Netesto

Network Testing Tools

Author: Lawrence Brakmo
brakmo<at>fb.com
netesto<at>brakmo.org

Overview

Netesto is a suite of tools for running multi-host network experiments that supports the collection and display of relevant data and statistics. It currently supports TCP RPC and STREAM transfers through netperf, with other type of transfers planned for the future as need arises. In addition to running the netperf transfers, it also runs ping to collect RTT information and it also collects TCP cwnd and rtt for each of the flows as well as retransmits and cpu load for each of the hosts.

Netesto has been developed and testing in a Linux environment and may need changes before it can be used in other environments. It requires the programs `netserver` and `netperf`. It also assumes the program `ss` is available to capture connection information.

The easiest way to use it is to first look at the sample scripts and modified them.

This is the initial release (v0.1). I have been using this tool for the last year and it is fully functional (although some bugs still need to be ironed out). I am currently working on cleaning out the code and adding more comments.

Introduction

When running netesto tests, one machine acts as the controller which communicates with the machines doing the actual sends (clients) and receives (servers). The controller starts the transfers between clients and servers and collects the experiment's data. This data is processed and put into a csv file in the controller, the raw data is also stored in the controller. In order to achieve this, the files in the netesto-remote directory must be copied first to the clients/servers and the netesto daemon must be started from the directory containing these files (./netesto.py -s &).

Another set of files, under the netesto-local directory must also be copied to the controller, and then a set of tests can be started by specifying a particular netesto script (./netesto.py < script.experiment).

For example, the following script starts one TCP RR netperf flow using 1MB requests and 1 byte replies lasting 60 seconds using TCP Cubic.

```
# Run one Request/Reply experiment
HOST_SUFFIX=dc1.mynetwork.com
```

```
SOURCE inlib
SET client=hostname1
SET server=hostname2
RUN MServerRR servers=$server clients=$client expName=1c1s1fr  ca=cubic dur=60
instances=1 reqs=1M reply=1
```

If you want to run multiple flows with different requests sizes, just specify them in the "reqs=" argument. For example,

```
RUN MServerRR servers=$server clients=$client expName=1c1s4fr  ca=cubic dur=60
instances=1 reqs=1M,100K,10K,1K reply=1
```

If you want to do a Stream test, you can replace the last line with:

```
RUN MServerStream servers=$server clients=$client expName=1c1s1fr  ca=cubic dur=60
instances=1
```

If you want to run 4 flows instead of 1, replace "`instances=1`" with "`instances=4`".

If you want 2 clients sending to 2 hosts (each client sends to both servers) use the following:

```
SET client=hostname1,hostname2
SET server=hostname3,hostname4
```

The results of each particular test are stored in a directory in the controlling host. The name of the directory is a number specified in the file "counter" and is incremented after each test. This directory contains the following:

0/: subdirectory containing system info collected before starting the experiment/transfers from all the hosts involved.
1/: subdirectory containing system info collected after the experiment/transfers end from all the hosts involved.
exp.html: overview of results for that experiment that includes the graphs described below.
all.exp.out: results stored as <key>:<value> lines.
rates.jpg: graph of goodput for all flows
cwnd.jpg: graph of cwnd for all flows
rtt.jpg: graph of RTTs (as seen by TCP) for all flows
vegas_minrtt.jpg: graph of minrtt for flows using a ca that collects this (Vegas, NV, etc.)

In addition, it creates or updates the file exp.csv with the results of the latest experiment.

Libraries

There are also macro library files that make running multiple tests very easy. For example, the following script runs multiple (6) network tests between 3 clients and one server. The first test does one 10KB RR flow and one 1MB RR flow between the each client and the server lasting 60 seconds. The second test does two 10KB RR flows and two 1MB RR flows, followed by 4, 8, 16, and 32 RR flows of each size.

```
HOST_SUFFIX=dc1.mynetwork.com
SOURCE inlib
SOURCE inlib.rateTest
SET client3=hostname1,hostname2,hostname3
SET server1=hostname4
SET ca=cubic
SET rate3p=1                               # enable rate3p tests in inlib.rateTest
SET instances=1                            # how many flow instances per host
SET dur=60                                 # duration of each run in seconds
SET reply=1
RUN ExpRate                                # Run enabled test in inlib.rateTest
END
```

To run multiple TCP congestion controls just replace “`SET ca=cubic`” with “`SET ca=cubic,reno,bbr`”.

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
  ```

RateTest 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 /
```

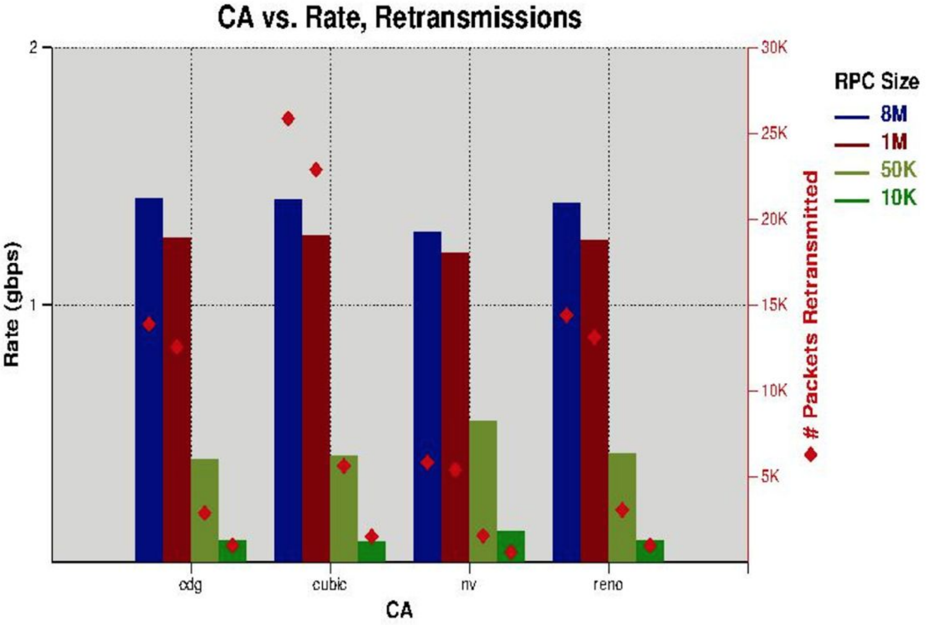
vsTest Library (inlib.vsTest)

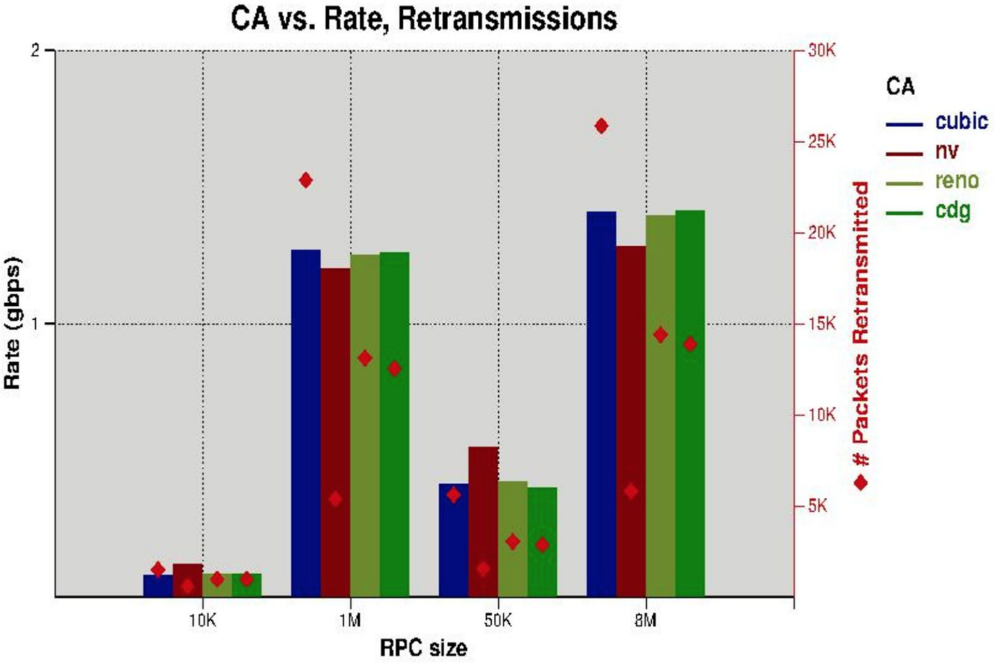
Samples of different versus tests (comparing one TCP ca with another).

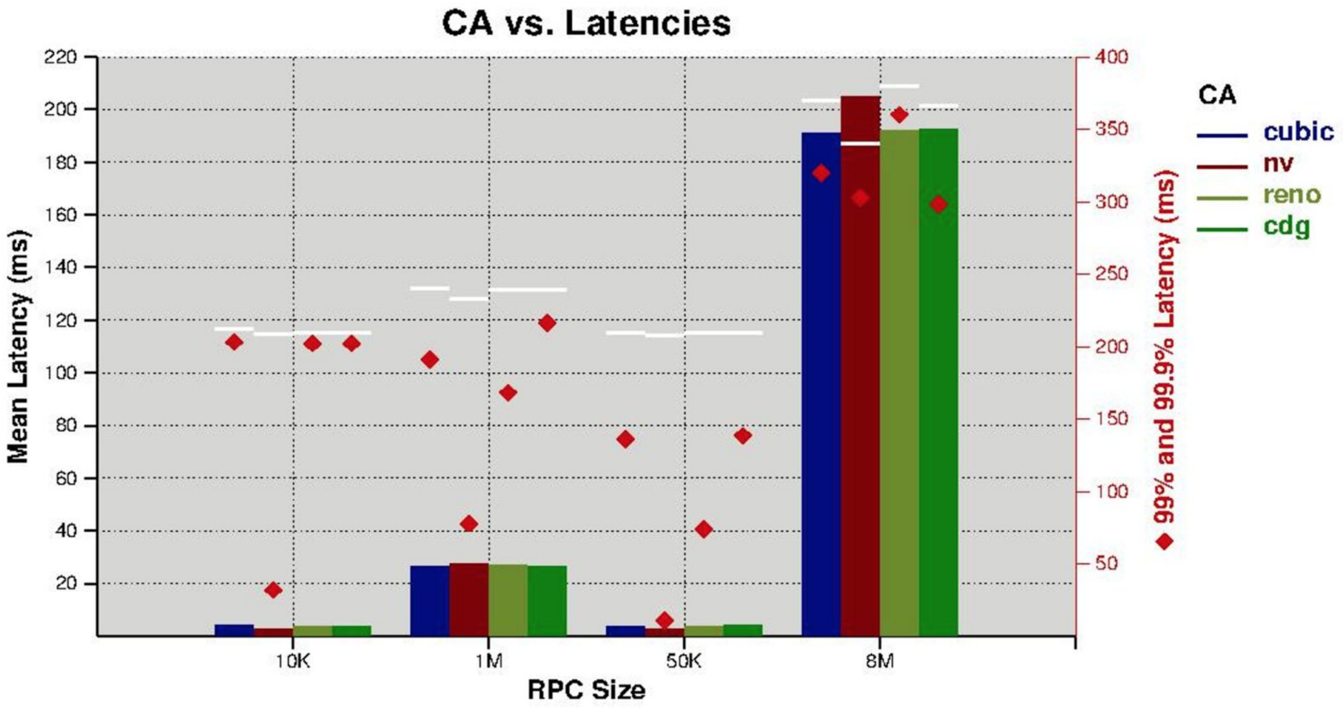
Analysis Tools

There is also a program (exp.py) available to display the data and results in a HTML page. The program allows filtering of the test results, averaging of results, printing tables and graphs. The following images show the output from the sample script of the exp.py program (exp.script). The data used was from a set of experiments comparing cubic, nv, reno and cdg.

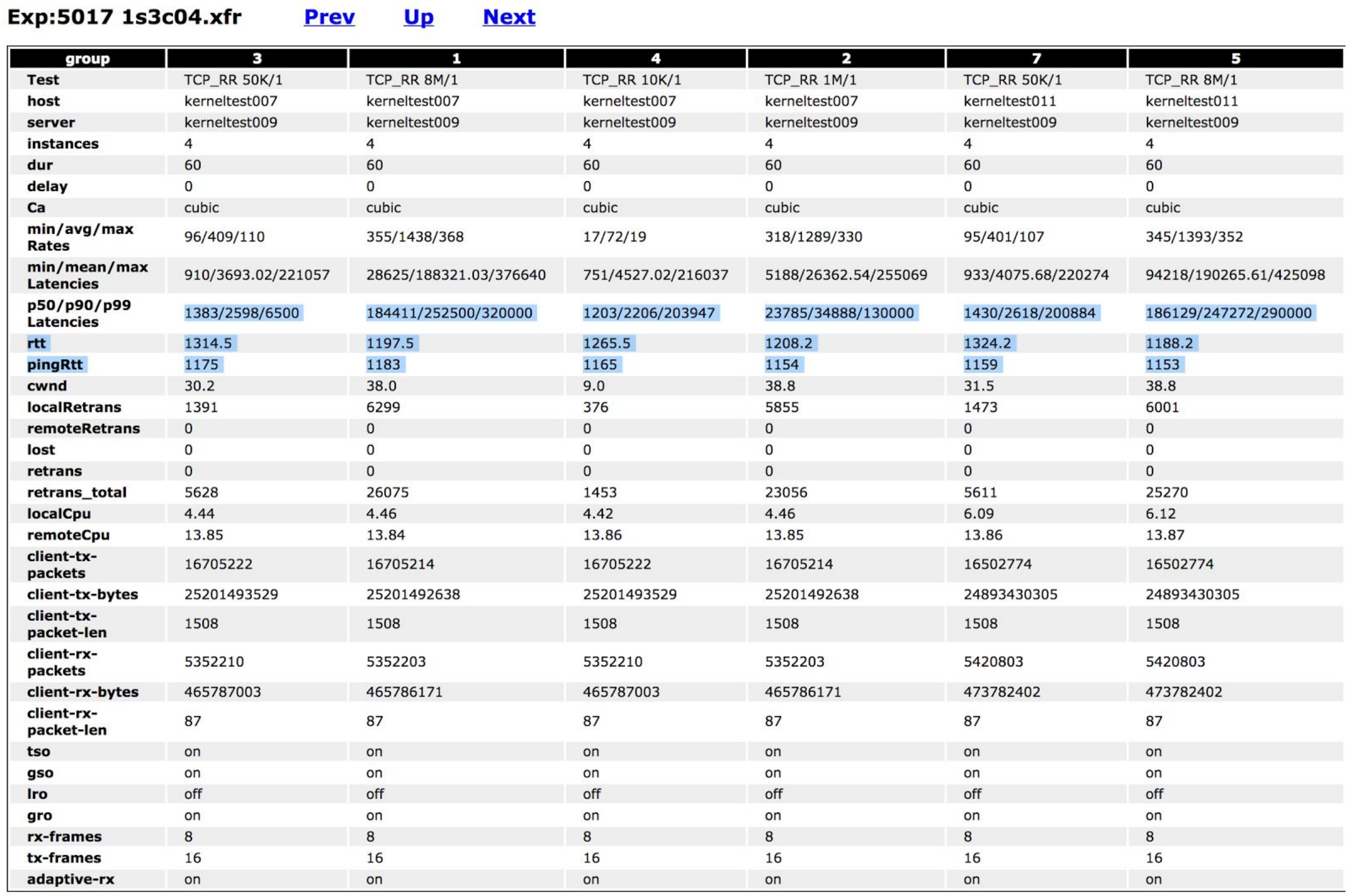
exp	expName	test	ca	req	cwnd	rtt	rate	retransPkts%	meanLatency	p50Latency	p90Latency	p99Latency	p999Latency	retrans_total	avgCnt
5017.10	1s3c04.xfr	TCP_RR	cubic	8M	38.43	1194.90	1408.33		191128.60	186596.67	249685.67	320000.00	370000.00	25847.00	3
5017.10	1s3c04.xfr	TCP_RR	cubic	1M	39.20	1211.40	1267.33		26546.07	23425.00	34391.00	190892.00	240000.00	22864.67	3
5017.11	1s3c04.xfr	TCP_RR	cubic	50K	30.97	1313.17	412.00		3863.47	1391.00	2601.00	135905.67	209697.33	5602.33	3
5017.12	1s3c04.xfr	TCP_RR	cubic	10K	9.00	1265.00	80.67		4114.77	1184.67	1938.67	203051.33	212346.67	1493.67	3
5018.10	1s3c04.xfr	TCP_RR	nv	8M	36.43	1247.33	1283.67		204956.13	208233.33	252078.00	302777.67	340000.00	5812.67	3
5018.10	1s3c04.xfr	TCP_RR	nv	1M	36.80	1263.83	1199.67		27463.07	25715.33	35616.00	77666.67	233333.00	5379.33	3
5018.11	1s3c04.xfr	TCP_RR	nv	50K	27.83	1342.60	548.33		3046.47	1371.67	2646.67	10850.00	208002.00	1542.33	3
5018.12	1s3c04.xfr	TCP_RR	nv	10K	8.00	1375.43	120.00		2670.67	1253.33	1733.67	31878.33	208335.67	581.67	3
5057.10	1s3c04.xfr	TCP_RR	reno	1M	40.00	1235.67	1253.00		27253.03	23729.67	38137.33	168125.00	238771.00	13100.67	3
5057.10	1s3c04.xfr	TCP_RR	reno	8M	38.93	1221.27	1395.67		192202.03	191938.00	252916.67	360000.00	380000.00	14382.33	3
5057.11	1s3c04.xfr	TCP_RR	reno	50K	29.40	1310.63	423.00		3737.90	1362.67	2661.67	73774.67	209062.00	3033.67	3
5057.12	1s3c04.xfr	TCP_RR	reno	10K	9.00	1275.47	82.67		3812.00	1225.00	1390.33	202056.33	209246.00	966.67	3
5058.10	1s3c04.xfr	TCP_RR	cdg	1M	39.67	1240.93	1259.33		26772.50	23249.67	38005.67	216111.00	239047.33	12527.67	3
5058.10	1s3c04.xfr	TCP_RR	cdg	8M	39.63	1225.07	1411.00		192725.47	195148.67	252042.33	298333.33	366666.67	13862.67	3
5058.11	1s3c04.xfr	TCP_RR	cdg	50K	27.87	1319.57	399.67		4101.40	1375.33	2754.33	138422.67	209200.33	2853.67	3
5058.12	1s3c04.xfr	TCP_RR	cdg	10K	8.87	1284.33	83.67		3885.43	1231.33	1406.33	202119.33	209319.33	959.00	3





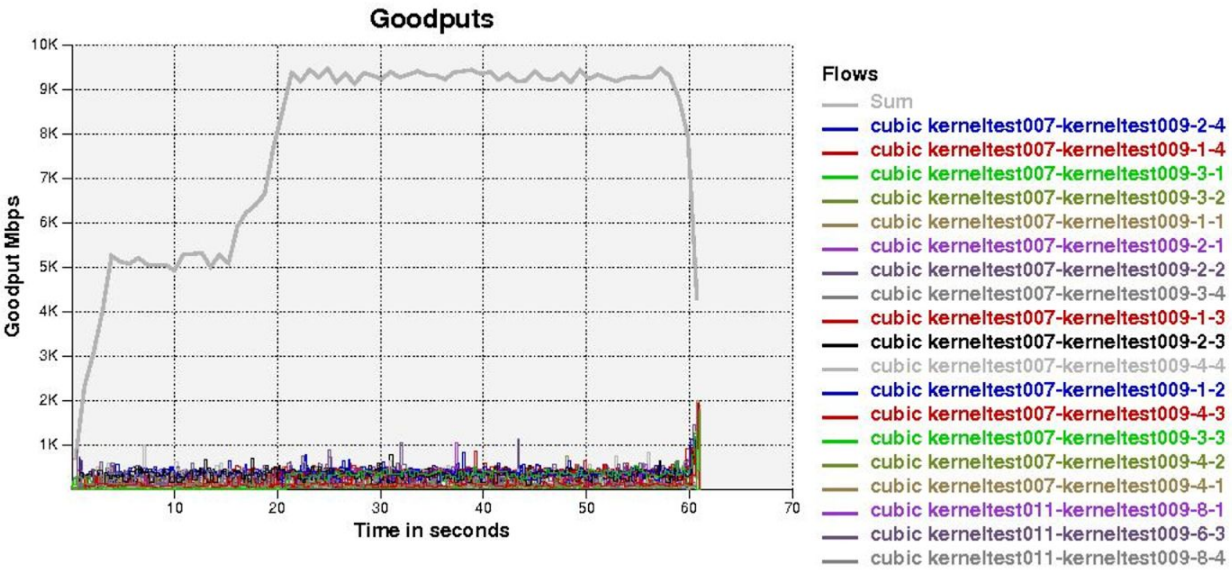


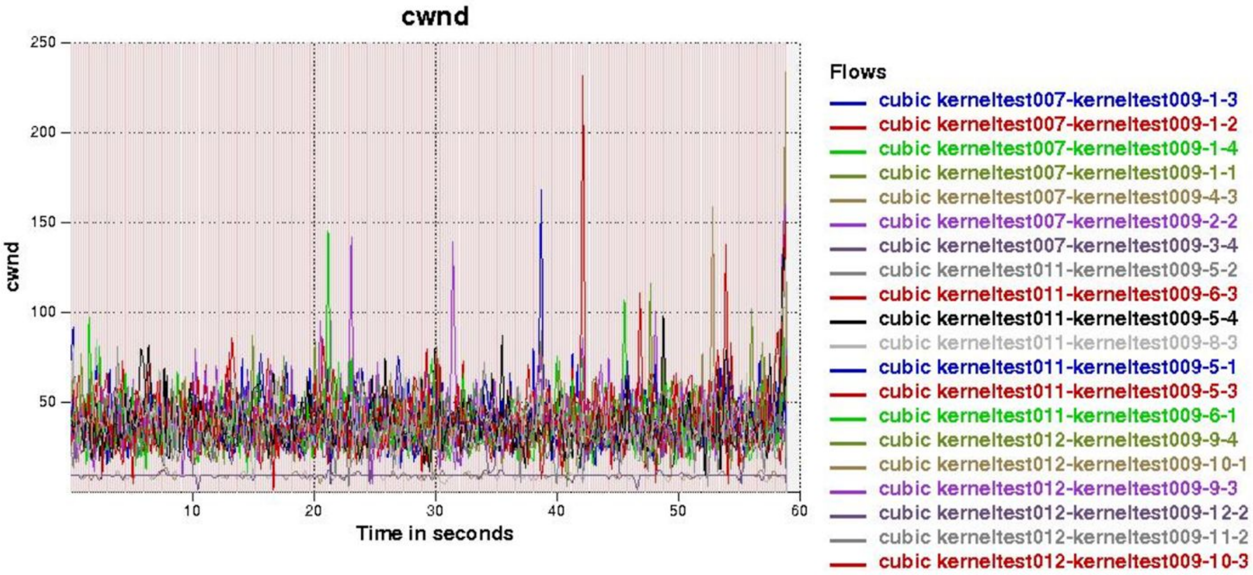
The experiment numbers in the table are links, which when clicked, show a summary for that particular test including various graphs such as cwnd, rates, rtt, etc. The images below show a sample:

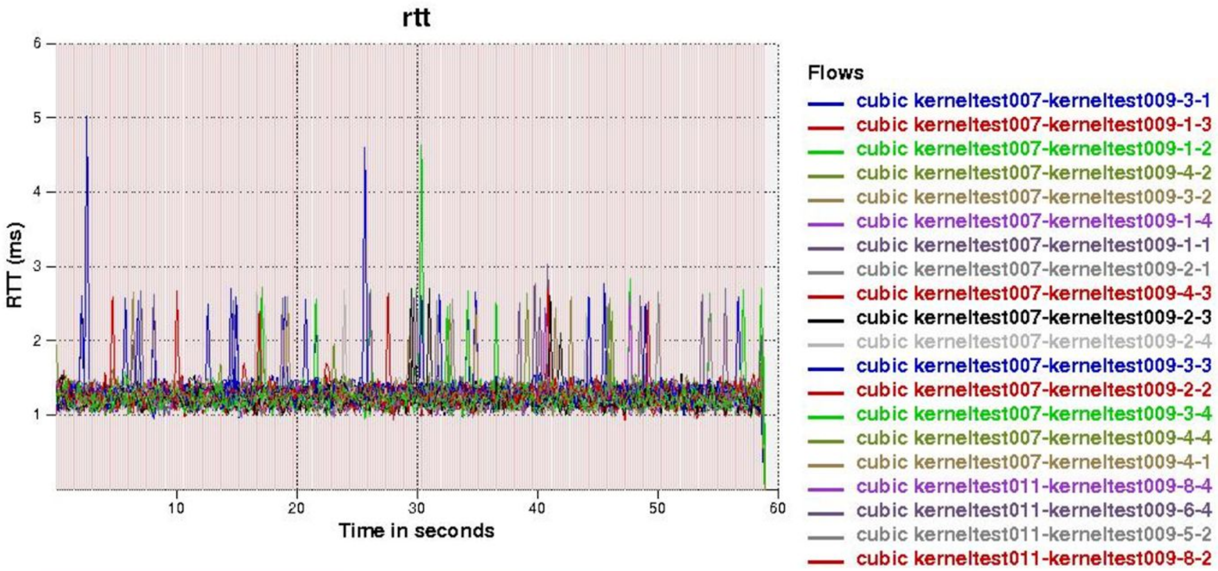


Exp:5017 1s3c04.xfr Prev Up Next

group	3	1	4	2	7	5
Test	TCP_RR 50K/1	TCP_RR 8M/1	TCP_RR 10K/1	TCP_RR 1M/1	TCP_RR 50K/1	TCP_RR 8M/1
host	kerneltest007	kerneltest007	kerneltest007	kerneltest007	kerneltest011	kerneltest011
server	kerneltest009	kerneltest009	kerneltest009	kerneltest009	kerneltest009	kerneltest009
instances	4	4	4	4	4	4
dur	60	60	60	60	60	60
delay	0	0	0	0	0	0
Ca	cubic	cubic	cubic	cubic	cubic	cubic
min/avg/max Rates	96/409/110	355/1438/368	17/72/19	318/1289/330	95/401/107	345/1393/352
min/mean/max Latencies	910/3693.02/221057	28625/188321.03/376640	751/4527.02/216037	5188/26362.54/255069	933/4075.68/220274	94218/190265.61/425098
p50/p90/p99 Latencies	1383/2598/6500	184411/252500/320000	1203/2206/203947	23785/34888/130000	1430/2618/200884	186129/247272/290000
rtt	1314.5	1197.5	1265.5	1208.2	1324.2	1188.2
pingRtt	1175	1183	1165	1154	1159	1153
cwnd	30.2	38.0	9.0	38.8	31.5	38.8
localRetrans	1391	6299	376	5855	1473	6001
remoteRetrans	0	0	0	0	0	0
lost	0	0	0	0	0	0
retrans	0	0	0	0	0	0
retrans_total	5628	26075	1453	23056	5611	25270
localCpu	4.44	4.46	4.42	4.46	6.09	6.12
remoteCpu	13.85	13.84	13.86	13.85	13.86	13.87
client-tx-packets	16705222	16705214	16705222	16705214	16502774	16502774
client-tx-bytes	25201493529	25201492638	25201493529	25201492638	24893430305	24893430305
client-tx-packet-len	1508	1508	1508	1508	1508	1508
client-rx-packets	5352210	5352203	5352210	5352203	5420803	5420803
client-rx-bytes	465787003	465786171	465787003	465786171	473782402	473782402
client-rx-packet-len	87	87	87	87	87	87
tso	on	on	on	on	on	on
gso	on	on	on	on	on	on
lro	off	off	off	off	off	off
gro	on	on	on	on	on	on
rx-frames	8	8	8	8	8	8
tx-frames	16	16	16	16	16	16
adaptive-rx	on	on	on	on	on	on


Goodputs
10K
9K
8K
7K
6K
5K
4K
3K
2K
1K
Goodput Mbps
10
20
30
40
50
60
70
Time in seconds
Flows
Sum
cubic kerneltest007-kerneltest009-2-4
cubic kerneltest007-kerneltest009-1-4
cubic kerneltest007-kerneltest009-3-1
cubic kerneltest007-kerneltest009-3-2
cubic kerneltest007-kerneltest009-1-1
cubic kerneltest007-kerneltest009-2-1
cubic kerneltest007-kerneltest009-2-2
cubic kerneltest007-kerneltest009-3-4
cubic kerneltest007-kerneltest009-1-3
cubic kerneltest007-kerneltest009-2-3
cubic kerneltest007-kerneltest009-4-4
cubic kerneltest007-kerneltest009-1-2
cubic kerneltest007-kerneltest009-4-3
cubic kerneltest007-kerneltest009-3-3
cubic kerneltest007-kerneltest009-4-2
cubic kerneltest007-kerneltest009-4-1
cubic kerneltest011-kerneltest009-8-1
cubic kerneltest011-kerneltest009-6-3
cubic kerneltest011-kerneltest009-8-4


cwnd
250
200
150
100
50
cwnd
10
20
30
40
50
60
Time in seconds
Flows
cubic kerneltest007-kerneltest009-1-3
cubic kerneltest007-kerneltest009-1-2
cubic kerneltest007-kerneltest009-1-4
cubic kerneltest007-kerneltest009-1-1
cubic kerneltest007-kerneltest009-4-3
cubic kerneltest007-kerneltest009-2-2
cubic kerneltest007-kerneltest009-3-4
cubic kerneltest011-kerneltest009-5-2
cubic kerneltest011-kerneltest009-6-3
cubic kerneltest011-kerneltest009-5-4
cubic kerneltest011-kerneltest009-8-3
cubic kerneltest011-kerneltest009-5-1
cubic kerneltest011-kerneltest009-5-3
cubic kerneltest011-kerneltest009-6-1
cubic kerneltest012-kerneltest009-9-4
cubic kerneltest012-kerneltest009-10-1
cubic kerneltest012-kerneltest009-9-3
cubic kerneltest012-kerneltest009-12-2
cubic kerneltest012-kerneltest009-11-2
cubic kerneltest012-kerneltest009-10-3



The red vertical lines in the rtt and cwnd graphs indicate times when packets were retransmitted.

Security

netesto only enforces basic security.

Only a limited subset of commands can be executed on the remote machines (i.e. clients and servers). Netperf transfers can be initiated and only data from the results directories can be copied back to the controller.
The remote machines will only accept connection from a controller in its whitelist. The file `clients.txt` in the netesto directory of the remote machines contains the whitelisted ipv6 addresses (one per line).

Command Syntax

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

`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> Start 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.