://www.psc.edu/networking/treno_info.html or e-mail Matt Mathis (mathis@psc.edu) or Jamshid Mahdavi (mahdavi@psc.edu).

Required Environment
C compiler, POSIX.1, raw sockets.

2.11. Ttcp

Author
Unknown

Category
Performance

Description
Originally written to move files around, ttcp became the classic throughput benchmark or load generator, with the addition of support for sourcing to/from memory. It can also be used as a traffic absorber. It has spawned many variants, recent ones include support for UDP, data pattern generation, page alignment, and even alignment offset control.

Automation
Command-line driven.

Availability
See ftp://ftp.arl.mil/pub/ttcp/ or e-mail ARL (ftp@arl.mil) which includes the most common variants available.

Required Environment
C compiler, BSD sockets.

2.12. Xplot

Author
Tim Shepard

Category
Functional Correctness / Performance

Description
This is a fairly conventional graphing/plotting tool (xplot itself), a script to turn tcpdump output into xplot input, and some sample code to generate xplot commands to plot the TCP time-sequence graph).

Automation
Command-line driven with a graphical user interface for the plot.

Availability
See ftp://mercury.lcs.mit.edu/pub/shep/xplot.tar.gz or e-mail Tim Shepard (shep@lcs.mit.edu).

Required Environment
C compiler, X11.

References
[She91]

3. Summary

This memo lists all TCP tests and testing tools reported to the authors as part of TCP Implementer's working group and is not exhaustive. These tools have been verified as available by the authors.

4. Security Considerations

Network analysis tools are improving at a steady pace. The continuing improvement in these tools such as the ones described make security concerns significant.

Some of the tools could be used to create rogue packets or denial-of-service attacks against other hosts. Also, some of the tools require changes to the kernel (foreign code) and might require root privileges to execute. So you are trusting code that you have fetched from some perhaps untrustworthy remote site. This code could contain malicious code that could present any kind of attack.

None of the listed tools evaluate security in any way or form.

There are privacy concerns when grabbing packets from the network in that you are now able to read other people's mail, files, etc. This impacts more than just the host running the tool but all traffic crossing the host's physical network.

5. References

[DJ94] Scott Dawson and Farnam Jahanian, "Probing and Fault Injection of Distributed Protocol Implementations", University of Michigan Technical Report CSE-TR-217-94, EECS Department.

[DJM96a] Scott Dawson, Farnam Jahanian, and Todd Mitton, "ORCHESTRA: A Fault Injection Environment for Distributed Systems", University of Michigan Technical Report CSE-TR-318-96, EECS Department.

[DJM96b] Scott Dawson, Farnam Jahanian, and Todd Mitton, "Experiments on Six Commercial TCP Implementations Using a Software Fault Injection Tool", University of Michigan Technical Report CSE-TR-298-96, EECS Department.

[Pax97a] Vern Paxson, "Automated Packet Trace Analysis of TCP Implementations", ACM SIGCOMM '97, September 1997, Cannes, France.

[PADHV98] Paxson, V., Allman, M., Dawson, S., Heavens, I., and B. Volz, "Known TCP Implementation Problems", Work In Progress.

[Riz97] Luigi Rizzo, "Dummynet: a simple approach to the evaluation of network protocols", ACM Computer Communication Review, Vol. 27, N. 1, January 1997, pp. 31-41.

[She91] Tim Shepard, "TCP Packet Trace Analysis", MIT Laboratory for Computer Science MIT-LCS-TR-494, February, 1991.

6. Authors' Addresses

Steve Parker
Sun Microsystems, Inc.
901 San Antonio Road, UMPK17-202
Palo Alto, CA 94043
USA

Phone: (650) 786-5176
EMail: sparker@eng.sun.com

Chris Schmechel
Sun Microsystems, Inc.
901 San Antonio Road, UMPK17-202
Palo Alto, CA, 94043
USA

Phone: (650) 786-4053
EMail: cschmec@eng.sun.com

7. Full Copyright Statement