Netesto, a Network Testing Toolkit

Lawrence Brakmo

Facebook
Menlo Park, USA
brakmo@fb.com

Abstract

Netesto (Network Testing Toolkit) is a suite of tools for running multi-host network experiments that supports the collection and display of relevant data and statistics. Netesto currently supports TCP RPC and STREAM transfers though Netperf [2], and it can be easily expanded to support other protocols and transfer programs.

Netesto can run complex scenarios consisting of multiple individual tests with just a few commands due to its use of macros and libraries. The end result is a csv file, a key-value file and a customizable HTML page consisting of tables and graphs (of RTTs, cwnds, goodputs).

Netesto has proved very valuable for comparing the performance of different TCP variants (such as Cubic, Reno, BBR, NV).

Keywords

Network, TCP, Testing, Linux.

Introduction

Network testing is hard, especially when it involves many hosts and complex traffic patterns. One must start flows at many hosts, and not at the same time, then one should collect enough information to analyze the results, and finally, we need to analyze the results. It is not enough to just collect basic data like goodputs or losses. We need to collect more detailed information to understand why it is performing the way it is.

Because of these difficulties, most network testing tends to be too simplistic. Only basic scenarios are tested, and as a result, we end up deploying technologies that have not been well evaluated. In addition, it is hard to reproduce these tests since all the necessary information is not always provided.

What is needed is a common framework that simplifies the specification of complex scenarios, that makes it easy to run them and to analyze the results. The test specifications can then be shared allowing others to reproduce the tests and to add them to their own testing arsenal.

Netesto (Network Testing Toolkit) was initially developed to support the evaluation of TCP congestion control and congestion avoidance variants in a Linux environment, but can be used for many other purposes. Its primary goals are: (1) multi-host support, (2) support for complex scenarios, (3) integrated collection of relevant information needed to evaluate the results, (4) tools for displaying results, (5) enabling scenario sharing, (6) extensibility, and (7) simplicity.

Approach

A Netesto environment consists of a host acting as the Netesto client (or controller) and hosts acting as Netesto servers (the same host can be both the client and a server).

The Netesto client then:

1. reads a test script
2. directs the Netesto servers to set things up
3. collects initial information from servers
4. starts the flows and starts flow information collection
5. collects local results as key-value pairs
6. copies test data and results back to the client,
7. aggregates the individual key-value data into per-test key-value file
8. adds the key-value data of the test to a csv (comma separated value file) consisting of a line for the aggregate result and additional lines for the individual flows
9. creates graphs of goodputs, RTTs, cwnds
10. processes the csv file and create an HTML page consisting of tables and graphs

In addition to Netesto client and servers, we also have test clients and test servers (all of which are Netesto servers). Test clients are hosts that start flows to test servers. Currently there are 2 types of flows supported: netperf TCP Stream and netperf TCP RR flows. TCP Stream flows send bulk data from test client to test server. TCP RR flows are bidirectional, test clients send requests to test servers which send back replies Figure 1 shows Netesto's components and their interactions.

Typically, a test scenario will consist of many individual tests. For example, a TCP congestion control (CC) testing scenario may start with 2 and 3 flow tests to analyze simple flow interactions, followed by multiple stream tests to analyze the behavior as congestion increases and could end with multiple request/reply flows to measure fairness between small and large flows.

Each of these mini-scenarios should ran multiple times using a range of values for the parameters of: CC version, base RTT, bottleneck bandwidth and buffer size at the bottleneck.

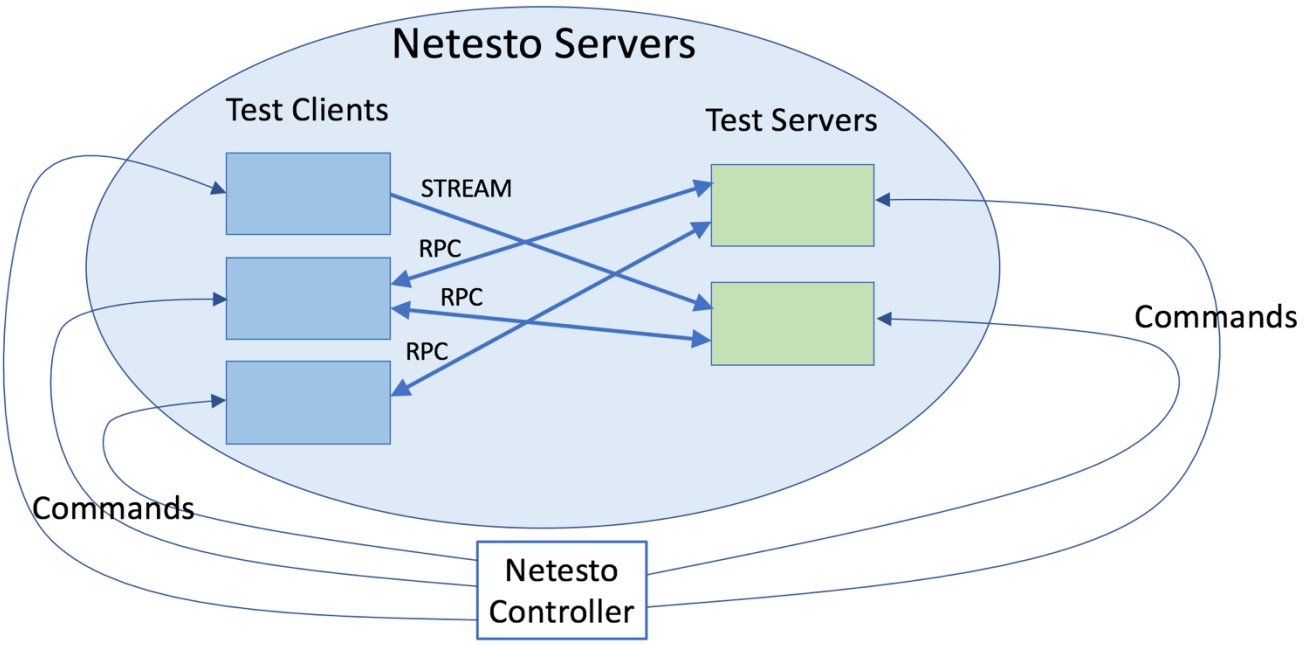

Figure 1: Netesto components and their interactions

Ideally each individual test specified by the mini-scenario and a set of parameter values should ran multiple times, with small perturbations of flow start times. Due to the nature of packet oriented networks, small changes in a setup can result in large changes in the results. For example, in simple two flow stream tests I have seen excellent fairness, where the two flows share the available bandwidth evenly, to 20x differences in the goodputs just by slight changes in the flow start times.

Implementation

As the name implies, Netesto is a collection of tools for network testing. These tools implement the various features of the toolkit and are written primarily in Python and Bash. The main program is *netesto.py*. It runs on both the Netesto servers (netesto.py –s) and at the controller where it reads a Netesto test scenario script, executes local commands and sends remote commands to the appropriate Netesto servers.

The Netesto distribution consists of two directories, a local and a remote directory. The remote directory is copied to Netesto Servers and it consists of netesto.py and other scripts to start the transfers and collect information. The local directory is copied to the Netesto controller and it consists of netesto.py and other programs to process tests data and results.

The initial idea was to specify scenarios in Python, thus having the full power of Python at one's disposal. However, it was ultimately decided to use a new scripting language for two reasons.

1. Simplicity. The scripts are easier to write and read in the scripting language once one is familiar with it.
2. Security. Allows sharing of scripts without worrying about intentional, or unintentional, damage to the hosts.

Netesto Script Language

The script language is a command type language where the 1st word specifies the command. There are two classes of commands, local to be executed by the Netesto controller and remote to be executed by a Netesto server.

Sample Script

```
# Sample Netesto test script
# Run two flow stream test

HOST_SUFFIX mynetwork.com
SOURCE inlib
SET dur=60
SET_EXP dur2=dur/3
SET_EXP delay2=$dur2+3
SET server=host1
SET client1=host2
SET client2=host3
SET ca=bbr

RUN twoFlowBaseStream,1 server=$server
client1=$client1 client2=$client2
expName=1s2c1fs ca1=$ca ca2=$ca dur1=$dur
dur2=$dur2 delay2=$delay2

END
```

The script starts by specifying the default host suffix. Next we read the main macro library (written in Netesto script) containing the macro TwoFlowBaseStream to be executed later. The SET commands are used to set variables, the

SET_EXP command is used to set a variable based on an integer expression. Finally, the RUN command is used to run the macro TwoFlowBaseStream. An optional number of times to run the macro can be specified by a comma followed by an integer. The arguments for the macro are taken from the RUN command line or from the global dictionary (2nd choice). Note that the list of available script commands is shown in Appendix A.

In summary, this script runs a two flow test using BBR[1], where the 2nd flow starts 23 seconds after the 1st one and only lasts 20 seconds. The script of the macro is show next.

TwoFlowBaseStream Script Macro

```
BEGIN TwoFlowBaseStream
SET exp=COUNTER
IF_DEF preServer: RUN preServer host=$server
DO_SERVER exp=$exp host=$server \
   expName=$expName order=0 start=1
#
IF_DEF preClient: RUN preClient \
   host=$client1
IF $tcpDump: DO_TCPDUMP host=$client1 \
   server=$server packets=$tcpDump
DO_CLIENT host=$client1 server=$server \
  ca=$ca1 dur=$dur1 delay=0 \
  instances=$instances test=TCP_STREAM stats=1
#
RAND_WAIT $randDelay
IF_DEF preClient: RUN preClient host=$client2
IF $tcpDump: DO_TCPDUMP host=$client2 \
   server=$server packets=$tcpDump
DO_CLIENT host=$client2 server=$server \
   ca=$ca2 dur=$dur2 delay=$delay2 \
   instances=$instances test=TCP_STREAM \
   stats=1
WAIT $dur1
WAIT 10
DO_SERVER host=$server order=1
WAIT 5
GET_DATA host=$client1
GET_DATA host=$client2
GET_DATA host=$server
WAIT 10
PROCESS_EXP
WAIT 15
END TwoFlowBaseStream
```

The BEGIN command is used to define a macro. The exp variable defines the name of this test instance. When set to COUNTER, it uses the value in the file *counter* after increasing it by one. If the macro preServer is defined, it is ran for the test server host. The DO_SERVER command initializes things at the server (order=0). There is a preClient macro test that applies to the clients. The DO_CLIENT command starts a netperf flow(s) on a Netesto server. RAND_WAIT chooses a uniformly distributed (floating) random number between 0 and its argument and waits that many seconds to introduces variability between runs of the command. The second flow is started (after the specified delay), and we wait until the test is finished. The DO_SERVER command with order=1 collects stats from the server, then we get the data from all the hosts. Finally we process the data of the experiment/test.

A more interesting script is MServerRR which will execute multiple request-reply transfers between many clients to many servers. The main arguments to the macro are *servers, clients and reqs*. The macro will start a netperf TCP_RR flow for each value in reqs (these should be separated by commas, with no spaces in between) between each client and server. An example of its use is:

```
RUN MServerRR servers=host1,host2 \
expName=2s3c3fr clients=host2,host3,host4 \
ca=cubic reqs=10K,100K,1M reply=1 dur=60
```

MServerRR Script Macro

```
BEGIN MServerRR
SET exp=COUNTER
FOR s IN $servers DO
  IF_DEF preServer: RUN preServer host=$s
  DO_SERVER exp=$exp host=$s order=0 \
     expName=$expName start=1
  SET exp=PREV
DONE
FOR c IN $clients DO
  IF_DEF preClient: RUN preClient host=$c
  FOR s IN $servers DO
    IF $tcpDump: DO_TCPDUMP host=$c \
       server=$s packets=$tcpDump
    FOR r IN $reqs DO
      DO_CLIENT host=$c server=$s ca=$ca \
         dur=$dur delay=0 test=TCP_RR \
         instances=$instances req=$r \
         reply=$reply stats=1
      RAND_WAIT $randDelay
    DONE
  DONE
DONE
WAIT $dur
WAIT 10
FOR s IN $servers DO
  DO_SERVER host=$s order=1
DONE
WAIT 5
FOR s IN $servers DO
  GET_DATA host=$s
DONE
FOR c IN $clients DO
  GET_DATA host=$c
DONE
WAIT 10
PROCESS_EXP
WAIT 15
END MServerRR
```

Netesto Output

After each individual test/experiment, a subdirectory is created under the current working directory of the controller with the current exp name. This subdirectory contains the data copied from the Netesto servers that were used (test clients and servers). In particular, there are two subdirectories (0 and 1) containing hosts stats before and after the test. These stats include ethtool, netstat, snmp, snmp5, ifconfig and sysctl output.

Other files in the main test directory are:

- Netperf output files
- Ping output from each client to each server
- Key-value summaries for each flow(s)
- Aggregate key-value summary
- ss output for each flow (every 200ms)

- graphs of rates, rtt, cwnd, etc.
- exp.html file containing a summary of the experiment (including the graphs)
- optional tcpdump files (if requested)

A sample of the summary in exp.html for the TwoFlowBaseStream macro is shown in Figure 2, and graphs samples of Goodput, cwnd and RTT in Figures 3-5.

In addition, new lines are appended to the file *exp.csv* in the current directory containing the aggregate result as well as individual results. The fields (and the order of the fields) used on the exp.csv file is specified by the file *fields.txt*. Each line of this file is of the form:

<field name>:<aggregation>

where *<aggregation>* specifies how the values from the individual flows are aggregated for the aggregate result. Possible values for aggregation are:

- one – just choose one (assumes all are the same)
- all – list all the values from the individual flows
- avg – average the values
- sum – sum the values from the individual flows
- min – minimum
- max – maximum

An example consisting of part of a *fields.txt* file:

```
expName:one
ca:all
dur:one
rtt:avg
rate:sum
minLatency:min
maxLatency:max
```

group	1	2
Test	TCP_STREAM 1M/1	TCP_STREAM 1M/1
host	kerneltest002	kerneltest004
server	kerneltest011	kerneltest011
instances	1	1
dur	60	20
delay	0	23
Ca	bbr	bbr
min/avg/max Rates	7609/7609/7609	4620/4620/4620
min/mean/max Latencies	16/275.29/23396	17/453.50/1123
p50/p90/p99 Latencies	25/753/1541	732/891/1044
rtt	246.0	345.0
pingRtt	209	329
cwnd	169.0	158.0
localRetrans	0	0
remoteRetrans	0	0
lost	0	0
retrans	0	0
retrans_total	0	0
localCpu	0.67	0.63
remoteCpu	1.82	2.87
client-tx-packets	39970699	8090870
client-tx-bytes	60512854442	12248205418
client-tx-packet-len	1513	1513
client-rx-packets	2628169	2483995
client-rx-bytes	226136028	213684379
client-rx-packet-len	86	86
tso	on	on
gso	on	on
lro	off	off
gro	on	on
rx-frames	44	44
tx-frames	16	16
adaptive-rx	on	on

Figure 3: Test summary in exp.html

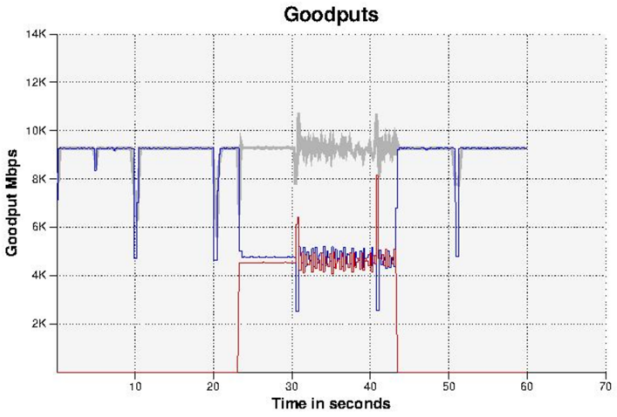

Figure 2: Goodput of 2 BBR flows

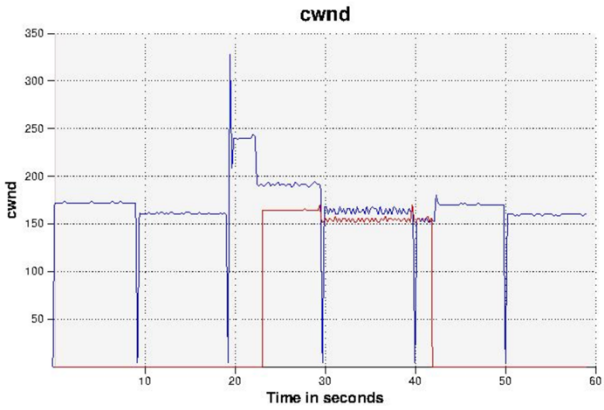

Figure 4: Graph of cwnd of 2 BBR flows

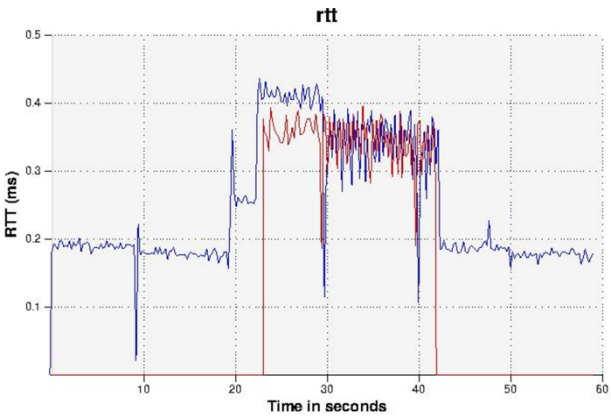

Figure 5: Graph of RTTs of 2 BBR flows

As mentioned earlier, Netesto can create some output automatically when using any of the pre-existing macros. However, while this output provides detailed per test output, it is not sufficient. For example, when analyzing the performance of BBR we also would like to look at summary graphs or tables. Figure 6 shows the Retransmits and Rates for two flow experiments described earlier.

The first element for each bandwidth shows the Cubic results, the 2nd element the BBR result. The bars show the goodput of the 2nd flow that starts in the middle of the 2nd one. With perfect fairness, the 2nd flow should get half of the bandwidth. The red diamonds show the average RTT seen by a ping between client and server done every second.

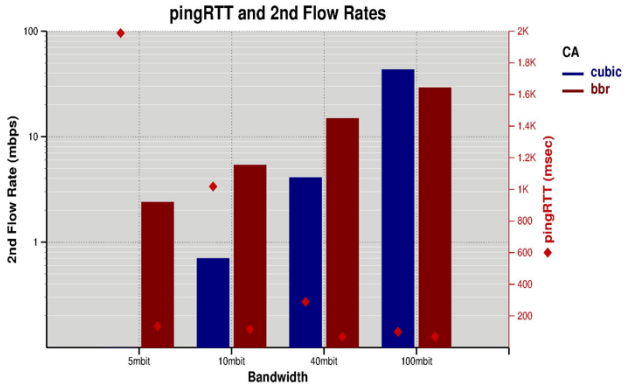

Figure 6: 2nd flow rate and ping RTT, base RTT of 40ms

The graph shows that in most cases BBR achieves better fairness and does not grow the queue size at the bottleneck (as reflected by the RTTs) as compared to Cubic.

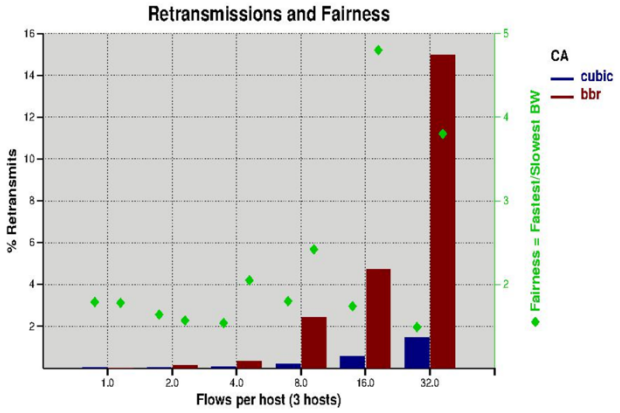

Figure 7: 3 clients sending to 1 server (TCP_STREAM)

Figure 7 shows the results of a different test. This test consists of 3 clients sending to 1 server starting with 1 flow per server and ending at 32 flows per server. The bandwidth is 100Mbps and the base RTT is 10ms.

Figure 7 shows the percent of retransmissions as bars and the fairness as evaluated by the ratio of the bandwidths of the fastest and slowest flow. Retransmissions are larger for BBR and grow up to 15% of packets vs. 1.5% for Cubic. In terms of fairness, a value of 1 is perfect fairness and Cubic starts worse than BBR, but as the load increases BBR's fairness becomes worst. The very high level of retransmissions at higher loads is worrisome.

These graphs were created with the exp.py program which reads the csv data file and a script file containing a series of commands to filter and display the data as tables or graphs. For example, the following script:

```
File=bbr/exp
Path=.
relPath=bbr
IterFile=exp.html
#
# Select rows to use
Select=exp[has].0,testType==3Stream1,netem==10
,bw==100mbit
#
# Create new fairness column
NewColField1=rateMax
NewColOp=NewColDiv
NewColField2=rateMin
NewColName=fairness
#
# Select columns to use
Cols=exp,expName,test,testType,instances,ca,cw
nd,netem,bw,pingRtt,rate,rateMin,rateMax,retra
nsPkts%,fairness
#
# Average rows that have the same values for
the specified columns
#Average=ca,testNote,req,ecn
#
# Write Table
doTable
#
# Write Plot
xUniform=on
Y1Units=1=us,1000=ms,1000000=secs
plotX=instances
plotY1='retransPkts%'
plotY2='fairness'
plotSeries='ca'
plotXTitle='Flows per host (3 hosts)'
plotY1Title='% Retransmits'
plotY2Title='Fairness = Fastest/Slowest BW'
plotSeriesTitle='CA'
plotTitle='Retransmissions and Fairness'
plotXSize=1300
plotYSize=650
doPlot=plotRate
```

Creates a table and a graph. The graph was used to create Figure 7. The whole output of the script is shown in Figure 8.

Other Netesto Features

- Using netem to add delays through the command
  `SET_NETEM host=<hostname> \`
  `  netem_delay=<delay in ms>`
  Some macros will automatically use SET_NETEM if the proper variable is set (macro dependent)
- Collecting tcpdumps through the command
  `DO_TCPDUMP host=<hostname> \`
  `  server=<servername> packets=<# packets>`
  This will collect packets at <hostname> going to <servername>. Some macros will automatically use DO_TCPDUMP when the variable tcpDump is defined to the number of packets to collect.
- Setting qdisc through the command
  `SET_QDISC host=<hostname> qdisc=<qdisc> \`
  `  rate=<rate> burst=<burst> limit=<limit>`

This can be used when doing experiments using a Linux box as a router to set bottleneck bandwidths. The additional use of SET_NETEM can create complex scenarios.

Security

The security features in Netesto are limited to:

- Using a script language for specifying Netesto commands. This prevents scripts from doing bad things. In addition, the command to set sysctls only supports a subset of sysctls.
- There is a clients.txt file used by the Netesto servers to whitelist IPv6 addresses allowed to communicate with the server.

Related Work

Flent[3], The Flexible Network Tester, is "a Python wrapper to run multiple simultaneous network benchmarking tool instances (e.g. netperf/iperf/ping) and aggregate the results."

Availability

Netesto will be available on April 14 at: https://github.com/facebook/fbkutils

References

1. N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. BBR: Congestion-based congestion control. Queue, 14(5), 2016.
2. Rick Jones. Netperf. Open source benchmarking software. URL: http://www.netperf.org
3. Toke Høiland-Jørgensen. Flent: The FLExible Network Tester. URL: https://flent.org/flent-the-flexible-network-tester.pdf

exp	expName	test	testType	instances	ca	cwnd	netem	bw	pingRtt	rate	rateMin	rateMax	retransPkts%	fairness
1514.0	1s3c01fs	TCP_STREAM	3Stream1	1.0	cubic	319.7	10	100mbit	114382.7	94.7	24.9	44.6	0.03	1.79
1515.0	1s3c02fs	TCP_STREAM	3Stream1	2.0	cubic	161.2	10	100mbit	114798.0	94.9	12.9	21.2	0.05	1.64
1516.0	1s3c04fs	TCP_STREAM	3Stream1	4.0	cubic	83.7	10	100mbit	118488.0	94.8	6.1	9.4	0.07	1.54
1517.0	1s3c08fs	TCP_STREAM	3Stream1	8.0	cubic	43.3	10	100mbit	120931.7	94.8	3.0	5.4	0.19	1.8
1518.0	1s3c16fs	TCP_STREAM	3Stream1	16.0	cubic	22.5	10	100mbit	123495.7	94.4	1.6	2.8	0.56	1.74
1519.0	1s3c32fs	TCP_STREAM	3Stream1	32.0	cubic	11.9	10	100mbit	126540.7	93.6	0.8	1.2	1.47	1.49
1520.0	1s3c01fs	TCP_STREAM	3Stream1	1.0	bbr	90.0	10	100mbit	28615.0	93.7	22.2	39.7	0.00	1.78
1521.0	1s3c02fs	TCP_STREAM	3Stream1	2.0	bbr	67.2	10	100mbit	44702.0	94.5	12.3	19.4	0.13	1.57
1522.0	1s3c04fs	TCP_STREAM	3Stream1	4.0	bbr	38.9	10	100mbit	51632.3	94.8	5.9	12.1	0.34	2.05
1523.0	1s3c08fs	TCP_STREAM	3Stream1	8.0	bbr	31.0	10	100mbit	73111.3	95.3	2.8	6.8	2.45	2.42
1524.0	1s3c16fs	TCP_STREAM	3Stream1	16.0	bbr	24.0	10	100mbit	92563.3	95.3	1.0	4.8	4.73	4.8
1525.0	1s3c32fs	TCP_STREAM	3Stream1	32.0	bbr	19.0	10	100mbit	127315.0	94.9	0.5	1.9	14.96	3.8

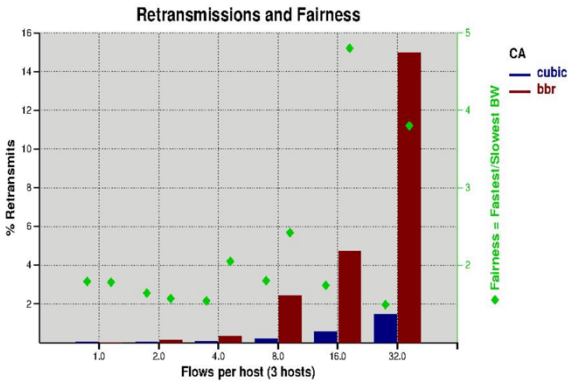

Figure 8: Sample output of exp.py visualization program

Author Biography

Lawrence Brakmo has been a software engineer at Facebook since 2014. Prior to that he was at Google for 8 years and at various research labs (Digital's WRL, HP Labs and DoCoMo USA Labs) before that. While at WRL he was a member of the team that ported Linux to the Itsy, the first handheld using the StrongARM. He is the main author of TCP-Vegas and the author of TCP-NV.

Appendix A: Netesto Command Syntax

SET <var_name>=<val>
Set value of a variable. It can be used later with: $<var_name>

SET_EXP <var_name>=<integer expression>
Set value of a variable using an integer expression consisting of the operators +,-,*,/ and parenthesis

HOST_SUFFIX <host-suffix>
Specify the suffix of a hostname. This suffix will be appended to hostnames lacking a suffix.

SOURCE <filename>
Read from the specified file.

DO_SERVER <arg-list>
Done for hosts that will act as servers (receivers). It should be done at the beginning and at the end to collect host statistics.
arglist:

exp=<expNum> Specify the experiment number/id. Results are stored under the subdirectory <expNum>. If not specified, it will use the previous value. Hence it only needs to be specified one per experiment. If COUNTER is used for the value, the number from the local file counter will be used and its value will be incremented.
host=<hostname> Hostname of the server.
expName=<experiment name> Name of the experiment/test.
order=<0 or 1> Use order=0 when doing it at the beginning of the experiment. User order=1 at the end of the experiment
start=<0 or 1> To start the program nerserver (used by netperf) if it is not already running.

DO_CLIENT <arg-list>
Done for client (sender) hosts. This is the command that starts the network transfers and collects appropriate data.
arglist:

exp=<expNum> Specify the experiment number/id. Results are stored under the subdirectory <expNum>. If not specified, an earlier value will be used. Hence it only needs to be specified once per experiment. If COUNTER is used for the value, the number from the local file counter will be used and its value will be incremented.
host=<hostname> Hostname of the client (sender)
server=<hostname> Hostname of the server (receiver)
expName=<experiment name> Name of the experiment/test. It not specified, an earlier value will be used. Hence it only needs to be specified once per experiment.
ca=<TCP congestion control> Name of TCP congestion control. Examples are: reno, bic, cubic, etc.
dur=<#> Duration of the transfers in seconds.
delay=<#> Delay (in seconds) before starting the transfer. Useful when starting multiple staggered transfers.
instances=<#> Number of transfers (ex. netperf instances) per request size (or per stream)
test=<test type> Currently supported are:
TCP_RR: Netperf TCP request/reply
TCP_STREAM: Netperf TCP stream
req=<request size when doing TCP_RR> Examples: 100 (100 bytes) or 1M (1 MB).
reply=<reply size when doing TCP_RR> Examples: 1 (1 bytes) or 1M (1 MB).
stats=<0|1> Collects stats at the beginning and end of experiment. Use only once per host/per experiment.

WAIT <time in secs>
Wait for specified time before continuing processing commands

RAND_WAIT <time in secs>
A value to wait is chosen randomly from [0 : <time in secs>)

GET_DATA host=<hostname>
Copy experiment data from specified hostname. The data is in a subdirectory named as specified by the exp parameter of the DO_SERVER or DO_CLIENT commands.

PROCESS_EXP
Executed in the local host. Will process the experiment data copied from the remote hosts and update exp.csv as well as create plots for rates, cwnd and rtts as well as an html page for each test. These files are stored in the test subdirectories.

END
Terminates processing of commands.

MACROS
BEGIN <macro name> Starts definition of macro. All commands between the BEGIN and END commands will belong to the new macro.
END <macro name> End definition of macro.
RUN <macro name>[,<reps>] <arg-list> This is how a specified macro is executed. the <reps> parameter specifies how many times to execute the macro. This is a shortcut to run an experiment multiple times. The <arg-list> provides values to any variables used within the commands in the macro. For example, if ca=reno is in <arg-list> and $ca appears in the argument list of a command, the value of $ca will be reno.

For example:

```
HOST_SUFFIX=dc1.mynetwork.com
SOURCE inlib
SET server=kerneltest001
SET client=kerneltest002
RUN OneFLowRR,1 exp=100 server=$server
client=$client expName=1f1hr ca=cubic
dur=60 req=1M reply=1
```

IF $<var_name>: <command>
Execute the specified command if the value of the variable <var_name> is non-zero. For example:

FOR <var_name> in <comma separated list> DO

Example:

```
FOR c in $clients DO
   FOR s in $servers DO
       DO_CLIENT host=$c server=$s ca=$ca\
         dur=$dur delay=0 \
         instances=$instances test=TCP_RR\
         req=$req reply=$reply stats=1
       RAND_WAIT $randDelay
   DONE
DONE
```

SET_NETEM <arglist>
Use netem to add delay (i.e. mimic a host further away).
arglist:

host=<hostname> Host in which to run netem
netem_delay=<delay in ms> Netem delay in ms

SET_SYSCTL <arglist>
Set the specified sysctls
Arglist:

host=<hostname>
<sysctl>=<values>

Supported sysctls:
- net.core.rmem.max
- net.core.wmem.max
- net.ipv4.tcp_wmem
- net.ipv4.tcp_rmem
- net.ipv4.tcp_allowed_congestion_control
- net.ipv4.tcp_ecn
- net.ipv4.tcp_congestion_control

Example:

```
SET_SYSCTL host=host1
net.ipv4.tcp_wmem=10000,262144,20971520
```

DO_TCPDUMP host=<host> server=<server> packets=<number of packets to collect>
Starts tcpdump at the "host" when the test starts and collects the number of packets specified that are going to the specified "server".

SET_QDISC host=<host> qdisc=<disc name> rate=<bw> burst=<burst> limit=<limit>
Sets the specified qdisc as root in the specified "host> and sets the other parameters as indicated.

Appendix B: Main Library (inlib)

Creates macros for basic tests. Examples:

MServerRR - Multiple (>=1) clients doing back-to-back Req/Reply to multiple servers
Args: exp, servers, clients, expName, ca, dur, instances, reqs, reply
Note: servers, clients, reqs, ca can have multiple values separated by commas with no spaces.
Example: ca=cubic,nv,cdg
Ex:

```
RUN MServerRR servers=$servers \
  clients=$clients expName=1s1c1fr ca=$ca \
  dur=$dur delay=0 instances=$instances \
  reqs=$reqs reply=$reply
RUN MServerRR servers=host1,host2 \
  clients=host3 expName=1s2c1fr \
  ca=cubic,bbr dur=60 delay=0 instances=1 \
  reqs=10K,1M reply=100
```

MServerStream - Same as above but doing streaming transfers
Args: exp, servers, clients, expName, ca, dur, Instances
Note: servers, clients, ca can have multiple values separated by commas with no spaces
Ex:

```
RUN MServerStream servers=host1,host2 \
  clients=host3 expName=1s2c1fs \
  ca=cubic,bbr dur=60 delay=0 instances=1
```

MServerRRvs - Two sets of clients, each with its own tcp ca.
Args: exp, servers, clients1, clients2, expName, ca1, ca2, dur, instances, reqs, reply
Note: servers, clients, reqs can have multiple values separated by commas with no spaces
Ex:

```
RUN MServerRRvs servers=host1 \
  clients1=host2 clients2=host3 \
  expName=1s2c4frv ca1=cubic ca2=bbr \
  dur=60 delay=0 instances=4 reqs=1M reply=1
```

Appendix C: Rate Test Library (inlib.rateTest)

Consists of the following 29 tests divided into 6 groups. The notation in

```
3->1:  4x 10K,1M        1s3c04.pfr SET rate3p4=1

Means the following:
3->1:       3 clients sending to 1 server,
4x 10K,1M      4 instances of 10KB and 1MB back-to-back RPCs
1s3c04.pfr     experiment name. Means 1server 3 clients 4 RPC instances at each size
SET rate3p4=1  Set variable rate3p4 to enable test

# Basic rate tests
#
# Consists of the following 29 tests:
#        1) 1->1:  1x1M             1s1c01.fr    SET rate1r=1   \_ SET rate1=1
#        2) 1->1:  1XS              1s1c01.fs    SET rate1s=1   /
#        3) 2->1:  1x1M             1s2c01.fr    SET rate2r1=1  \
#        4) 2->1:  1xS              1s2c01.fr    SET rate2s1=1  |
#        5) 2->1:  2x1M             1s2c02.fr    SET rate2r1=1  |- SET rate2=1
#        6) 2->1:  2x16M            1s2c02.fr                   |
#        7) 2->1:  2x1M,10K         1s2c02.1fr   SET rate2r1=1  |
#        8) 2->1:  2x16M,10K        1s2c02.1fr                  /
#        9) 3->1:  1M               1s3c01.0fr   SET rate3r1=1  \
#       10) 3->1:  2x1M             1s3c02.0fr   SET rate3r2=1  |
#       11) 3->1:  4x1M             1s3c04.0fr   SET rate3r4=1  |- SET rate3=1
#       12) 3->1:  8x1M runs twice  1s3c08.0fr                  |
#       13) 3->1: 16x1M runs twice  1s3c16.0fr                  |
#       14) 3->1: 32x1M runs twice  1s3c32.0fr                  /
#         ) 3->1: 1xSTREAM          1s3c01fs     SET rate3s1=1  \
#         ) 3->1: 2xSTREAM          1s3c02fs     SET rate3s2=1  |
#       15) 3->1: 4xSTREAM          1s3c04fs     SET rate3s4=1  |
#       16) 3->1: 8xSTREAM          1s3c08fs                    |- SET rate3s=1
#       17) 3->1:16xSTREAM          1s3c16fs                    /
#       18) 3->1:  1x 10K,1M        1s3c01.pfr SET rate3p1=1  \
#       19) 3->1:  2x 10K,1M        1s3c02.pfr SET rate3p2=1  |
#       20) 3->1:  4x 10K,1M        1s3c04.pfr SET rate3p4=1  |- SET rate3p=1
#       21) 3->1:  8x 10K,1M        1s3c08.pfr SET rate3p8=1  |
#       22) 3->1: 16x 10K,1M        1s3c16.pfr SET rate3p16=1 |
#       23) 3->1: 32x 10K,1M        1s3c32.pfr SET rate3p32=1 /
#       24) 3->1:  1x 8M,1M,50K,10K  1s3c01.xfr \
#       25) 3->1:  2x 8M,1M,50K,10K  1s3c02.xfr |
#       26) 3->1:  4x 8M,1M,50K,10K  1s3c04.xfr |
#       27) 3->1:  8x 8M,1M,50K,10K  1s3c08.xfr |- SET rate3x=1
#       28) 3->1: 16x 8M,1M,50K,10K  1s3c16.xfr |
#       29) 3->1: 32x 8M,1M,50K,10K  1s3c32.xfr /
```