Care and Feeding of Netperf

Versions 2.6.0 and Later

Rick Jones rick.jones2@hp.com

This is Rick Jones' feeble attempt at a Texinfo-based manual for the netperf benchmark.

Table of Contents

1 Introduction

Netperf is a benchmark that can be use to measure various aspect of networking performance. The primary foci are bulk (aka unidirectional) data transfer and request/response performance using either TCP or UDP and the Berkeley Sockets interface. As of this writing, the tests available either unconditionally or conditionally include:

- TCP and UDP unidirectional transfer and request/response over IPv4 and IPv6 using the Sockets interface.
- TCP and UDP unidirectional transfer and request/response over IPv4 using the XTI interface.
- Link-level unidirectional transfer and request/response using the DLPI interface.
- Unix domain sockets
- SCTP unidirectional transfer and request/response over IPv4 and IPv6 using the sockets interface.

While not every revision of netperf will work on every platform listed, the intention is that at least some version of netperf will work on the following platforms:

- Unix - at least all the major variants.
- Linux
- Windows
- Others

Netperf is maintained and informally supported primarily by Rick Jones, who can perhaps be best described as Netperf Contributing Editor. Non-trivial and very appreciated assistance comes from others in the network performance community, who are too numerous to mention here. While it is often used by them, netperf is NOT supported via any of the formal Hewlett-Packard support channels. You should feel free to make enhancements and modifications to netperf to suit your nefarious porpoises, so long as you stay within the guidelines of the netperf copyright. If you feel so inclined, you can send your changes to netperf-feedback for possible inclusion into subsequent versions of netperf.

It is the Contributing Editor's belief that the netperf license walks like open source and talks like open source. However, the license was never submitted for "certification" as an open source license. If you would prefer to make contributions to a networking benchmark using a certified open source license, please consider netperf4, which is distributed under the terms of the GPLv2.

The netperf-talk mailing list is available to discuss the care and feeding of netperf with others who share your interest in network performance benchmarking. The netperf-talk mailing list is a closed list (to deal with spam) and you must first subscribe by sending email to netperf-talk-request.

1.1 Conventions

A *sizespec* is a one or two item, comma-separated list used as an argument to a command-line option that can set one or two, related netperf parameters. If you wish to set both parameters to separate values, items should be separated by a comma:

```
parameter1,parameter2
```

If you wish to set the first parameter without altering the value of the second from its default, you should follow the first item with a comma:

```
parameter1,
```

Likewise, precede the item with a comma if you wish to set only the second parameter:

```
,parameter2
```

An item with no commas:

```
parameter1and2
```

will set both parameters to the same value. This last mode is one of the most frequently used.

There is another variant of the comma-separated, two-item list called a *optionspec* which is like a sizespec with the exception that a single item with no comma:

```
parameter1
```

will only set the value of the first parameter and will leave the second parameter at its default value.

Netperf has two types of command-line options. The first are global command line options. They are essentially any option not tied to a particular test or group of tests. An example of a global command-line option is the one which sets the test type - '`-t`'.

The second type of options are test-specific options. These are options which are only applicable to a particular test or set of tests. An example of a test-specific option would be the send socket buffer size for a TCP_STREAM test.

Global command-line options are specified first with test-specific options following after a `--` as in:

```
netperf <global> -- <test-specific>
```

2 Installing Netperf

Netperf's primary form of distribution is source code. This allows installation on systems other than those to which the authors have ready access and thus the ability to create binaries. There are two styles of netperf installation. The first runs the netperf server program - netserver - as a child of inetd. This requires the installer to have sufficient privileges to edit the files '`/etc/services`' and '`/etc/inetd.conf`' or their platform-specific equivalents.

The second style is to run netserver as a standalone daemon. This second method does not require edit privileges on '`/etc/services`' and '`/etc/inetd.conf`' but does mean you must remember to run the netserver program explicitly after every system reboot.

This manual assumes that those wishing to measure networking performance already know how to use anonymous FTP and/or a web browser. It is also expected that you have at least a passing familiarity with the networking protocols and interfaces involved. In all honesty, if you do not have such familiarity, likely as not you have some experience to gain before attempting network performance measurements. The excellent texts by authors such as Stevens, Fenner and Rudoff and/or Stallings would be good starting points. There are likely other excellent sources out there as well.

2.1 Getting Netperf Bits

Gzipped tar files of netperf sources can be retrieved via anonymous FTP for "released" versions of the bits. Pre-release versions of the bits can be retrieved via anonymous FTP from the experimental subdirectory.

For convenience and ease of remembering, a link to the download site is provided via the NetperfPage

The bits corresponding to each discrete release of netperf are tagged for retrieval via subversion. For example, there is a tag for the first version corresponding to this version of the manual - netperf 2.6.0. Those wishing to be on the bleeding edge of netperf development can use subversion to grab the top of trunk. When fixing bugs or making enhancements, patches against the top-of-trunk are preferred.

There are likely other places around the Internet from which one can download netperf bits. These may be simple mirrors of the main netperf site, or they may be local variants on netperf. As with anything one downloads from the Internet, take care to make sure it is what you really wanted and isn't some malicious Trojan or whatnot. Caveat downloader.

As a general rule, binaries of netperf and netserver are not distributed from ftp.netperf.org. From time to time a kind soul or souls has packaged netperf as a Debian package available via the apt-get mechanism or as an RPM. I would be most interested in learning how to enhance the makefiles to make that easier for people.

2.2 Installing Netperf

Once you have downloaded the tar file of netperf sources onto your system(s), it is necessary to unpack the tar file, cd to the netperf directory, run configure and then make. Most of the time it should be sufficient to just:

```
gzcat netperf-<version>.tar.gz | tar xf -
cd netperf-<version>
./configure
make
make install
```

Most of the "usual" configure script options should be present dealing with where to install binaries and whatnot.

```
./configure --help
```

should list all of those and more. You may find the `--prefix` option helpful in deciding where the binaries and such will be put during the `make install`.

If the netperf configure script does not know how to automagically detect which CPU utilization mechanism to use on your platform you may want to add a `--enable-cpuutil=mumble` option to the configure command. If you have knowledge and/or experience to contribute to that area, feel free to contact netperf-feedback@netperf.org.

Similarly, if you want tests using the XTI interface, Unix Domain Sockets, DLPI or SCTP it will be necessary to add one or more `--enable-[xti|unixdomain|dlpi|sctp]=yes` options to the configure command. As of this writing, the configure script will not include those tests automagically.

Starting with version 2.5.0, netperf began migrating most of the "classic" netperf tests found in '`src/nettest_bsd.c`' to the so-called "omni" tests (aka "two routines to run them all") found in '`src/nettest_omni.c`'. This migration enables a number of new features such as greater control over what output is included, and new things to output. The "omni" test is enabled by default in 2.5.0 and a number of the classic tests are migrated - you can tell if a test has been migrated from the presence of `MIGRATED` in the test banner. If you encounter problems with either the omni or migrated tests, please first attempt to obtain resolution via netperf-talk@netperf.org or netperf-feedback@netperf.org. If that is unsuccessful, you can add a `--enable-omni=no` to the configure command and the omni tests will not be compiled-in and the classic tests will not be migrated.

Starting with version 2.5.0, netperf includes the "burst mode" functionality in a default compilation of the bits. If you encounter problems with this, please first attempt to obtain help via netperf-talk@netperf.org or netperf-feedback@netperf.org. If that is unsuccessful, you can add a `--enable-burst=no` to the configure command and the burst mode functionality will not be compiled-in.

On some platforms, it may be necessary to precede the configure command with a CFLAGS and/or LIBS variable as the netperf configure script is not yet smart enough to set them itself. Whenever possible, these requirements will be found in '`README.`*`platform`*' files. Expertise and assistance in making that more automagic in the configure script would be most welcome.

Other optional configure-time settings include `--enable-intervals=yes` to give netperf the ability to "pace" its _STREAM tests and `--enable-histogram=yes` to have netperf keep a histogram of interesting times. Each of these will have some effect on the measured result. If your system supports `gethrtime()` the effect of the histogram measurement should be minimized but probably still measurable. For example, the histogram of a netperf TCP_RR test will be of the individual transaction times:

```
netperf -t TCP_RR -H lag -v 2
TCP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to lag.hpl.hp.com (15.
Local /Remote
Socket Size   Request  Resp.   Elapsed  Trans.
Send   Recv   Size     Size    Time     Rate
bytes  Bytes  bytes    bytes   secs.    per sec

16384  87380  1        1       10.00    3538.82
32768  32768
Alignment      Offset
Local  Remote  Local  Remote
Send   Recv    Send   Recv
    8      0       0      0
Histogram of request/response times
UNIT_USEC     :    0:    0:    0:    0:    0:    0:    0:    0:    0:    0
TEN_USEC      :    0:    0:    0:    0:    0:    0:    0:    0:    0:    0
HUNDRED_USEC  :    0: 34480:  111:   13:   12:    6:    9:    3:    4:    7
UNIT_MSEC     :    0:   60:   50:   51:   44:   44:   72:  119:  100:  101
TEN_MSEC      :    0:  105:    0:    0:    0:    0:    0:    0:    0:    0
HUNDRED_MSEC  :    0:    0:    0:    0:    0:    0:    0:    0:    0:    0
UNIT_SEC      :    0:    0:    0:    0:    0:    0:    0:    0:    0:    0
TEN_SEC       :    0:    0:    0:    0:    0:    0:    0:    0:    0:    0
>100_SECS: 0
HIST_TOTAL:      35391
```

The histogram you see above is basically a base-10 log histogram where we can see that most of the transaction times were on the order of one hundred to one-hundred, ninety-nine microseconds, but they were occasionally as long as ten to nineteen milliseconds

The '`--enable-demo=yes`' configure option will cause code to be included to report interim results during a test run. The rate at which interim results are reported can then be controlled via the global '`-D`' option. Here is an example of '`-D`' output:

```
$ src/netperf -D 1.35 -H tardy.hpl.hp.com -f M
MIGRATED TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to tardy.hpl.hp.com (15
Interim result:    5.41 MBytes/s over 1.35 seconds ending at 1308789765.848
Interim result:   11.07 MBytes/s over 1.36 seconds ending at 1308789767.206
Interim result:   16.00 MBytes/s over 1.36 seconds ending at 1308789768.566
Interim result:   20.66 MBytes/s over 1.36 seconds ending at 1308789769.922
Interim result:   22.74 MBytes/s over 1.36 seconds ending at 1308789771.285
Interim result:   23.07 MBytes/s over 1.36 seconds ending at 1308789772.647
Interim result:   23.77 MBytes/s over 1.37 seconds ending at 1308789774.016
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    MBytes/sec

 87380  16384  16384    10.06      17.81
```

Notice how the units of the interim result track that requested by the '`-f`' option. Also notice that sometimes the interval will be longer than the value specified in the '`-D`' option. This is normal and stems from how demo mode is implemented not by relying on interval timers or frequent calls to get the current time, but by calculating how many units of work must be performed to take at least the desired interval.

Those familiar with this option in earlier versions of netperf will note the addition of the "ending at" text. This is the time as reported by a `gettimeofday()` call (or its emulation) with a `NULL` timezone pointer. This addition is intended to make it easier to insert interim results into an rrdtool Round-Robin Database (RRD). A likely bug-riddled example of doing so can be found in '`doc/examples/netperf_interim_to_rrd.sh`'. The time is reported out to milliseconds rather than microseconds because that is the most rrdtool understands as of the time of this writing.

As of this writing, a `make install` will not actually update the files '`/etc/services`' and/or '`/etc/inetd.conf`' or their platform-specific equivalents. It remains necessary to perform that bit of installation magic by hand. Patches to the makefile sources to effect an automagic editing of the necessary files to have netperf installed as a child of inetd would be most welcome.

Starting the netserver as a standalone daemon should be as easy as:

```
$ netserver
Starting netserver at port 12865
Starting netserver at hostname 0.0.0.0 port 12865 and family 0
```

Over time the specifics of the messages netserver prints to the screen may change but the gist will remain the same.

If the compilation of netperf or netserver happens to fail, feel free to contact netperf-feedback@netperf.org or join and ask in netperf-talk@netperf.org. However, it is quite important that you include the actual compilation errors and perhaps even the configure log in your email. Otherwise, it will be that much more difficult for someone to assist you.

2.3 Verifying Installation

Basically, once netperf is installed and netserver is configured as a child of inetd, or launched as a standalone daemon, simply typing:

```
netperf
```

should result in output similar to the following:

```
$ netperf
TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to localhost.localdomain (127.0.
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 87380  16384  16384    10.00    2997.84
```

3 The Design of Netperf

Netperf is designed around a basic client-server model. There are two executables - netperf and netserver. Generally you will only execute the netperf program, with the netserver program being invoked by the remote system's inetd or having been previously started as its own standalone daemon.

When you execute netperf it will establish a "control connection" to the remote system. This connection will be used to pass test configuration information and results to and from the remote system. Regardless of the type of test to be run, the control connection will be a TCP connection using BSD sockets. The control connection can use either IPv4 or IPv6.

Once the control connection is up and the configuration information has been passed, a separate "data" connection will be opened for the measurement itself using the API's and protocols appropriate for the specified test. When the test is completed, the data connection will be torn-down and results from the netserver will be passed-back via the control connection and combined with netperf's result for display to the user.

Netperf places no traffic on the control connection while a test is in progress. Certain TCP options, such as SO_KEEPALIVE, if set as your systems' default, may put packets out on the control connection while a test is in progress. Generally speaking this will have no effect on the results.

3.1 CPU Utilization

CPU utilization is an important, and alas all-too infrequently reported component of networking performance. Unfortunately, it can be one of the most difficult metrics to measure accurately and portably. Netperf will do its level best to report accurate CPU utilization figures, but some combinations of processor, OS and configuration may make that difficult.

CPU utilization in netperf is reported as a value between 0 and 100% regardless of the number of CPUs involved. In addition to CPU utilization, netperf will report a metric called a *service demand*. The service demand is the normalization of CPU utilization and work performed. For a _STREAM test it is the microseconds of CPU time consumed to transfer on KB (K == 1024) of data. For a _RR test it is the microseconds of CPU time consumed processing a single transaction. For both CPU utilization and service demand, lower is better.

Service demand can be particularly useful when trying to gauge the effect of a performance change. It is essentially a measure of efficiency, with smaller values being more efficient and thus "better."

Netperf is coded to be able to use one of several, generally platform-specific CPU utilization measurement mechanisms. Single letter codes will be included in the CPU portion of the test banner to indicate which mechanism was used on each of the local (netperf) and remote (netserver) system.

As of this writing those codes are:

`U` The CPU utilization measurement mechanism was unknown to netperf or netperf/netserver was not compiled to include CPU utilization measurements. The code for the null CPU utilization mechanism can be found in '`src/netcpu_none.c`'.

`I` An HP-UX-specific CPU utilization mechanism whereby the kernel incremented a per-CPU counter by one for each trip through the idle loop. This mechanism was only available on specially-compiled HP-UX kernels prior to HP-UX 10 and is mentioned here only for the sake of historical completeness and perhaps as a suggestion to those who might be altering other operating systems. While rather simple, perhaps even simplistic, this mechanism was quite robust and was not affected by the concerns of statistical methods, or methods attempting to track time in each of user, kernel, interrupt and idle modes which require quite careful accounting. It can be thought-of as the in-kernel version of the looper `L` mechanism without the context switch overhead. This mechanism required calibration.

`P` An HP-UX-specific CPU utilization mechanism whereby the kernel keeps-track of time (in the form of CPU cycles) spent in the kernel idle loop (HP-UX 10.0 to 11.31 inclusive), or where the kernel keeps track of time spent in idle, user, kernel and interrupt processing (HP-UX 11.23 and later). The former requires calibration, the latter does not. Values in either case are retrieved via one of the pstat(2) family of calls, hence the use of the letter `P`. The code for these mechanisms is found in '`src/netcpu_pstat.c`' and '`src/netcpu_pstatnew.c`' respectively.

`K` A Solaris-specific CPU utilization mechanism whereby the kernel keeps track of ticks (eg HZ) spent in the idle loop. This method is statistical and is known to be inaccurate when the interrupt rate is above epsilon as time spent processing interrupts is not subtracted from idle. The value is retrieved via a kstat() call - hence the use of the letter `K`. Since this mechanism uses units of ticks (HZ) the calibration value should invariably match HZ. (Eg 100) The code for this mechanism is implemented in '`src/netcpu_kstat.c`'.

`M` A Solaris-specific mechanism available on Solaris 10 and latter which uses the new microstate accounting mechanisms. There are two, alas, overlapping, mechanisms. The first tracks nanoseconds spent in user, kernel, and idle modes. The second mechanism tracks nanoseconds spent in interrupt. Since the mechanisms overlap, netperf goes through some hand-waving to try to "fix" the problem. Since the accuracy of the handwaving cannot be completely determined, one must presume that while better than the `K` mechanism, this mechanism too is not without issues. The values are retrieved via kstat() calls, but the letter code is set to `M` to distinguish this mechanism from the even less accurate `K` mechanism. The code for this mechanism is implemented in '`src/netcpu_kstat10.c`'.

`L` A mechanism based on "looper"or "soaker" processes which sit in tight loops counting as fast as they possibly can. This mechanism starts a looper process for each known CPU on the system. The effect of processor hyperthreading on the mechanism is not yet known. This mechanism definitely requires calibration. The code for the "looper"mechanism can be found in '`src/netcpu_looper.c`'

`N` A Microsoft Windows-specific mechanism, the code for which can be found in '`src/netcpu_ntperf.c`'. This mechanism too is based on what appears to be a form of micro-state accounting and requires no calibration. On laptops, or other systems which may dynamically alter the CPU frequency to minimize

power consumption, it has been suggested that this mechanism may become slightly confused, in which case using BIOS/uEFI settings to disable the power saving would be indicated.

`S` This mechanism uses '`/proc/stat`' on Linux to retrieve time (ticks) spent in idle mode. It is thought but not known to be reasonably accurate. The code for this mechanism can be found in '`src/netcpu_procstat.c`'.

`C` A mechanism somewhat similar to `S` but using the sysctl() call on BSD-like Operating systems (*BSD and MacOS X). The code for this mechanism can be found in '`src/netcpu_sysctl.c`'.

`Others` Other mechanisms included in netperf in the past have included using the times() and getrusage() calls. These calls are actually rather poorly suited to the task of measuring CPU overhead for networking as they tend to be process-specific and much network-related processing can happen outside the context of a process, in places where it is not a given it will be charged to the correct, or even a process. They are mentioned here as a warning to anyone seeing those mechanisms used in other networking benchmarks. These mechanisms are not available in netperf 2.4.0 and later.

For many platforms, the configure script will chose the best available CPU utilization mechanism. However, some platforms have no particularly good mechanisms. On those platforms, it is probably best to use the "LOOPER" mechanism which is basically some number of processes (as many as there are processors) sitting in tight little loops counting as fast as they can. The rate at which the loopers count when the system is believed to be idle is compared with the rate when the system is running netperf and the ratio is used to compute CPU utilization.

In the past, netperf included some mechanisms that only reported CPU time charged to the calling process. Those mechanisms have been removed from netperf versions 2.4.0 and later because they are hopelessly inaccurate. Networking can and often results in CPU time being spent in places - such as interrupt contexts - that do not get charged to a or the correct process.

In fact, time spent in the processing of interrupts is a common issue for many CPU utilization mechanisms. In particular, the "PSTAT" mechanism was eventually known to have problems accounting for certain interrupt time prior to HP-UX 11.11 (11iv1). HP-UX 11iv2 and later are known/presumed to be good. The "KSTAT" mechanism is known to have problems on all versions of Solaris up to and including Solaris 10. Even the microstate accounting available via kstat in Solaris 10 has issues, though perhaps not as bad as those of prior versions.

The /proc/stat mechanism under Linux is in what the author would consider an "uncertain" category as it appears to be statistical, which may also have issues with time spent processing interrupts.

In summary, be sure to "sanity-check" the CPU utilization figures with other mechanisms. However, platform tools such as top, vmstat or mpstat are often based on the same mechanisms used by netperf.

3.1.1 CPU Utilization in a Virtual Guest

The CPU utilization mechanisms used by netperf are "inline" in that they are run by the same netperf or netserver process as is running the test itself. This works just fine for "bare iron" tests but runs into a problem when using virtual machines.

The relationship between virtual guest and hypervisor can be thought of as being similar to that between a process and kernel in a bare iron system. As such, (m)any CPU utilization mechanisms used in the virtual guest are similar to "process-local" mechanisms in a bare iron situation. However, just as with bare iron and process-local mechanisms, much networking processing happens outside the context of the virtual guest. It takes place in the hypervisor, and is not visible to mechanisms running in the guest(s). For this reason, one should not really trust CPU utilization figures reported by netperf or netserver when running in a virtual guest.

If one is looking to measure the added overhead of a virtualization mechanism, rather than rely on CPU utilization, one can rely instead on netperf _RR tests - path-lengths and overheads can be a significant fraction of the latency, so increases in overhead should appear as decreases in transaction rate. Whatever you do, **DO NOT** rely on the throughput of a _STREAM test. Achieving link-rate can be done via a multitude of options that mask overhead rather than eliminate it.

4 Global Command-line Options

This section describes each of the global command-line options available in the netperf and netserver binaries. Essentially, it is an expanded version of the usage information displayed by netperf or netserver when invoked with the '`-h`' global command-line option.

4.1 Command-line Options Syntax

Revision 1.8 of netperf introduced enough new functionality to overrun the English alphabet for mnemonic command-line option names, and the author was not and is not quite ready to switch to the contemporary '`--mumble`' style of command-line options. (Call him a Luddite if you wish :).

For this reason, the command-line options were split into two parts - the first are the global command-line options. They are options that affect nearly any and every test type of netperf. The second type are the test-specific command-line options. Both are entered on the same command line, but they must be separated from one another by a `--` for correct parsing. Global command-line options come first, followed by the `--` and then test-specific command-line options. If there are no test-specific options to be set, the `--` may be omitted. If there are no global command-line options to be set, test-specific options must still be preceded by a `--`. For example:

```
netperf <global> -- <test-specific>
```

sets both global and test-specific options:

```
netperf <global>
```

sets just global options and:

```
netperf -- <test-specific>
```

sets just test-specific options.

4.2 Global Options

`-a <sizespec>`
: This option allows you to alter the alignment of the buffers used in the sending and receiving calls on the local system.. Changing the alignment of the buffers can force the system to use different copy schemes, which can have a measurable effect on performance. If the page size for the system were 4096 bytes, and you want to pass page-aligned buffers beginning on page boundaries, you could use '`-a 4096`'. By default the units are bytes, but suffix of "G," "M," or "K" will specify the units to be 2^30 (GB), 2^20 (MB) or 2^10 (KB) respectively. A suffix of "g," "m" or "k" will specify units of 10^9, 10^6 or 10^3 bytes respectively. [Default: 8 bytes]

`-A <sizespec>`
: This option is identical to the '`-a`' option with the difference being it affects alignments for the remote system.

`-b <size>`
: This option is only present when netperf has been configure with –enable-intervals=yes prior to compilation. It sets the size of the burst of send calls in a _STREAM test. When used in conjunction with the '`-w`' option it can cause the rate at which data is sent to be "paced."

`-B <string>`
: This option will cause '`<string>`' to be appended to the brief (see -P) output of netperf.

`-c [rate]`
: This option will ask that CPU utilization and service demand be calculated for the local system. For those CPU utilization mechanisms requiring calibration, the options rate parameter may be specified to preclude running another calibration step, saving 40 seconds of time. For those CPU utilization mechanisms requiring no calibration, the optional rate parameter will be utterly and completely ignored. [Default: no CPU measurements]

`-C [rate]`
: This option requests CPU utilization and service demand calculations for the remote system. It is otherwise identical to the '`-c`' option.

`-d`
: Each instance of this option will increase the quantity of debugging output displayed during a test. If the debugging output level is set high enough, it may have a measurable effect on performance. Debugging information for the local system is printed to stdout. Debugging information for the remote system is sent by default to the file '`/tmp/netperf.debug`'. [Default: no debugging output]

`-D [interval,units]`
: This option is only available when netperf is configured with –enable-demo=yes. When set, it will cause netperf to emit periodic reports of performance during the run. [*interval*,*units*] follow the semantics of an optionspec. If specified, *interval* gives the minimum interval in real seconds, it does not have to be whole seconds. The *units* value can be used for the first guess as to how many units of work (bytes or transactions) must be done to take at least *interval* seconds. If omitted, *interval* defaults to one second and *units* to values specific to each test type.

`-f G|M|K|g|m|k|x`
: This option can be used to change the reporting units for _STREAM tests. Arguments of "G," "M," or "K" will set the units to 2^30, 2^20 or 2^10 bytes/s respectively (EG power of two GB, MB or KB). Arguments of "g," ",m" or "k" will set the units to 10^9, 10^6 or 10^3 bits/s respectively. An argument of "x" requests the units be transactions per second and is only meaningful for a request-response test. [Default: "m" or 10^6 bits/s]

`-F <fillfile>`
: This option specified the file from which send which buffers will be pre-filled . While the buffers will contain data from the specified file, the file is not fully transferred to the remote system as the receiving end of the test will not write the contents of what it receives to a file. This can be used to pre-fill the send buffers with data having different compressibility and so is useful when measuring performance over mechanisms which perform compression.

  While previously required for a TCP_SENDFILE test, later versions of netperf removed that restriction, creating a temporary file as needed. While the author cannot recall exactly when that took place, it is known to be unnecessary in version 2.5.0 and later.

`-h` This option causes netperf to display its "global" usage string and exit to the exclusion of all else.

`-H <optionspec>`

This option will set the name of the remote system and or the address family used for the control connection. For example:

```
-H linger,4
```

will set the name of the remote system to "linger" and tells netperf to use IPv4 addressing only.

```
-H ,6
```

will leave the name of the remote system at its default, and request that only IPv6 addresses be used for the control connection.

```
-H lag
```

will set the name of the remote system to "lag" and leave the address family to AF_UNSPEC which means selection of IPv4 vs IPv6 is left to the system's address resolution.

A value of "inet" can be used in place of "4" to request IPv4 only addressing. Similarly, a value of "inet6" can be used in place of "6" to request IPv6 only addressing. A value of "0" can be used to request either IPv4 or IPv6 addressing as name resolution dictates.

By default, the options set with the global '`-H`' option are inherited by the test for its data connection, unless a test-specific '`-H`' option is specified.

If a '`-H`' option follows either the '`-4`' or '`-6`' options, the family setting specified with the -H option will override the '`-4`' or '`-6`' options for the remote address family. If no address family is specified, settings from a previous '`-4`' or '`-6`' option will remain. In a nutshell, the last explicit global command-line option wins.

[Default: "localhost" for the remote name/IP address and "0" (eg AF_UNSPEC) for the remote address family.]

`-I <optionspec>`

This option enables the calculation of confidence intervals and sets the confidence and width parameters with the first half of the optionspec being either 99 or 95 for 99% or 95% confidence respectively. The second value of the optionspec specifies the width of the desired confidence interval. For example

```
-I 99,5
```

asks netperf to be 99% confident that the measured mean values for throughput and CPU utilization are within +/- 2.5% of the "real" mean values. If the '`-i`' option is specified and the '`-I`' option is omitted, the confidence defaults to 99% and the width to 5% (giving +/- 2.5%)

If classic netperf test calculates that the desired confidence intervals have not been met, it emits a noticeable warning that cannot be suppressed with the '`-P`' or '`-v`' options:

```
netperf -H tardy.cup -i 3 -I 99,5
TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to tardy.cup.hp.com (1
```

```
!!! WARNING
!!! Desired confidence was not achieved within the specified iterations.
!!! This implies that there was variability in the test environment that
!!! must be investigated before going further.
!!! Confidence intervals: Throughput      :  6.8%
!!!                       Local CPU util  :  0.0%
!!!                       Remote CPU util :  0.0%

Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 32768  16384  16384    10.01      40.23
```

In the example above we see that netperf did not meet the desired confidence intervals. Instead of being 99% confident it was within +/- 2.5% of the real mean value of throughput it is only confident it was within +/-3.4%. In this example, increasing the '`-i`' option (described below) and/or increasing the iteration length with the '`-l`' option might resolve the situation.

In an explicit "omni" test, failure to meet the confidence intervals will not result in netperf emitting a warning. To verify the hitting, or not, of the confidence intervals one will need to include them as part of an Section 9.3 [Omni Output Selection], page 50 in the test-specific '`-o`', '`-O`' or '`k`' output selection options. The warning about not hitting the confidence intervals will remain in a "migrated" classic netperf test.

`-i <sizespec>`

This option enables the calculation of confidence intervals and sets the minimum and maximum number of iterations to run in attempting to achieve the desired confidence interval. The first value sets the maximum number of iterations to run, the second, the minimum. The maximum number of iterations is silently capped at 30 and the minimum is silently floored at 3. Netperf repeats the measurement the minimum number of iterations and continues until it reaches either the desired confidence interval, or the maximum number of iterations, whichever comes first. A classic or migrated netperf test will not display the actual number of iterations run. An Chapter 9 [The Omni Tests], page 47 will emit the number of iterations run if the `CONFIDENCE_ITERATION` output selector is included in the Section 9.3 [Omni Output Selection], page 50.

If the '`-I`' option is specified and the '`-i`' option omitted the maximum number of iterations is set to 10 and the minimum to three.

Output of a warning upon not hitting the desired confidence intervals follows the description provided for the '`-I`' option.

The total test time will be somewhere between the minimum and maximum number of iterations multiplied by the test length supplied by the '`-l`' option.

`-j`

This option instructs netperf to keep additional timing statistics when explicitly running an Chapter 9 [The Omni Tests], page 47. These can be output when

the test-specific '`-o`', '`-O`' or '`-k`' Section 9.3.1 [Omni Output Selectors], page 51 include one or more of:

- MIN_LATENCY
- MAX_LATENCY
- P50_LATENCY
- P90_LATENCY
- P99_LATENCY
- MEAN_LATENCY
- STDDEV_LATENCY

These statistics will be based on an expanded (100 buckets per row rather than 10) histogram of times rather than a terribly long list of individual times. As such, there will be some slight error thanks to the bucketing. However, the reduction in storage and processing overheads is well worth it. When running a request/response test, one might get some idea of the error by comparing the Section 9.3.1 [Omni Output Selectors], page 51 calculated from the histogram with the `RT_LATENCY` calculated from the number of request/response transactions and the test run time.

In the case of a request/response test the latencies will be transaction latencies. In the case of a receive-only test they will be time spent in the receive call. In the case of a send-only test they will be time spent in the send call. The units will be microseconds. Added in netperf 2.5.0.

`-l testlen`

This option controls the length of any **one** iteration of the requested test. A positive value for *testlen* will run each iteration of the test for at least *testlen* seconds. A negative value for *testlen* will run each iteration for the absolute value of *testlen* transactions for a _RR test or bytes for a _STREAM test. Certain tests, notably those using UDP can only be timed, they cannot be limited by transaction or byte count. This limitation may be relaxed in an Chapter 9 [The Omni Tests], page 47 test.

In some situations, individual iterations of a test may run for longer for the number of seconds specified by the '`-l`' option. In particular, this may occur for those tests where the socket buffer size(s) are significantly longer than the bandwidthXdelay product of the link(s) over which the data connection passes, or those tests where there may be non-trivial numbers of retransmissions.

If confidence intervals are enabled via either '`-I`' or '`-i`' the total length of the netperf test will be somewhere between the minimum and maximum iteration count multiplied by *testlen*.

`-L <optionspec>`

This option is identical to the '`-H`' option with the difference being it sets the _local_ hostname/IP and/or address family information. This option is generally unnecessary, but can be useful when you wish to make sure that the netperf control and data connections go via different paths. It can also come-in handy if one is trying to run netperf through those evil, end-to-end breaking things known as firewalls.

[Default: 0.0.0.0 (eg INADDR_ANY) for IPv4 and ::0 for IPv6 for the local name. AF_UNSPEC for the local address family.]

`-n numcpus`

This option tells netperf how many CPUs it should ass-u-me are active on the system running netperf. In particular, this is used for the Section 3.1 [CPU Utilization], page 7 and service demand calculations. On certain systems, netperf is able to determine the number of CPU's automagically. This option will override any number netperf might be able to determine on its own.

Note that this option does _not_ set the number of CPUs on the system running netserver. When netperf/netserver cannot automagically determine the number of CPUs that can only be set for netserver via a netserver '`-n`' command-line option.

As it is almost universally possible for netperf/netserver to determine the number of CPUs on the system automagically, 99 times out of 10 this option should not be necessary and may be removed in a future release of netperf.

`-N` This option tells netperf to forgo establishing a control connection. This makes it is possible to run some limited netperf tests without a corresponding netserver on the remote system.

With this option set, the test to be run is to get all the addressing information it needs to establish its data connection from the command line or internal defaults. If not otherwise specified by test-specific command line options, the data connection for a "STREAM" or "SENDFILE" test will be to the "discard" port, an "RR" test will be to the "echo" port, and a "MEARTS" test will be to the chargen port.

The response size of an "RR" test will be silently set to be the same as the request size. Otherwise the test would hang if the response size was larger than the request size, or would report an incorrect, inflated transaction rate if the response size was less than the request size.

Since there is no control connection when this option is specified, it is not possible to set "remote" properties such as socket buffer size and the like via the netperf command line. Nor is it possible to retrieve such interesting remote information as CPU utilization. These items will be displayed as values which should make it immediately obvious that was the case.

The only way to change remote characteristics such as socket buffer size or to obtain information such as CPU utilization is to employ platform-specific methods on the remote system. Frankly, if one has access to the remote system to employ those methods one aught to be able to run a netserver there. However, that ability may not be present in certain "support" situations, hence the addition of this option.

Added in netperf 2.4.3.

`-o <sizespec>`

The value(s) passed-in with this option will be used as an offset added to the alignment specified with the '`-a`' option. For example:

```
-o 3 -a 4096
```

will cause the buffers passed to the local (netperf) send and receive calls to begin three bytes past an address aligned to 4096 bytes. [Default: 0 bytes]

`-O <sizespec>`
This option behaves just as the '`-o`' option but on the remote (netserver) system and in conjunction with the '`-A`' option. [Default: 0 bytes]

`-p <optionspec>`
The first value of the optionspec passed-in with this option tells netperf the port number at which it should expect the remote netserver to be listening for control connections. The second value of the optionspec will request netperf to bind to that local port number before establishing the control connection. For example

```
-p 12345
```

tells netperf that the remote netserver is listening on port 12345 and leaves selection of the local port number for the control connection up to the local TCP/IP stack whereas

```
-p ,32109
```

leaves the remote netserver port at the default value of 12865 and causes netperf to bind to the local port number 32109 before connecting to the remote netserver.

In general, setting the local port number is only necessary when one is looking to run netperf through those evil, end-to-end breaking things known as firewalls.

`-P 0|1` A value of "1" for the '`-P`' option will enable display of the test banner. A value of "0" will disable display of the test banner. One might want to disable display of the test banner when running the same basic test type (eg TCP_STREAM) multiple times in succession where the test banners would then simply be redundant and unnecessarily clutter the output. [Default: 1 - display test banners]

`-s <seconds>`
This option will cause netperf to sleep '`<seconds>`' before actually transferring data over the data connection. This may be useful in situations where one wishes to start a great many netperf instances and do not want the earlier ones affecting the ability of the later ones to get established.

Added somewhere between versions 2.4.3 and 2.5.0.

`-S` This option will cause an attempt to be made to set SO_KEEPALIVE on the data socket of a test using the BSD sockets interface. The attempt will be made on the netperf side of all tests, and will be made on the netserver side of an Chapter 9 [The Omni Tests], page 47 or Section 9.2 [Migrated Tests], page 49 test. No indication of failure is given unless debug output is enabled with the global '`-d`' option.

Added in version 2.5.0.

`-t testname`
This option is used to tell netperf which test you wish to run. As of this writing, valid values for *testname* include:

- Section 5.2.1 [TCP_STREAM], page 23, Section 5.2.2 [TCP_MAERTS], page 24, Section 5.2.3 [TCP_SENDFILE], page 25, Section 6.2.1 [TCP_RR], page 31, Section 6.2.3 [TCP_CRR], page 33, Section 6.2.2 [TCP_CC], page 32
- Section 5.2.4 [UDP_STREAM], page 25, Section 6.2.4 [UDP_RR], page 33
- Section 5.2.5 [XTI_TCP_STREAM], page 27, Section 6.2.5 [XTI_TCP_RR], page 34, Section 6.2.7 [XTI_TCP_CRR], page 34, Section 6.2.6 [XTI_TCP_CC], page 34
- Section 5.2.6 [XTI_UDP_STREAM], page 27, Section 6.2.8 [XTI_UDP_RR], page 34
- Section 5.2.7 [SCTP_STREAM], page 27, Section 6.2.11 [SCTP_RR], page 35
- Section 5.2.8 [DLCO_STREAM], page 27, Section 6.2.10 [DLCO_RR], page 34, Section 5.2.9 [DLCL_STREAM], page 28, Section 6.2.9 [DLCL_RR], page 34
- Chapter 10 [Other Netperf Tests], page 65, Chapter 10 [Other Netperf Tests], page 65
- Chapter 9 [The Omni Tests], page 47

Not all tests are always compiled into netperf. In particular, the "XTI," "SCTP," "UNIXDOMAIN," and "DL*" tests are only included in netperf when configured with '`--enable-[xti|sctp|unixdomain|dlpi]=yes`'.

Netperf only runs one type of test no matter how many '`-t`' options may be present on the command-line. The last '`-t`' global command-line option will determine the test to be run. [Default: TCP_STREAM]

`-T <optionspec>`

This option controls the CPU, and probably by extension memory, affinity of netperf and/or netserver.

```
netperf -T 1
```

will bind both netperf and netserver to "CPU 1" on their respective systems.

```
netperf -T 1,
```

will bind just netperf to "CPU 1" and will leave netserver unbound.

```
netperf -T ,2
```

will leave netperf unbound and will bind netserver to "CPU 2."

```
netperf -T 1,2
```

will bind netperf to "CPU 1" and netserver to "CPU 2."

This can be particularly useful when investigating performance issues involving where processes run relative to where NIC interrupts are processed or where NICs allocate their DMA buffers.

`-v verbosity`

This option controls how verbose netperf will be in its output, and is often used in conjunction with the '`-P`' option. If the verbosity is set to a value of "0" then only the test's SFM (Single Figure of Merit) is displayed. If local Section 3.1

[CPU Utilization], page 7 is requested via the '`-c`' option then the SFM is the local service demand. Othersise, if remote CPU utilization is requested via the '`-C`' option then the SFM is the remote service demand. If neither local nor remote CPU utilization are requested the SFM will be the measured throughput or transaction rate as implied by the test specified with the '`-t`' option.

If the verbosity level is set to "1" then the "normal" netperf result output for each test is displayed.

If the verbosity level is set to "2" then "extra" information will be displayed. This may include, but is not limited to the number of send or recv calls made and the average number of bytes per send or recv call, or a histogram of the time spent in each send() call or for each transaction if netperf was configured with '`--enable-histogram=yes`'. [Default: 1 - normal verbosity]

In an Chapter 9 [The Omni Tests], page 47 test the verbosity setting is largely ignored, save for when asking for the time histogram to be displayed. In version 2.5.0 and later there is no Section 9.3.1 [Omni Output Selectors], page 51 for the histogram and so it remains displayed only when the verbosity level is set to 2.

`-V` This option displays the netperf version and then exits.

Added in netperf 2.4.4.

`-w time` If netperf was configured with '`--enable-intervals=yes`' then this value will set the inter-burst time to time milliseconds, and the '`-b`' option will set the number of sends per burst. The actual inter-burst time may vary depending on the system's timer resolution.

`-W <sizespec>`
This option controls the number of buffers in the send (first or only value) and or receive (second or only value) buffer rings. Unlike some benchmarks, netperf does not continuously send or receive from a single buffer. Instead it rotates through a ring of buffers. [Default: One more than the size of the send or receive socket buffer sizes ('`-s`' and/or '`-S`' options) divided by the send '`-m`' or receive '`-M`' buffer size respectively]

`-4` Specifying this option will set both the local and remote address families to AF_INET - that is use only IPv4 addresses on the control connection. This can be overridden by a subsequent '`-6`', '`-H`' or '`-L`' option. Basically, the last option explicitly specifying an address family wins. Unless overridden by a test-specific option, this will be inherited for the data connection as well.

`-6` Specifying this option will set both local and and remote address families to AF_INET6 - that is use only IPv6 addresses on the control connection. This can be overridden by a subsequent '`-4`', '`-H`' or '`-L`' option. Basically, the last address family explicitly specified wins. Unless overridden by a test-specific option, this will be inherited for the data connection as well.

5 Using Netperf to Measure Bulk Data Transfer

The most commonly measured aspect of networked system performance is that of bulk or unidirectional transfer performance. Everyone wants to know how many bits or bytes per second they can push across the network. The classic netperf convention for a bulk data transfer test name is to tack a "_STREAM" suffix to a test name.

5.1 Issues in Bulk Transfer

There are any number of things which can affect the performance of a bulk transfer test.

Certainly, absent compression, bulk-transfer tests can be limited by the speed of the slowest link in the path from the source to the destination. If testing over a gigabit link, you will not see more than a gigabit :) Such situations can be described as being *network-limited* or *NIC-limited*.

CPU utilization can also affect the results of a bulk-transfer test. If the networking stack requires a certain number of instructions or CPU cycles per KB of data transferred, and the CPU is limited in the number of instructions or cycles it can provide, then the transfer can be described as being *CPU-bound*.

A bulk-transfer test can be CPU bound even when netperf reports less than 100% CPU utilization. This can happen on an MP system where one or more of the CPUs saturate at 100% but other CPU's remain idle. Typically, a single flow of data, such as that from a single instance of a netperf _STREAM test cannot make use of much more than the power of one CPU. Exceptions to this generally occur when netperf and/or netserver run on CPU(s) other than the CPU(s) taking interrupts from the NIC(s). In that case, one might see as much as two CPUs' worth of processing being used to service the flow of data.

Distance and the speed-of-light can affect performance for a bulk-transfer; often this can be mitigated by using larger windows. One common limit to the performance of a transport using window-based flow-control is:

```
Throughput <= WindowSize/RoundTripTime
```

As the sender can only have a window's-worth of data outstanding on the network at any one time, and the soonest the sender can receive a window update from the receiver is one RoundTripTime (RTT). TCP and SCTP are examples of such protocols.

Packet losses and their effects can be particularly bad for performance. This is especially true if the packet losses result in retransmission timeouts for the protocol(s) involved. By the time a retransmission timeout has happened, the flow or connection has sat idle for a considerable length of time.

On many platforms, some variant on the `netstat` command can be used to retrieve statistics about packet loss and retransmission. For example:

```
netstat -p tcp
```

will retrieve TCP statistics on the HP-UX Operating System. On other platforms, it may not be possible to retrieve statistics for a specific protocol and something like:

```
netstat -s
```

would be used instead.

Many times, such network statistics are keep since the time the stack started, and we are only really interested in statistics from when netperf was running. In such situations something along the lines of:

```
netstat -p tcp > before
netperf -t TCP_mumble...
netstat -p tcp > after
```

is indicated. The beforeafter utility can be used to subtract the statistics in 'before' from the statistics in 'after':

```
beforeafter before after > delta
```

and then one can look at the statistics in 'delta'. Beforeafter is distributed in source form so one can compile it on the platform(s) of interest.

If running a version 2.5.0 or later "omni" test under Linux one can include either or both of:

- LOCAL_TRANSPORT_RETRANS
- REMOTE_TRANSPORT_RETRANS

in the values provided via a test-specific '-o', '-O', or '-k' output selction option and netperf will report the retransmissions experienced on the data connection, as reported via a `getsockopt(TCP_INFO)` call. If confidence intervals have been requested via the global '-I' or '-i' options, the reported value(s) will be for the last iteration. If the test is over a protocol other than TCP, or on a platform other than Linux, the results are undefined.

While it was written with HP-UX's netstat in mind, the annotated netstat writeup may be helpful with other platforms as well.

5.2 Options common to TCP UDP and SCTP tests

Many "test-specific" options are actually common across the different tests. For those tests involving TCP, UDP and SCTP, whether using the BSD Sockets or the XTI interface those common options include:

`-h`
: Display the test-suite-specific usage string and exit. For a TCP_ or UDP_ test this will be the usage string from the source file nettest_bsd.c. For an XTI_ test, this will be the usage string from the source file nettest_xti.c. For an SCTP test, this will be the usage string from the source file nettest_sctp.c.

`-H <optionspec>`
: Normally, the remote hostname|IP and address family information is inherited from the settings for the control connection (eg global command-line '-H', '-4' and/or '-6' options). The test-specific '-H' will override those settings for the data (aka test) connection only. Settings for the control connection are left unchanged.

`-L <optionspec>`
: The test-specific '-L' option is identical to the test-specific '-H' option except it affects the local hostname|IP and address family information. As with its global command-line counterpart, this is generally only useful when measuring though those evil, end-to-end breaking things called firewalls.

`-m bytes` Set the size of the buffer passed-in to the "send" calls of a _STREAM test. Note that this may have only an indirect effect on the size of the packets sent over the network, and certain Layer 4 protocols do _not_ preserve or enforce message boundaries, so setting '`-m`' for the send size does not necessarily mean the receiver will receive that many bytes at any one time. By default the units are bytes, but suffix of "G," "M," or "K" will specify the units to be 2^30 (GB), 2^20 (MB) or 2^10 (KB) respectively. A suffix of "g," "m" or "k" will specify units of 10^9, 10^6 or 10^3 bytes respectively. For example:

```
-m 32K
```

will set the size to 32KB or 32768 bytes. [Default: the local send socket buffer size for the connection - either the system's default or the value set via the '`-s`' option.]

`-M bytes` Set the size of the buffer passed-in to the "recv" calls of a _STREAM test. This will be an upper bound on the number of bytes received per receive call. By default the units are bytes, but suffix of "G," "M," or "K" will specify the units to be 2^30 (GB), 2^20 (MB) or 2^10 (KB) respectively. A suffix of "g," "m" or "k" will specify units of 10^9, 10^6 or 10^3 bytes respectively. For example:

```
-M 32K
```

will set the size to 32KB or 32768 bytes. [Default: the remote receive socket buffer size for the data connection - either the system's default or the value set via the '`-S`' option.]

`-P <optionspec>`
Set the local and/or remote port numbers for the data connection.

`-s <sizespec>`
This option sets the local (netperf) send and receive socket buffer sizes for the data connection to the value(s) specified. Often, this will affect the advertised and/or effective TCP or other window, but on some platforms it may not. By default the units are bytes, but suffix of "G," "M," or "K" will specify the units to be 2^30 (GB), 2^20 (MB) or 2^10 (KB) respectively. A suffix of "g," "m" or "k" will specify units of 10^9, 10^6 or 10^3 bytes respectively. For example:

```
-s 128K
```

Will request the local send and receive socket buffer sizes to be 128KB or 131072 bytes.

While the historic expectation is that setting the socket buffer size has a direct effect on say the TCP window, today that may not hold true for all stacks. Further, while the historic expectation is that the value specified in a `setsockopt()` call will be the value returned via a `getsockopt()` call, at least one stack is known to deliberately ignore history. When running under Windows a value of 0 may be used which will be an indication to the stack the user wants to enable a form of copy avoidance. [Default: -1 - use the system's default socket buffer sizes]

`-S <sizespec>`
This option sets the remote (netserver) send and/or receive socket buffer sizes for the data connection to the value(s) specified. Often, this will affect the

advertised and/or effective TCP or other window, but on some platforms it may not. By default the units are bytes, but suffix of "G," "M," or "K" will specify the units to be 2^30 (GB), 2^20 (MB) or 2^10 (KB) respectively. A suffix of "g," "m" or "k" will specify units of 10^9, 10^6 or 10^3 bytes respectively. For example:

```
-S 128K
```

Will request the remote send and receive socket buffer sizes to be 128KB or 131072 bytes.

While the historic expectation is that setting the socket buffer size has a direct effect on say the TCP window, today that may not hold true for all stacks. Further, while the historic expectation is that the value specified in a `setsockopt()` call will be the value returned via a `getsockopt()` call, at least one stack is known to deliberately ignore history. When running under Windows a value of 0 may be used which will be an indication to the stack the user wants to enable a form of copy avoidance. [Default: -1 - use the system's default socket buffer sizes]

`-4` Set the local and remote address family for the data connection to AF_INET - ie use IPv4 addressing only. Just as with their global command-line counterparts the last of the '`-4`', '`-6`', '`-H`' or '`-L`' option wins for their respective address families.

`-6` This option is identical to its '`-4`' cousin, but requests IPv6 addresses for the local and remote ends of the data connection.

5.2.1 TCP_STREAM

The TCP_STREAM test is the default test in netperf. It is quite simple, transferring some quantity of data from the system running netperf to the system running netserver. While time spent establishing the connection is not included in the throughput calculation, time spent flushing the last of the data to the remote at the end of the test is. This is how netperf knows that all the data it sent was received by the remote. In addition to the Section 5.2 [Options common to TCP UDP and SCTP tests], page 21, the following test-specific options can be included to possibly alter the behavior of the test:

`-C` This option will set TCP_CORK mode on the data connection on those systems where TCP_CORK is defined (typically Linux). A full description of TCP_CORK is beyond the scope of this manual, but in a nutshell it forces sub-MSS sends to be buffered so every segment sent is Maximum Segment Size (MSS) unless the application performs an explicit flush operation or the connection is closed. At present netperf does not perform any explicit flush operations. Setting TCP_CORK may improve the bitrate of tests where the "send size" ('`-m`' option) is smaller than the MSS. It should also improve (make smaller) the service demand.

The Linux tcp(7) manpage states that TCP_CORK cannot be used in conjunction with TCP_NODELAY (set via the '`-d`' option), however netperf does not validate command-line options to enforce that.

`-D` This option will set TCP_NODELAY on the data connection on those systems where TCP_NODELAY is defined. This disables something known as the Nagle

Algorithm, which is intended to make the segments TCP sends as large as reasonably possible. Setting TCP_NODELAY for a TCP_STREAM test should either have no effect when the send size ('-m' option) is larger than the MSS or will decrease reported bitrate and increase service demand when the send size is smaller than the MSS. This stems from TCP_NODELAY causing each sub-MSS send to be its own TCP segment rather than being aggregated with other small sends. This means more trips up and down the protocol stack per KB of data transferred, which means greater CPU utilization.

If setting TCP_NODELAY with '-D' affects throughput and/or service demand for tests where the send size ('-m') is larger than the MSS it suggests the TCP/IP stack's implementation of the Nagle Algorithm _may_ be broken, perhaps interpreting the Nagle Algorithm on a segment by segment basis rather than the proper user send by user send basis. However, a better test of this can be achieved with the Section 6.2.1 [TCP_RR], page 31 test.

Here is an example of a basic TCP_STREAM test, in this case from a Debian Linux (2.6 kernel) system to an HP-UX 11iv2 (HP-UX 11.23) system:

```
$ netperf -H lag
TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to lag.hpl.hp.com (15.4.89.214)
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 32768  16384  16384    10.00      80.42
```

We see that the default receive socket buffer size for the receiver (lag - HP-UX 11.23) is 32768 bytes, and the default socket send buffer size for the sender (Debian 2.6 kernel) is 16384 bytes, however Linux does "auto tuning" of socket buffer and TCP window sizes, which means the send socket buffer size may be different at the end of the test than it was at the beginning. This is addressed in the Chapter 9 [The Omni Tests], page 47 added in version 2.5.0 and Section 9.3 [Omni Output Selection], page 50. Throughput is expressed as 10^6 (aka Mega) bits per second, and the test ran for 10 seconds. IPv4 addresses (AF_INET) were used.

5.2.2 TCP_MAERTS

A TCP_MAERTS (MAERTS is STREAM backwards) test is "just like" a Section 5.2.1 [TCP_STREAM], page 23 test except the data flows from the netserver to the netperf. The global command-line '-F' option is ignored for this test type. The test-specific command-line '-C' option is ignored for this test type.

Here is an example of a TCP_MAERTS test between the same two systems as in the example for the Section 5.2.1 [TCP_STREAM], page 23 test. This time we request larger socket buffers with '-s' and '-S' options:

```
$ netperf -H lag -t TCP_MAERTS -- -s 128K -S 128K
TCP MAERTS TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to lag.hpl.hp.com (15.4.89.214)
Recv   Send    Send
Socket Socket  Message  Elapsed
```

```
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

221184 131072 131072    10.03      81.14
```

Where we see that Linux, unlike HP-UX, may not return the same value in a `getsockopt()` as was requested in the prior `setsockopt()`.

This test is included more for benchmarking convenience than anything else.

5.2.3 TCP_SENDFILE

The TCP_SENDFILE test is "just like" a Section 5.2.1 [TCP_STREAM], page 23 test except netperf the platform's `sendfile()` call instead of calling `send()`. Often this results in a *zero-copy* operation where data is sent directly from the filesystem buffer cache. This _should_ result in lower CPU utilization and possibly higher throughput. If it does not, then you may want to contact your vendor(s) because they have a problem on their hands.

Zero-copy mechanisms may also alter the characteristics (size and number of buffers per) of packets passed to the NIC. In many stacks, when a copy is performed, the stack can "reserve" space at the beginning of the destination buffer for things like TCP, IP and Link headers. This then has the packet contained in a single buffer which can be easier to DMA to the NIC. When no copy is performed, there is no opportunity to reserve space for headers and so a packet will be contained in two or more buffers.

As of some time before version 2.5.0, the Section 4.2 [Global Options], page 11 is no longer required for this test. If it is not specified, netperf will create a temporary file, which it will delete at the end of the test. If the '`-F`' option is specified it must reference a file of at least the size of the send ring (See Section 4.2 [Global Options], page 11.) multiplied by the send size (See Section 5.2 [Options common to TCP UDP and SCTP tests], page 21.). All other TCP-specific options remain available and optional.

In this first example:

```
$ netperf -H lag -F ../src/netperf -t TCP_SENDFILE -- -s 128K -S 128K
TCP SENDFILE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to lag.hpl.hp.com (15.4.89.214
alloc_sendfile_buf_ring: specified file too small.
file must be larger than send_width * send_size
```

we see what happens when the file is too small. Here:

```
$ netperf -H lag -F /boot/vmlinuz-2.6.8-1-686 -t TCP_SENDFILE -- -s 128K -S 128K
TCP SENDFILE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to lag.hpl.hp.com (15.4.89.214
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

131072 221184 221184    10.02      81.83
```

we resolve that issue by selecting a larger file.

5.2.4 UDP_STREAM

A UDP_STREAM test is similar to a Section 5.2.1 [TCP_STREAM], page 23 test except UDP is used as the transport rather than TCP.

A UDP_STREAM test has no end-to-end flow control - UDP provides none and neither does netperf. However, if you wish, you can configure netperf with `--enable-intervals=yes` to enable the global command-line '`-b`' and '`-w`' options to pace bursts of traffic onto the network.

This has a number of implications.

The biggest of these implications is the data which is sent might not be received by the remote. For this reason, the output of a UDP_STREAM test shows both the sending and receiving throughput. On some platforms, it may be possible for the sending throughput to be reported as a value greater than the maximum rate of the link. This is common when the CPU(s) are faster than the network and there is no *intra-stack* flow-control.

Here is an example of a UDP_STREAM test between two systems connected by a 10 Gigabit Ethernet link:

```
$ netperf -t UDP_STREAM -H 192.168.2.125 -- -m 32768
UDP UNIDIRECTIONAL SEND TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 192.168.2.125 (1
Socket  Message  Elapsed      Messages
Size    Size     Time         Okay Errors   Throughput
bytes   bytes    secs            #      #   10^6bits/sec

124928   32768   10.00      105672      0    2770.20
135168           10.00      104844           2748.50
```

The first line of numbers are statistics from the sending (netperf) side. The second line of numbers are from the receiving (netserver) side. In this case, 105672 - 104844 or 828 messages did not make it all the way to the remote netserver process.

If the value of the '`-m`' option is larger than the local send socket buffer size ('`-s`' option) netperf will likely abort with an error message about how the send call failed:

```
netperf -t UDP_STREAM -H 192.168.2.125
UDP UNIDIRECTIONAL SEND TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 192.168.2.125 (1
udp_send: data send error: Message too long
```

If the value of the '`-m`' option is larger than the remote socket receive buffer, the reported receive throughput will likely be zero as the remote UDP will discard the messages as being too large to fit into the socket buffer.

```
$ netperf -t UDP_STREAM -H 192.168.2.125 -- -m 65000 -S 32768
UDP UNIDIRECTIONAL SEND TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 192.168.2.125 (1
Socket  Message  Elapsed      Messages
Size    Size     Time         Okay Errors   Throughput
bytes   bytes    secs            #      #   10^6bits/sec

124928   65000   10.00       53595      0    2786.99
 65536           10.00           0             0.00
```

The example above was between a pair of systems running a "Linux" kernel. Notice that the remote Linux system returned a value larger than that passed-in to the '`-S`' option. In fact, this value was larger than the message size set with the '`-m`' option. That the remote socket buffer size is reported as 65536 bytes would suggest to any sane person that a message

of 65000 bytes would fit, but the socket isn't _really_ 65536 bytes, even though Linux is telling us so. Go figure.

5.2.5 XTI_TCP_STREAM

An XTI_TCP_STREAM test is simply a Section 5.2.1 [TCP_STREAM], page 23 test using the XTI rather than BSD Sockets interface. The test-specific '`-X <devspec>`' option can be used to specify the name of the local and/or remote XTI device files, which is required by the `t_open()` call made by netperf XTI tests.

The XTI_TCP_STREAM test is only present if netperf was configured with `--enable-xti=yes`. The remote netserver must have also been configured with `--enable-xti=yes`.

5.2.6 XTI_UDP_STREAM

An XTI_UDP_STREAM test is simply a Section 5.2.4 [UDP_STREAM], page 25 test using the XTI rather than BSD Sockets Interface. The test-specific '`-X <devspec>`' option can be used to specify the name of the local and/or remote XTI device files, which is required by the `t_open()` call made by netperf XTI tests.

The XTI_UDP_STREAM test is only present if netperf was configured with `--enable-xti=yes`. The remote netserver must have also been configured with `--enable-xti=yes`.

5.2.7 SCTP_STREAM

An SCTP_STREAM test is essentially a Section 5.2.1 [TCP_STREAM], page 23 test using the SCTP rather than TCP. The '`-D`' option will set SCTP_NODELAY, which is much like the TCP_NODELAY option for TCP. The '`-C`' option is not applicable to an SCTP test as there is no corresponding SCTP_CORK option. The author is still figuring-out what the test-specific '`-N`' option does :)

The SCTP_STREAM test is only present if netperf was configured with `--enable-sctp=yes`. The remote netserver must have also been configured with `--enable-sctp=yes`.

5.2.8 DLCO_STREAM

A DLPI Connection Oriented Stream (DLCO_STREAM) test is very similar in concept to a Section 5.2.1 [TCP_STREAM], page 23 test. Both use reliable, connection-oriented protocols. The DLPI test differs from the TCP test in that its protocol operates only at the link-level and does not include TCP-style segmentation and reassembly. This last difference means that the value passed-in with the '`-m`' option must be less than the interface MTU. Otherwise, the '`-m`' and '`-M`' options are just like their TCP/UDP/SCTP counterparts.

Other DLPI-specific options include:

`-D <devspec>`
: This option is used to provide the fully-qualified names for the local and/or remote DLPI device files. The syntax is otherwise identical to that of a *sizespec*.

`-p <ppaspec>`
: This option is used to specify the local and/or remote DLPI PPA(s). The PPA is used to identify the interface over which traffic is to be sent/received. The syntax of a *ppaspec* is otherwise the same as a *sizespec*.

`-s sap`
: This option specifies the 802.2 SAP for the test. A SAP is somewhat like either the port field of a TCP or UDP header or the protocol field of an IP header. The

specified SAP should not conflict with any other active SAPs on the specified PPA's ('-p' option).

`-w <sizespec>`
: This option specifies the local send and receive window sizes in units of frames on those platforms which support setting such things.

`-W <sizespec>`
: This option specifies the remote send and receive window sizes in units of frames on those platforms which support setting such things.

The DLCO_STREAM test is only present if netperf was configured with `--enable-dlpi=yes`. The remote netserver must have also been configured with `--enable-dlpi=yes`.

5.2.9 DLCL_STREAM

A DLPI ConnectionLess Stream (DLCL_STREAM) test is analogous to a Section 5.2.4 [UDP_STREAM], page 25 test in that both make use of unreliable/best-effort, connectionless transports. The DLCL_STREAM test differs from the Section 5.2.4 [UDP_STREAM], page 25 test in that the message size ('-m' option) must always be less than the link MTU as there is no IP-like fragmentation and reassembly available and netperf does not presume to provide one.

The test-specific command-line options for a DLCL_STREAM test are the same as those for a Section 5.2.8 [DLCO_STREAM], page 27 test.

The DLCL_STREAM test is only present if netperf was configured with `--enable-dlpi=yes`. The remote netserver must have also been configured with `--enable-dlpi=yes`.

5.2.10 STREAM_STREAM

A Unix Domain Stream Socket Stream test (STREAM_STREAM) is similar in concept to a Section 5.2.1 [TCP_STREAM], page 23 test, but using Unix Domain sockets. It is, naturally, limited to intra-machine traffic. A STREAM_STREAM test shares the '-m', '-M', '-s' and '-S' options of the other _STREAM tests. In a STREAM_STREAM test the '-p' option sets the directory in which the pipes will be created rather than setting a port number. The default is to create the pipes in the system default for the `tempnam()` call.

The STREAM_STREAM test is only present if netperf was configured with `--enable-unixdomain=yes`. The remote netserver must have also been configured with `--enable-unixdomain=yes`.

5.2.11 DG_STREAM

A Unix Domain Datagram Socket Stream test (SG_STREAM) is very much like a Section 5.2.1 [TCP_STREAM], page 23 test except that message boundaries are preserved. In this way, it may also be considered similar to certain flavors of SCTP test which can also preserve message boundaries.

All the options of a Section 5.2.10 [STREAM_STREAM], page 28 test are applicable to a DG_STREAM test.

The DG_STREAM test is only present if netperf was configured with `--enable-unixdomain=yes`. The remote netserver must have also been configured with `--enable-unixdomain=yes`.

6 Using Netperf to Measure Request/Response

Request/response performance is often overlooked, yet it is just as important as bulk-transfer performance. While things like larger socket buffers and TCP windows, and stateless offloads like TSO and LRO can cover a multitude of latency and even path-length sins, those sins cannot easily hide from a request/response test. The convention for a request/response test is to have a _RR suffix. There are however a few "request/response" tests that have other suffixes.

A request/response test, particularly synchronous, one transaction at a time test such as those found by default in netperf, is particularly sensitive to the path-length of the networking stack. An _RR test can also uncover those platforms where the NICs are strapped by default with overbearing interrupt avoidance settings in an attempt to increase the bulk-transfer performance (or rather, decrease the CPU utilization of a bulk-transfer test). This sensitivity is most acute for small request and response sizes, such as the single-byte default for a netperf _RR test.

While a bulk-transfer test reports its results in units of bits or bytes transferred per second, by default a mumble_RR test reports transactions per second where a transaction is defined as the completed exchange of a request and a response. One can invert the transaction rate to arrive at the average round-trip latency. If one is confident about the symmetry of the connection, the average one-way latency can be taken as one-half the average round-trip latency. As of version 2.5.0 (actually slightly before) netperf still does not do the latter, but will do the former if one sets the verbosity to 2 for a classic netperf test, or includes the appropriate Section 9.3.1 [Omni Output Selectors], page 51 in an Chapter 9 [The Omni Tests], page 47. It will also allow the user to switch the throughput units from transactions per second to bits or bytes per second with the global '`-f`' option.

6.1 Issues in Request/Response

Most if not all the Section 5.1 [Issues in Bulk Transfer], page 20 apply to request/response. The issue of round-trip latency is even more important as netperf generally only has one transaction outstanding at a time.

A single instance of a one transaction outstanding _RR test should _never_ completely saturate the CPU of a system. If testing between otherwise evenly matched systems, the symmetric nature of a _RR test with equal request and response sizes should result in equal CPU loading on both systems. However, this may not hold true on MP systems, particularly if one CPU binds the netperf and netserver differently via the global '`-T`' option.

For smaller request and response sizes packet loss is a bigger issue as there is no opportunity for a *fast retransmit* or retransmission prior to a retransmission timer expiring.

Virtualization may considerably increase the effective path length of a networking stack. While this may not preclude achieving link-rate on a comparatively slow link (eg 1 Gigabit Ethernet) on a _STREAM test, it can show-up as measurably fewer transactions per second on an _RR test. However, this may still be masked by interrupt coalescing in the NIC/driver.

Certain NICs have ways to minimize the number of interrupts sent to the host. If these are strapped badly they can significantly reduce the performance of something like a single-byte request/response test. Such setups are distinguished by seriously low reported CPU utilization and what seems like a low (even if in the thousands) transaction per second

rate. Also, if you run such an OS/driver combination on faster or slower hardware and do not see a corresponding change in the transaction rate, chances are good that the driver is strapping the NIC with aggressive interrupt avoidance settings. Good for bulk throughput, but bad for latency.

Some drivers may try to automagically adjust the interrupt avoidance settings. If they are not terribly good at it, you will see considerable run-to-run variation in reported transaction rates. Particularly if you "mix-up" _STREAM and _RR tests.

6.2 Options Common to TCP UDP and SCTP _RR tests

Many "test-specific" options are actually common across the different tests. For those tests involving TCP, UDP and SCTP, whether using the BSD Sockets or the XTI interface those common options include:

`-h` Display the test-suite-specific usage string and exit. For a TCP_ or UDP_ test this will be the usage string from the source file '`nettest_bsd.c`'. For an XTI_ test, this will be the usage string from the source file '`src/nettest_xti.c`'. For an SCTP test, this will be the usage string from the source file '`src/nettest_sctp.c`'.

`-H <optionspec>`
Normally, the remote hostname|IP and address family information is inherited from the settings for the control connection (eg global command-line '`-H`', '`-4`' and/or '`-6`' options. The test-specific '`-H`' will override those settings for the data (aka test) connection only. Settings for the control connection are left unchanged. This might be used to cause the control and data connections to take different paths through the network.

`-L <optionspec>`
The test-specific '`-L`' option is identical to the test-specific '`-H`' option except it affects the local hostname|IP and address family information. As with its global command-line counterpart, this is generally only useful when measuring though those evil, end-to-end breaking things called firewalls.

`-P <optionspec>`
Set the local and/or remote port numbers for the data connection.

`-r <sizespec>`
This option sets the request (first value) and/or response (second value) sizes for an _RR test. By default the units are bytes, but a suffix of "G," "M," or "K" will specify the units to be 2^30 (GB), 2^20 (MB) or 2^10 (KB) respectively. A suffix of "g," "m" or "k" will specify units of 10^9, 10^6 or 10^3 bytes respectively. For example:

```
-r 128,16K
```

Will set the request size to 128 bytes and the response size to 16 KB or 16384 bytes. [Default: 1 - a single-byte request and response]

`-s <sizespec>`
This option sets the local (netperf) send and receive socket buffer sizes for the data connection to the value(s) specified. Often, this will affect the advertised

and/or effective TCP or other window, but on some platforms it may not. By default the units are bytes, but a suffix of "G," "M," or "K" will specify the units to be 2^30 (GB), 2^20 (MB) or 2^10 (KB) respectively. A suffix of "g," "m" or "k" will specify units of 10^9, 10^6 or 10^3 bytes respectively. For example:

```
-s 128K
```

Will request the local send (netperf) and receive socket buffer sizes to be 128KB or 131072 bytes.

While the historic expectation is that setting the socket buffer size has a direct effect on say the TCP window, today that may not hold true for all stacks. When running under Windows a value of 0 may be used which will be an indication to the stack the user wants to enable a form of copy avoidance. [Default: -1 - use the system's default socket buffer sizes]

`-S <sizespec>`

This option sets the remote (netserver) send and/or receive socket buffer sizes for the data connection to the value(s) specified. Often, this will affect the advertised and/or effective TCP or other window, but on some platforms it may not. By default the units are bytes, but a suffix of "G," "M," or "K" will specify the units to be 2^30 (GB), 2^20 (MB) or 2^10 (KB) respectively. A suffix of "g," "m" or "k" will specify units of 10^9, 10^6 or 10^3 bytes respectively. For example:

```
-S 128K
```

Will request the remote (netserver) send and receive socket buffer sizes to be 128KB or 131072 bytes.

While the historic expectation is that setting the socket buffer size has a direct effect on say the TCP window, today that may not hold true for all stacks. When running under Windows a value of 0 may be used which will be an indication to the stack the user wants to enable a form of copy avoidance. [Default: -1 - use the system's default socket buffer sizes]

`-4` Set the local and remote address family for the data connection to AF_INET - ie use IPv4 addressing only. Just as with their global command-line counterparts the last of the '`-4`', '`-6`', '`-H`' or '`-L`' option wins for their respective address families.

`-6` This option is identical to its '`-4`' cousin, but requests IPv6 addresses for the local and remote ends of the data connection.

6.2.1 TCP_RR

A TCP_RR (TCP Request/Response) test is requested by passing a value of "TCP_RR" to the global '`-t`' command-line option. A TCP_RR test can be thought-of as a user-space to user-space `ping` with no think time - it is by default a synchronous, one transaction at a time, request/response test.

The transaction rate is the number of complete transactions exchanged divided by the length of time it took to perform those transactions.

If the two Systems Under Test are otherwise identical, a TCP_RR test with the same request and response size should be symmetric - it should not matter which way the test is run, and the CPU utilization measured should be virtually the same on each system. If not, it suggests that the CPU utilization mechanism being used may have some, well, issues measuring CPU utilization completely and accurately.

Time to establish the TCP connection is not counted in the result. If you want connection setup overheads included, you should consider the Section 6.2.2 [TCP_CC], page 32 or Section 6.2.3 [TCP_CRR], page 33 tests.

If specifying the '`-D`' option to set TCP_NODELAY and disable the Nagle Algorithm increases the transaction rate reported by a TCP_RR test, it implies the stack(s) over which the TCP_RR test is running have a broken implementation of the Nagle Algorithm. Likely as not they are interpreting Nagle on a segment by segment basis rather than a user send by user send basis. You should contact your stack vendor(s) to report the problem to them.

Here is an example of two systems running a basic TCP_RR test over a 10 Gigabit Ethernet link:

```
netperf -t TCP_RR -H 192.168.2.125
TCP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 192.168.2.125 (192.
Local /Remote
Socket Size   Request  Resp.   Elapsed  Trans.
Send   Recv   Size     Size    Time     Rate
bytes  Bytes  bytes    bytes   secs.    per sec

16384  87380  1        1       10.00    29150.15
16384  87380
```

In this example the request and response sizes were one byte, the socket buffers were left at their defaults, and the test ran for all of 10 seconds. The transaction per second rate was rather good for the time :)

6.2.2 TCP_CC

A TCP_CC (TCP Connect/Close) test is requested by passing a value of "TCP_CC" to the global '`-t`' option. A TCP_CC test simply measures how fast the pair of systems can open and close connections between one another in a synchronous (one at a time) manner. While this is considered an _RR test, no request or response is exchanged over the connection.

The issue of TIME_WAIT reuse is an important one for a TCP_CC test. Basically, TIME_WAIT reuse is when a pair of systems churn through connections fast enough that they wrap the 16-bit port number space in less time than the length of the TIME_WAIT state. While it is indeed theoretically possible to "reuse" a connection in TIME_WAIT, the conditions under which such reuse is possible are rather rare. An attempt to reuse a connection in TIME_WAIT can result in a non-trivial delay in connection establishment.

Basically, any time the connection churn rate approaches:

Sizeof(clientportspace) / Lengthof(TIME_WAIT)

there is the risk of TIME_WAIT reuse. To minimize the chances of this happening, netperf will by default select its own client port numbers from the range of 5000 to 65535. On systems with a 60 second TIME_WAIT state, this should allow roughly 1000 transactions

per second. The size of the client port space used by netperf can be controlled via the test-specific '`-p`' option, which takes a *sizespec* as a value setting the minimum (first value) and maximum (second value) port numbers used by netperf at the client end.

Since no requests or responses are exchanged during a TCP_CC test, only the '`-H`', '`-L`', '`-4`' and '`-6`' of the "common" test-specific options are likely to have an effect, if any, on the results. The '`-s`' and '`-S`' options _may_ have some effect if they alter the number and/or type of options carried in the TCP SYNchronize segments, such as Window Scaling or Timestamps. The '`-P`' and '`-r`' options are utterly ignored.

Since connection establishment and tear-down for TCP is not symmetric, a TCP_CC test is not symmetric in its loading of the two systems under test.

6.2.3 TCP_CRR

The TCP Connect/Request/Response (TCP_CRR) test is requested by passing a value of "TCP_CRR" to the global '`-t`' command-line option. A TCP_CRR test is like a merger of a Section 6.2.1 [TCP_RR], page 31 and Section 6.2.2 [TCP_CC], page 32 test which measures the performance of establishing a connection, exchanging a single request/response transaction, and tearing-down that connection. This is very much like what happens in an HTTP 1.0 or HTTP 1.1 connection when HTTP Keepalives are not used. In fact, the TCP_CRR test was added to netperf to simulate just that.

Since a request and response are exchanged the '`-r`', '`-s`' and '`-S`' options can have an effect on the performance.

The issue of TIME_WAIT reuse exists for the TCP_CRR test just as it does for the TCP_CC test. Similarly, since connection establishment and tear-down is not symmetric, a TCP_CRR test is not symmetric even when the request and response sizes are the same.

6.2.4 UDP_RR

A UDP Request/Response (UDP_RR) test is requested by passing a value of "UDP_RR" to a global '`-t`' option. It is very much the same as a TCP_RR test except UDP is used rather than TCP.

UDP does not provide for retransmission of lost UDP datagrams, and netperf does not add anything for that either. This means that if _any_ request or response is lost, the exchange of requests and responses will stop from that point until the test timer expires. Netperf will not really "know" this has happened - the only symptom will be a low transaction per second rate. If '`--enable-burst`' was included in the `configure` command and a test-specific '`-b`' option used, the UDP_RR test will "survive" the loss of requests and responses until the sum is one more than the value passed via the '`-b`' option. It will though almost certainly run more slowly.

The netperf side of a UDP_RR test will call `connect()` on its data socket and thenceforth use the `send()` and `recv()` socket calls. The netserver side of a UDP_RR test will not call `connect()` and will use `recvfrom()` and `sendto()` calls. This means that even if the request and response sizes are the same, a UDP_RR test is _not_ symmetric in its loading of the two systems under test.

Here is an example of a UDP_RR test between two otherwise identical two-CPU systems joined via a 1 Gigabit Ethernet network:

```
$ netperf -T 1 -H 192.168.1.213 -t UDP_RR -c -C
```

```
UDP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 192.168.1.213 (192.
Local /Remote
Socket Size   Request Resp.  Elapsed Trans.   CPU    CPU    S.dem   S.dem
Send   Recv   Size    Size   Time    Rate     local  remote local   remote
bytes  bytes  bytes   bytes  secs.   per sec  % I    % I    us/Tr   us/Tr

65535  65535  1       1      10.01   15262.48   13.90  16.11  18.221  21.116
65535  65535
```

This example includes the '`-c`' and '`-C`' options to enable CPU utilization reporting and shows the asymmetry in CPU loading. The '`-T`' option was used to make sure netperf and netserver ran on a given CPU and did not move around during the test.

6.2.5 XTI_TCP_RR

An XTI_TCP_RR test is essentially the same as a Section 6.2.1 [TCP_RR], page 31 test only using the XTI rather than BSD Sockets interface. It is requested by passing a value of "XTI_TCP_RR" to the '`-t`' global command-line option.

The test-specific options for an XTI_TCP_RR test are the same as those for a TCP_RR test with the addition of the '`-X <devspec>`' option to specify the names of the local and/or remote XTI device file(s).

6.2.6 XTI_TCP_CC

An XTI_TCP_CC test is essentially the same as a Section 6.2.2 [TCP_CC], page 32 test, only using the XTI rather than BSD Sockets interface.

The test-specific options for an XTI_TCP_CC test are the same as those for a TCP_CC test with the addition of the '`-X <devspec>`' option to specify the names of the local and/or remote XTI device file(s).

6.2.7 XTI_TCP_CRR

The XTI_TCP_CRR test is essentially the same as a Section 6.2.3 [TCP_CRR], page 33 test, only using the XTI rather than BSD Sockets interface.

The test-specific options for an XTI_TCP_CRR test are the same as those for a TCP_RR test with the addition of the '`-X <devspec>`' option to specify the names of the local and/or remote XTI device file(s).

6.2.8 XTI_UDP_RR

An XTI_UDP_RR test is essentially the same as a UDP_RR test only using the XTI rather than BSD Sockets interface. It is requested by passing a value of "XTI_UDP_RR" to the '`-t`' global command-line option.

The test-specific options for an XTI_UDP_RR test are the same as those for a UDP_RR test with the addition of the '`-X <devspec>`' option to specify the name of the local and/or remote XTI device file(s).

6.2.9 DLCL_RR

6.2.10 DLCO_RR

6.2.11 SCTP_RR

7 Using Netperf to Measure Aggregate Performance

Ultimately, Chapter 13 [Netperf4], page 69 will be the preferred benchmark to use when one wants to measure aggregate performance because netperf has no support for explicit synchronization of concurrent tests. Until netperf4 is ready for prime time, one can make use of the heuristics and procedures mentioned here for the 85% solution.

There are a few ways to measure aggregate performance with netperf. The first is to run multiple, concurrent netperf tests and can be applied to any of the netperf tests. The second is to configure netperf with `--enable-burst` and is applicable to the TCP_RR test. The third is a variation on the first.

7.1 Running Concurrent Netperf Tests

Chapter 13 [Netperf4], page 69 is the preferred benchmark to use when one wants to measure aggregate performance because netperf has no support for explicit synchronization of concurrent tests. This leaves netperf2 results vulnerable to *skew* errors.

However, since there are times when netperf4 is unavailable it may be necessary to run netperf. The skew error can be minimized by making use of the confidence interval functionality. Then one simply launches multiple tests from the shell using a `for` loop or the like:

```
for i in 1 2 3 4
do
netperf -t TCP_STREAM -H tardy.cup.hp.com -i 10 -P 0 &
done
```

which will run four, concurrent Section 5.2.1 [TCP_STREAM], page 23 tests from the system on which it is executed to tardy.cup.hp.com. Each concurrent netperf will iterate 10 times thanks to the '`-i`' option and will omit the test banners (option '`-P`') for brevity. The output looks something like this:

```
 87380  16384  16384    10.03     235.15
 87380  16384  16384    10.03     235.09
 87380  16384  16384    10.03     235.38
 87380  16384  16384    10.03     233.96
```

We can take the sum of the results and be reasonably confident that the aggregate performance was 940 Mbits/s. This method does not need to be limited to one system speaking to one other system. It can be extended to one system talking to N other systems. It could be as simple as:

```
for host in 'foo bar baz bing'
do
netperf -t TCP_STREAM -H $hosts -i 10 -P 0 &
done
```

A more complicated/sophisticated example can be found in '`doc/examples/runemomniagg2.sh`' where.

If you see warnings about netperf not achieving the confidence intervals, the best thing to do is to increase the number of iterations with '`-i`' and/or increase the run length of each iteration with '`-l`'.

You can also enable local ('`-c`') and/or remote ('`-C`') CPU utilization:

```
for i in 1 2 3 4
do
netperf -t TCP_STREAM -H tardy.cup.hp.com -i 10 -P 0 -c -C &
done

87380  16384  16384    10.03       235.47   3.67     5.09     10.226  14.180
87380  16384  16384    10.03       234.73   3.67     5.09     10.260  14.225
87380  16384  16384    10.03       234.64   3.67     5.10     10.263  14.231
87380  16384  16384    10.03       234.87   3.67     5.09     10.253  14.215
```

If the CPU utilizations reported for the same system are the same or very very close you can be reasonably confident that skew error is minimized. Presumably one could then omit '`-i`' but that is not advised, particularly when/if the CPU utilization approaches 100 percent. In the example above we see that the CPU utilization on the local system remains the same for all four tests, and is only off by 0.01 out of 5.09 on the remote system. As the number of CPUs in the system increases, and so too the odds of saturating a single CPU, the accuracy of similar CPU utilization implying little skew error is diminished. This is also the case for those increasingly rare single CPU systems if the utilization is reported as 100% or very close to it.

> **NOTE: It is very important to remember that netperf is calculating system-wide CPU utilization. When calculating the service demand (those last two columns in the output above) each netperf assumes it is the only thing running on the system. This means that for concurrent tests the service demands reported by netperf will be wrong. One has to compute service demands for concurrent tests by hand.**

If you wish you can add a unique, global '`-B`' option to each command line to append the given string to the output:

```
for i in 1 2 3 4
do
netperf -t TCP_STREAM -H tardy.cup.hp.com -B "this is test $i" -i 10 -P 0 &
done

87380  16384  16384    10.03     234.90   this is test 4
87380  16384  16384    10.03     234.41   this is test 2
87380  16384  16384    10.03     235.26   this is test 1
87380  16384  16384    10.03     235.09   this is test 3
```

You will notice that the tests completed in an order other than they were started from the shell. This underscores why there is a threat of skew error and why netperf4 will eventually be the preferred tool for aggregate tests. Even if you see the Netperf Contributing Editor acting to the contrary!-)

7.1.1 Issues in Running Concurrent Tests

In addition to the aforementioned issue of skew error, there can be other issues to consider when running concurrent netperf tests.

For example, when running concurrent tests over multiple interfaces, one is not always assured that the traffic one thinks went over a given interface actually did so. In particular,

the Linux networking stack takes a particularly strong stance on its following the so called '`weak end system model`'. As such, it is willing to answer ARP requests for any of its local IP addresses on any of its interfaces. If multiple interfaces are connected to the same broadcast domain, then even if they are configured into separate IP subnets there is no a priori way of knowing which interface was actually used for which connection(s). This can be addressed by setting the '`arp_ignore`' sysctl before configuring interfaces.

As it is quite important, we will repeat that it is very important to remember that each concurrent netperf instance is calculating system-wide CPU utilization. When calculating the service demand each netperf assumes it is the only thing running on the system. This means that for concurrent tests the service demands reported by netperf **will be wrong**. One has to compute service demands for concurrent tests by hand

Running concurrent tests can also become difficult when there is no one "central" node. Running tests between pairs of systems may be more difficult, calling for remote shell commands in the for loop rather than netperf commands. This introduces more skew error, which the confidence intervals may not be able to sufficiently mitigate. One possibility is to actually run three consecutive netperf tests on each node - the first being a warm-up, the last being a cool-down. The idea then is to ensure that the time it takes to get all the netperfs started is less than the length of the first netperf command in the sequence of three. Similarly, it assumes that all "middle" netperfs will complete before the first of the "last" netperfs complete.

7.2 Using - -enable-burst

Starting in version 2.5.0 `--enable-burst=yes` is the default, which means one no longer must:

```
configure --enable-burst
```

To have burst-mode functionality present in netperf. This enables a test-specific '`-b num`' option in Section 6.2.1 [TCP_RR], page 31, Section 6.2.4 [UDP_RR], page 33 and Chapter 9 [The Omni Tests], page 47 tests.

Normally, netperf will attempt to ramp-up the number of outstanding requests to '`num`' plus one transactions in flight at one time. The ramp-up is to avoid transactions being smashed together into a smaller number of segments when the transport's congestion window (if any) is smaller at the time than what netperf wants to have outstanding at one time. If, however, the user specifies a negative value for '`num`' this ramp-up is bypassed and the burst of sends is made without consideration of transport congestion window.

This burst-mode is used as an alternative to or even in conjunction with multiple-concurrent _RR tests and as a way to implement a single-connection, bidirectional bulk-transfer test. When run with just a single instance of netperf, increasing the burst size can determine the maximum number of transactions per second which can be serviced by a single process:

```
for b in 0 1 2 4 8 16 32
do
 netperf -v 0 -t TCP_RR -B "-b $b" -H hpcpc108 -P 0 -- -b $b
done

9457.59 -b 0
```

```
9975.37 -b 1
10000.61 -b 2
20084.47 -b 4
29965.31 -b 8
71929.27 -b 16
109718.17 -b 32
```

The global '-v' and '-P' options were used to minimize the output to the single figure of merit which in this case the transaction rate. The global -B option was used to more clearly label the output, and the test-specific '-b' option enabled by --enable-burst increase the number of transactions in flight at one time.

Now, since the test-specific '-D' option was not specified to set TCP_NODELAY, the stack was free to "bundle" requests and/or responses into TCP segments as it saw fit, and since the default request and response size is one byte, there could have been some considerable bundling even in the absence of transport congestion window issues. If one wants to try to achieve a closer to one-to-one correspondence between a request and response and a TCP segment, add the test-specific '-D' option:

```
for b in 0 1 2 4 8 16 32
do
 netperf -v 0 -t TCP_RR -B "-b $b -D" -H hpcpc108 -P 0 -- -b $b -D
done

 8695.12 -b 0 -D
 19966.48 -b 1 -D
 20691.07 -b 2 -D
 49893.58 -b 4 -D
 62057.31 -b 8 -D
 108416.88 -b 16 -D
 114411.66 -b 32 -D
```

You can see that this has a rather large effect on the reported transaction rate. In this particular instance, the author believes it relates to interactions between the test and interrupt coalescing settings in the driver for the NICs used.

NOTE: Even if you set the '-D' option that is still not a guarantee that each transaction is in its own TCP segments. You should get into the habit of verifying the relationship between the transaction rate and the packet rate via other means.

You can also combine --enable-burst functionality with concurrent netperf tests. This would then be an "aggregate of aggregates" if you like:

```
for i in 1 2 3 4
do
 netperf -H hpcpc108 -v 0 -P 0 -i 10 -B "aggregate $i -b 8 -D" -t TCP_RR -- -b 8 -D &
done

 46668.38 aggregate 4 -b 8 -D
 44890.64 aggregate 2 -b 8 -D
```

```
 45702.04 aggregate 1 -b 8 -D
 46352.48 aggregate 3 -b 8 -D
```

Since each netperf did hit the confidence intervals, we can be reasonably certain that the aggregate transaction per second rate was the sum of all four concurrent tests, or something just shy of 184,000 transactions per second. To get some idea if that was also the packet per second rate, we could bracket that `for` loop with something to gather statistics and run the results through beforeafter:

```
/usr/sbin/ethtool -S eth2 > before
for i in 1 2 3 4
do
 netperf -H 192.168.2.108 -l 60 -v 0 -P 0 -B "aggregate $i -b 8 -D" -t TCP_RR -- -b 8
done
wait
/usr/sbin/ethtool -S eth2 > after

 52312.62 aggregate 2 -b 8 -D
 50105.65 aggregate 4 -b 8 -D
 50890.82 aggregate 1 -b 8 -D
 50869.20 aggregate 3 -b 8 -D

beforeafter before after > delta

grep packets delta
     rx_packets: 12251544
     tx_packets: 12251550
```

This example uses `ethtool` because the system being used is running Linux. Other platforms have other tools - for example HP-UX has lanadmin:

```
lanadmin -g mibstats <ppa>
```

and of course one could instead use `netstat`.

The `wait` is important because we are launching concurrent netperfs in the background. Without it, the second ethtool command would be run before the tests finished and perhaps even before the last of them got started!

The sum of the reported transaction rates is 204178 over 60 seconds, which is a total of 12250680 transactions. Each transaction is the exchange of a request and a response, so we multiply that by 2 to arrive at 24501360.

The sum of the ethtool stats is 24503094 packets which matches what netperf was reporting very well.

Had the request or response size differed, we would need to know how it compared with the *MSS* for the connection.

Just for grins, here is the exercise repeated, using `netstat` instead of `ethtool`

```
netstat -s -t > before
for i in 1 2 3 4
```

```
do
 netperf -l 60 -H 192.168.2.108 -v 0 -P 0 -B "aggregate $i -b 8 -D" -t TCP_RR -- -b 8
wait
netstat -s -t > after

 51305.88 aggregate 4 -b 8 -D
 51847.73 aggregate 2 -b 8 -D
 50648.19 aggregate 3 -b 8 -D
 53605.86 aggregate 1 -b 8 -D

beforeafter before after > delta

grep segments delta
    12445708 segments received
    12445730 segments send out
    1 segments retransmited
    0 bad segments received.
```

The sums are left as an exercise to the reader :)

Things become considerably more complicated if there are non-trvial packet losses and/or retransmissions.

Of course all this checking is unnecessary if the test is a UDP_RR test because UDP "never" aggregates multiple sends into the same UDP datagram, and there are no ACKnowledgements in UDP. The loss of a single request or response will not bring a "burst" UDP_RR test to a screeching halt, but it will reduce the number of transactions outstanding at any one time. A "burst" UDP_RR test **will** come to a halt if the sum of the lost requests and responses reaches the value specified in the test-specific '`-b`' option.

7.3 Using - -enable-demo

One can

```
configure --enable-demo
```

and compile netperf to enable netperf to emit "interim results" at semi-regular intervals. This enables a global `-D` option which takes a reporting interval as an argument. With that specified, the output of netperf will then look something like

```
$ src/netperf -D 1.25
MIGRATED TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to localhost.localdomai
Interim result: 25425.52 10^6bits/s over 1.25 seconds ending at 1327962078.405
Interim result: 25486.82 10^6bits/s over 1.25 seconds ending at 1327962079.655
Interim result: 25474.96 10^6bits/s over 1.25 seconds ending at 1327962080.905
Interim result: 25523.49 10^6bits/s over 1.25 seconds ending at 1327962082.155
Interim result: 25053.57 10^6bits/s over 1.27 seconds ending at 1327962083.429
Interim result: 25349.64 10^6bits/s over 1.25 seconds ending at 1327962084.679
Interim result: 25292.84 10^6bits/s over 1.25 seconds ending at 1327962085.932
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
```

```
bytes  bytes   bytes    secs.    10^6bits/sec

 87380  16384  16384    10.00    25375.66
```

The units of the "Interim result" lines will follow the units selected via the global `-f` option. If the test-specific `-o` option is specified on the command line, the format will be CSV:

```
...
2978.81,MBytes/s,1.25,1327962298.035
...
```

If the test-specific `-k` option is used the format will be keyval with each keyval being given an index:

```
...
NETPERF_INTERIM_RESULT[2]=25.00
NETPERF_UNITS[2]=10^9bits/s
NETPERF_INTERVAL[2]=1.25
NETPERF_ENDING[2]=1327962357.249
...
```

The expectation is it may be easier to utilize the keyvals if they have indices.

But how does this help with aggregate tests? Well, what one can do is start the netperfs via a script, giving each a Very Long (tm) run time. Direct the output to a file per instance. Then, once all the netperfs have been started, take a timestamp and wait for some desired test interval. Once that interval expires take another timestamp and then start terminating the netperfs by sending them a SIGALRM signal via the likes of the `kill` or `pkill` command. The netperfs will terminate and emit the rest of the "usual" output, and you can then bring the files to a central location for post processing to find the aggregate performance over the "test interval."

This method has the advantage that it does not require advance knowledge of how long it takes to get netperf tests started and/or stopped. It does though require sufficiently synchronized clocks on all the test systems.

While calls to get the current time can be inexpensive, that neither has been nor is universally true. For that reason netperf tries to minimize the number of such "timestamping" calls (eg `gettimeofday`) calls it makes when in demo mode. Rather than take a timestamp after each `send` or `recv` call completes netperf tries to guess how many units of work will be performed over the desired interval. Only once that many units of work have been completed will netperf check the time. If the reporting interval has passed, netperf will emit an "interim result." If the interval has not passed, netperf will update its estimate for units and continue.

After a bit of thought one can see that if things "speed-up" netperf will still honor the interval. However, if things "slow-down" netperf may be late with an "interim result." Here is an example of both of those happening during a test - with the interval being honored while throughput increases, and then about half-way through when another netperf (not shown) is started we see things slowing down and netperf not hitting the interval as desired.

```
$ src/netperf -D 2 -H tardy.hpl.hp.com -l 20
MIGRATED TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to tardy.hpl.hp.com () p
Interim result:   36.46 10^6bits/s over 2.01 seconds ending at 1327963880.565
```

```
Interim result:   59.19 10^6bits/s over 2.00 seconds ending at 1327963882.569
Interim result:   73.39 10^6bits/s over 2.01 seconds ending at 1327963884.576
Interim result:   84.01 10^6bits/s over 2.03 seconds ending at 1327963886.603
Interim result:   75.63 10^6bits/s over 2.21 seconds ending at 1327963888.814
Interim result:   55.52 10^6bits/s over 2.72 seconds ending at 1327963891.538
Interim result:   70.94 10^6bits/s over 2.11 seconds ending at 1327963893.650
Interim result:   80.66 10^6bits/s over 2.13 seconds ending at 1327963895.777
Interim result:   86.42 10^6bits/s over 2.12 seconds ending at 1327963897.901
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 87380  16384  16384    20.34      68.87
```

So long as your post-processing mechanism can account for that, there should be no problem. As time passes there may be changes to try to improve the netperf's honoring the interval but one should not ass-u-me it will always do so. One should not assume the precision will remain fixed - future versions may change it - perhaps going beyond tenths of seconds in reporting the interval length etc.

8 Using Netperf to Measure Bidirectional Transfer

There are two ways to use netperf to measure the performance of bidirectional transfer. The first is to run concurrent netperf tests from the command line. The second is to configure netperf with `--enable-burst` and use a single instance of the Section 6.2.1 [TCP_RR], page 31 test.

While neither method is more "correct" than the other, each is doing so in different ways, and that has possible implications. For instance, using the concurrent netperf test mechanism means that multiple TCP connections and multiple processes are involved, whereas using the single instance of TCP_RR there is only one TCP connection and one process on each end. They may behave differently, especially on an MP system.

8.1 Bidirectional Transfer with Concurrent Tests

If we had two hosts Fred and Ethel, we could simply run a netperf Section 5.2.1 [TCP_STREAM], page 23 test on Fred pointing at Ethel, and a concurrent netperf TCP_STREAM test on Ethel pointing at Fred, but since there are no mechanisms to synchronize netperf tests and we would be starting tests from two different systems, there is a considerable risk of skew error.

Far better would be to run simultaneous TCP_STREAM and Section 5.2.2 [TCP_MAERTS], page 24 tests from just **one** system, using the concepts and procedures outlined in Section 7.1 [Running Concurrent Netperf Tests], page 36. Here then is an example:

```
for i in 1
do
 netperf -H 192.168.2.108 -t TCP_STREAM -B "outbound" -i 10 -P 0 -v 0 \
   -- -s 256K -S 256K &
 netperf -H 192.168.2.108 -t TCP_MAERTS -B "inbound"  -i 10 -P 0 -v 0 \
   -- -s 256K -S 256K &
done

 892.66 outbound
 891.34 inbound
```

We have used a `for` loop in the shell with just one iteration because that will be **much** easier to get both tests started at more or less the same time than doing it by hand. The global '`-P`' and '`-v`' options are used because we aren't interested in anything other than the throughput, and the global '`-B`' option is used to tag each output so we know which was inbound and which outbound relative to the system on which we were running netperf. Of course that sense is switched on the system running netserver :) The use of the global '`-i`' option is explained in Section 7.1 [Running Concurrent Netperf Tests], page 36.

Beginning with version 2.5.0 we can accomplish a similar result with the Chapter 9 [The Omni Tests], page 47 and Section 9.3.1 [Omni Output Selectors], page 51:

```
for i in 1
do
  netperf -H 192.168.1.3 -t omni -l 10 -P 0 -- \
    -d stream -s 256K -S 256K -o throughput,direction &
```

```
  netperf -H 192.168.1.3 -t omni -l 10 -P 0 -- \
    -d maerts -s 256K -S 256K -o throughput,direction &
done

805.26,Receive
828.54,Send
```

8.2 Bidirectional Transfer with TCP_RR

Starting with version 2.5.0 the `--enable-burst` configure option defaults to `yes`, and starting some time before version 2.5.0 but after 2.4.0 the global '`-f`' option would affect the "throughput" reported by request/response tests. If one uses the test-specific '`-b`' option to have several "transactions" in flight at one time and the test-specific '`-r`' option to increase their size, the test looks more and more like a single-connection bidirectional transfer than a simple request/response test.

So, putting it all together one can do something like:

```
netperf -f m -t TCP_RR -H 192.168.1.3 -v 2 -- -b 6 -r 32K -S 256K -S 256K
MIGRATED TCP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 192.168.1.
Local /Remote
Socket Size   Request  Resp.   Elapsed
Send   Recv   Size     Size    Time     Throughput
bytes  Bytes  bytes    bytes   secs.    10^6bits/sec

16384  87380  32768    32768   10.00    1821.30
524288 524288
Alignment      Offset         RoundTrip  Trans    Throughput
Local  Remote  Local  Remote  Latency    Rate     10^6bits/s
Send   Recv    Send   Recv    usec/Tran  per sec  Outbound   Inbound
    8      0       0      0   2015.402   3473.252 910.492    910.492
```

to get a bidirectional bulk-throughput result. As one can see, the -v 2 output will include a number of interesting, related values.

> **NOTE: The logic behind `--enable-burst` is very simple, and there are no calls to `poll()` or `select()` which means we want to make sure that the `send()` calls will never block, or we run the risk of deadlock with each side stuck trying to call `send()` and neither calling `recv()`.**

Fortunately, this is easily accomplished by setting a "large enough" socket buffer size with the test-specific '`-s`' and '`-S`' options. Presently this must be performed by the user. Future versions of netperf might attempt to do this automagically, but there are some issues to be worked-out.

8.3 Implications of Concurrent Tests vs Burst Request/Response

There are perhaps subtle but important differences between using concurrent unidirectional tests vs a burst-mode request to measure bidirectional performance.

Broadly speaking, a single "connection" or "flow" of traffic cannot make use of the services of more than one or two CPUs at either end. Whether one or two CPUs will be

used processing a flow will depend on the specifics of the stack(s) involved and whether or not the global '`-T`' option has been used to bind netperf/netserver to specific CPUs.

When using concurrent tests there will be two concurrent connections or flows, which means that upwards of four CPUs will be employed processing the packets (global '`-T`' used, no more than two if not), however, with just a single, bidirectional request/response test no more than two CPUs will be employed (only one if the global '`-T`' is not used).

If there is a CPU bottleneck on either system this may result in rather different results between the two methods.

Also, with a bidirectional request/response test there is something of a natural balance or synchronization between inbound and outbound - a response will not be sent until a request is received, and (once the burst level is reached) a subsequent request will not be sent until a response is received. This may mask favoritism in the NIC between inbound and outbound processing.

With two concurrent unidirectional tests there is no such synchronization or balance and any favoritism in the NIC may be exposed.

9 The Omni Tests

Beginning with version 2.5.0, netperf begins a migration to the 'omni' tests or "Two routines to measure them all." The code for the omni tests can be found in 'src/nettest_omni.c' and the goal is to make it easier for netperf to support multiple protocols and report a great many additional things about the systems under test. Additionally, a flexible output selection mechanism is present which allows the user to chose specifically what values she wishes to have reported and in what format.

The omni tests are included by default in version 2.5.0. To disable them, one must:

```
./configure --enable-omni=no ...
```

and remake netperf. Remaking netserver is optional because even in 2.5.0 it has "unmigrated" netserver side routines for the classic (eg 'src/nettest_bsd.c') tests.

9.1 Native Omni Tests

One access the omni tests "natively" by using a value of "OMNI" with the global '-t' test-selection option. This will then cause netperf to use the code in 'src/nettest_omni.c' and in particular the test-specific options parser for the omni tests. The test-specific options for the omni tests are a superset of those for "classic" tests. The options added by the omni tests are:

`-c` This explicitly declares that the test is to include connection establishment and tear-down as in either a TCP_CRR or TCP_CC test.

`-d <direction>`
This option sets the direction of the test relative to the netperf process. As of version 2.5.0 one can use the following in a case-insensitive manner:

`send, stream, transmit, xmit or 2`
Any of which will cause netperf to send to the netserver.

`recv, receive, maerts or 4`
Any of which will cause netserver to send to netperf.

`rr or 6` Either of which will cause a request/response test.

Additionally, one can specify two directions separated by a '|' character and they will be OR'ed together. In this way one can use the "Send|Recv" that will be emitted by the Section 9.3.1 [Omni Output Selectors], page 51 Section 9.3 [Omni Output Selection], page 50 when used with a request/response test.

`-k [Section 9.3 [Omni Output Selection], page 50]`
This option sets the style of output to "keyval" where each line of output has the form:

```
key=value
```

For example:

```
$ netperf -t omni -- -d rr -k "THROUGHPUT,THROUGHPUT_UNITS"
OMNI TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to localhost.localdomain (12
THROUGHPUT=59092.65
THROUGHPUT_UNITS=Trans/s
```

Using the '-k' option will override any previous, test-specific '-o' or '-O' option.

`-o` [Section 9.3 [Omni Output Selection], page 50]
: This option sets the style of output to "CSV" where there will be one line of comma-separated values, preceded by one line of column names unless the global '`-P`' option is used with a value of 0:

```
$ netperf -t omni -- -d rr -o "THROUGHPUT,THROUGHPUT_UNITS"
OMNI TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to localhost.localdomain (12
Throughput,Throughput Units
60999.07,Trans/s
```

Using the '`-o`' option will override any previous, test-specific '`-k`' or '`-O`' option.

`-O` [Section 9.3 [Omni Output Selection], page 50]
: This option sets the style of output to "human readable" which will look quite similar to classic netperf output:

```
$ netperf -t omni -- -d rr -O "THROUGHPUT,THROUGHPUT_UNITS"
OMNI TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to localhost.localdomain (12
Throughput Throughput
           Units


60492.57   Trans/s
```

Using the '`-O`' option will override any previous, test-specific '`-k`' or '`-o`' option.

`-t`
: This option explicitly sets the socket type for the test's data connection. As of version 2.5.0 the known socket types include "stream" and "dgram" for SOCK_STREAM and SOCK_DGRAM respectively.

`-T <protocol>`
: This option is used to explicitly set the protocol used for the test. It is case-insensitive. As of version 2.5.0 the protocols known to netperf include:

`TCP`	Select the Transmission Control Protocol
`UDP`	Select the User Datagram Protocol
`SDP`	Select the Sockets Direct Protocol
`DCCP`	Select the Datagram Congestion Control Protocol
`SCTP`	Select the Stream Control Transport Protocol
`udplite`	Select UDP Lite

The default is implicit based on other settings.

The omni tests also extend the interpretation of some of the classic, test-specific options for the BSD Sockets tests:

`-m <optionspec>`
: This can set the send size for either or both of the netperf and netserver sides of the test:

```
-m 32K
```

sets only the netperf-side send size to 32768 bytes, and or's-in transmit for the direction. This is effectively the same behaviour as for the classic tests.

```
-m ,32K
```

sets only the netserver side send size to 32768 bytes and or's-in receive for the direction.

```
-m 16K,32K
sets the netperf side send size to 16284 bytes, the netserver side
send size to 32768 bytes and the direction will be "Send|Recv."
```

`-M <optionspec>`

This can set the receive size for either or both of the netperf and netserver sides of the test:

```
-M 32K
```

sets only the netserver side receive size to 32768 bytes and or's-in send for the test direction.

```
-M ,32K
```

sets only the netperf side receive size to 32768 bytes and or's-in receive for the test direction.

```
-M 16K,32K
```

sets the netserver side receive size to 16384 bytes and the netperf side receive size to 32768 bytes and the direction will be "Send|Recv."

9.2 Migrated Tests

As of version 2.5.0 several tests have been migrated to use the omni code in '`src/nettest_omni.c`' for the core of their testing. A migrated test retains all its previous output code and so should still "look and feel" just like a pre-2.5.0 test with one exception - the first line of the test banners will include the word "MIGRATED" at the beginning as in:

```
$ netperf
MIGRATED TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to localhost.localdomai
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 87380  16384  16384    10.00    27175.27
```

The tests migrated in version 2.5.0 are:

- TCP_STREAM
- TCP_MAERTS
- TCP_RR
- TCP_CRR
- UDP_STREAM
- UDP_RR

It is expected that future releases will have additional tests migrated to use the "omni" functionality.

If one uses "omni-specific" test-specific options in conjunction with a migrated test, instead of using the classic output code, the new omni output code will be used. For example if one uses the '-k' test-specific option with a value of "MIN_LATENCY,MAX_LATENCY" with a migrated TCP_RR test one will see:

```
$ netperf -t tcp_rr -- -k THROUGHPUT,THROUGHPUT_UNITS
MIGRATED TCP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to localhost.
THROUGHPUT=60074.74
THROUGHPUT_UNITS=Trans/s
```

rather than:

```
$ netperf -t tcp_rr
MIGRATED TCP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to localhost.
Local /Remote
Socket Size   Request  Resp.   Elapsed  Trans.
Send   Recv   Size     Size    Time     Rate
bytes  Bytes  bytes    bytes   secs.    per sec

16384  87380  1        1       10.00    59421.52
16384  87380
```

9.3 Omni Output Selection

The omni test-specific '-k', '-o' and '-O' options take an optional `output selector` by which the user can configure what values are reported. The output selector can take several forms:

'filename'
: The output selections will be read from the named file. Within the file there can be up to four lines of comma-separated output selectors. This controls how many multi-line blocks of output are emitted when the '-O' option is used. This output, while not identical to "classic" netperf output, is inspired by it. Multiple lines have no effect for '-k' and '-o' options. Putting output selections in a file can be useful when the list of selections is long.

comma and/or semi-colon-separated list
: The output selections will be parsed from a comma and/or semi-colon-separated list of output selectors. When the list is given to a '-O' option a semi-colon specifies a new output block should be started. Semi-colons have the same meaning as commas when used with the '-k' or '-o' options. Depending on the command interpreter being used, the semi-colon may have to be escaped somehow to keep it from being interpreted by the command interpreter. This can often be done by enclosing the entire list in quotes.

all
: If the keyword **all** is specified it means that all known output values should be displayed at the end of the test. This can be a great deal of output. As of version 2.5.0 there are 157 different output selectors.

?
: If a "?" is given as the output selection, the list of all known output selectors will be displayed and no test actually run. When passed to the '-O' option they will be listed one per line. Otherwise they will be listed as a comma-separated

list. It may be necessary to protect the "?" from the command interpreter by escaping it or enclosing it in quotes.

`no selector`
: If nothing is given to the '`-k`', '`-o`' or '`-O`' option then the code selects a default set of output selectors inspired by classic netperf output. The format will be the '`human readable`' format emitted by the test-specific '`-O`' option.

The order of evaluation will first check for an output selection. If none is specified with the '`-k`', '`-o`' or '`-O`' option netperf will select a default based on the characteristics of the test. If there is an output selection, the code will first check for '`?`', then check to see if it is the magic '`all`' keyword. After that it will check for either '`,`' or '`;`' in the selection and take that to mean it is a comma and/or semi-colon-separated list. If none of those checks match, netperf will then assume the output specification is a filename and attempt to open and parse the file.

9.3.1 Omni Output Selectors

As of version 2.5.0 the output selectors are:

`OUTPUT_NONE`
: This is essentially a null output. For '`-k`' output it will simply add a line that reads "OUTPUT_NONE=" to the output. For '`-o`' it will cause an empty "column" to be included. For '`-O`' output it will cause extra spaces to separate "real" output.

`SOCKET_TYPE`
: This will cause the socket type (eg SOCK_STREAM, SOCK_DGRAM) for the data connection to be output.

`PROTOCOL` This will cause the protocol used for the data connection to be displayed.

`DIRECTION`
: This will display the data flow direction relative to the netperf process. Units: Send or Recv for a unidirectional bulk-transfer test, or Send|Recv for a request/response test.

`ELAPSED_TIME`
: This will display the elapsed time in seconds for the test.

`THROUGHPUT`
: This will display the throughput for the test. Units: As requested via the global '`-f`' option and displayed by the THROUGHPUT_UNITS output selector.

`THROUGHPUT_UNITS`
: This will display the units for what is displayed by the `THROUGHPUT` output selector.

`LSS_SIZE_REQ`
: This will display the local (netperf) send socket buffer size (aka SO_SNDBUF) requested via the command line. Units: Bytes.

`LSS_SIZE` This will display the local (netperf) send socket buffer size (SO_SNDBUF) immediately after the data connection socket was created. Peculiarities of dif-

ferent networking stacks may lead to this differing from the size requested via the command line. Units: Bytes.

LSS_SIZE_END
This will display the local (netperf) send socket buffer size (SO_SNDBUF) immediately before the data connection socket is closed. Peculiarities of different networking stacks may lead this to differ from the size requested via the command line and/or the size immediately after the data connection socket was created. Units: Bytes.

LSR_SIZE_REQ
This will display the local (netperf) receive socket buffer size (aka SO_RCVBUF) requested via the command line. Units: Bytes.

LSR_SIZE This will display the local (netperf) receive socket buffer size (SO_RCVBUF) immediately after the data connection socket was created. Peculiarities of different networking stacks may lead to this differing from the size requested via the command line. Units: Bytes.

LSR_SIZE_END
This will display the local (netperf) receive socket buffer size (SO_RCVBUF) immediately before the data connection socket is closed. Peculiarities of different networking stacks may lead this to differ from the size requested via the command line and/or the size immediately after the data connection socket was created. Units: Bytes.

RSS_SIZE_REQ
This will display the remote (netserver) send socket buffer size (aka SO_SNDBUF) requested via the command line. Units: Bytes.

RSS_SIZE This will display the remote (netserver) send socket buffer size (SO_SNDBUF) immediately after the data connection socket was created. Peculiarities of different networking stacks may lead to this differing from the size requested via the command line. Units: Bytes.

RSS_SIZE_END
This will display the remote (netserver) send socket buffer size (SO_SNDBUF) immediately before the data connection socket is closed. Peculiarities of different networking stacks may lead this to differ from the size requested via the command line and/or the size immediately after the data connection socket was created. Units: Bytes.

RSR_SIZE_REQ
This will display the remote (netserver) receive socket buffer size (aka SO_RCVBUF) requested via the command line. Units: Bytes.

RSR_SIZE This will display the remote (netserver) receive socket buffer size (SO_RCVBUF) immediately after the data connection socket was created. Peculiarities of different networking stacks may lead to this differing from the size requested via the command line. Units: Bytes.

RSR_SIZE_END
: This will display the remote (netserver) receive socket buffer size (SO_RCVBUF) immediately before the data connection socket is closed. Peculiarities of different networking stacks may lead this to differ from the size requested via the command line and/or the size immediately after the data connection socket was created. Units: Bytes.

LOCAL_SEND_SIZE
: This will display the size of the buffers netperf passed in any "send" calls it made on the data connection for a non-request/response test. Units: Bytes.

LOCAL_RECV_SIZE
: This will display the size of the buffers netperf passed in any "receive" calls it made on the data connection for a non-request/response test. Units: Bytes.

REMOTE_SEND_SIZE
: This will display the size of the buffers netserver passed in any "send" calls it made on the data connection for a non-request/response test. Units: Bytes.

REMOTE_RECV_SIZE
: This will display the size of the buffers netserver passed in any "receive" calls it made on the data connection for a non-request/response test. Units: Bytes.

REQUEST_SIZE
: This will display the size of the requests netperf sent in a request-response test. Units: Bytes.

RESPONSE_SIZE
: This will display the size of the responses netserver sent in a request-response test. Units: Bytes.

LOCAL_CPU_UTIL
: This will display the overall CPU utilization during the test as measured by netperf. Units: 0 to 100 percent.

LOCAL_CPU_PERCENT_USER
: This will display the CPU fraction spent in user mode during the test as measured by netperf. Only supported by netcpu_procstat. Units: 0 to 100 percent.

LOCAL_CPU_PERCENT_SYSTEM
: This will display the CPU fraction spent in system mode during the test as measured by netperf. Only supported by netcpu_procstat. Units: 0 to 100 percent.

LOCAL_CPU_PERCENT_IOWAIT
: This will display the fraction of time waiting for I/O to complete during the test as measured by netperf. Only supported by netcpu_procstat. Units: 0 to 100 percent.

LOCAL_CPU_PERCENT_IRQ
: This will display the fraction of time servicing interrupts during the test as measured by netperf. Only supported by netcpu_procstat. Units: 0 to 100 percent.

LOCAL_CPU_PERCENT_SWINTR
: This will display the fraction of time servicing softirqs during the test as measured by netperf. Only supported by netcpu_procstat. Units: 0 to 100 percent.

LOCAL_CPU_METHOD
: This will display the method used by netperf to measure CPU utilization. Units: single character denoting method.

LOCAL_SD
: This will display the service demand, or units of CPU consumed per unit of work, as measured by netperf. Units: microseconds of CPU consumed per either KB (K==1024) of data transferred or request/response transaction.

REMOTE_CPU_UTIL
: This will display the overall CPU utilization during the test as measured by netserver. Units 0 to 100 percent.

REMOTE_CPU_PERCENT_USER
: This will display the CPU fraction spent in user mode during the test as measured by netserver. Only supported by netcpu_procstat. Units: 0 to 100 percent.

REMOTE_CPU_PERCENT_SYSTEM
: This will display the CPU fraction spent in system mode during the test as measured by netserver. Only supported by netcpu_procstat. Units: 0 to 100 percent.

REMOTE_CPU_PERCENT_IOWAIT
: This will display the fraction of time waiting for I/O to complete during the test as measured by netserver. Only supported by netcpu_procstat. Units: 0 to 100 percent.

REMOTE_CPU_PERCENT_IRQ
: This will display the fraction of time servicing interrupts during the test as measured by netserver. Only supported by netcpu_procstat. Units: 0 to 100 percent.

REMOTE_CPU_PERCENT_SWINTR
: This will display the fraction of time servicing softirqs during the test as measured by netserver. Only supported by netcpu_procstat. Units: 0 to 100 percent.

REMOTE_CPU_METHOD
: This will display the method used by netserver to measure CPU utilization. Units: single character denoting method.

REMOTE_SD
: This will display the service demand, or units of CPU consumed per unit of work, as measured by netserver. Units: microseconds of CPU consumed per either KB (K==1024) of data transferred or request/response transaction.

SD_UNITS
: This will display the units for LOCAL_SD and REMOTE_SD

CONFIDENCE_LEVEL
: This will display the confidence level requested by the user either explicitly via the global '`-I`' option, or implicitly via the global '`-i`' option. The value will

be either 95 or 99 if confidence intervals have been requested or 0 if they were not. Units: Percent

`CONFIDENCE_INTERVAL`
: This will display the width of the confidence interval requested either explicitly via the global '`-I`' option or implicitly via the global '`-i`' option. Units: Width in percent of mean value computed. A value of -1.0 means that confidence intervals were not requested.

`CONFIDENCE_ITERATION`
: This will display the number of test iterations netperf undertook, perhaps while attempting to achieve the requested confidence interval and level. If confidence intervals were requested via the command line then the value will be between 3 and 30. If confidence intervals were not requested the value will be 1. Units: Iterations

`THROUGHPUT_CONFID`
: This will display the width of the confidence interval actually achieved for `THROUGHPUT` during the test. Units: Width of interval as percentage of reported throughput value.

`LOCAL_CPU_CONFID`
: This will display the width of the confidence interval actually achieved for overall CPU utilization on the system running netperf (`LOCAL_CPU_UTIL`) during the test, if CPU utilization measurement was enabled. Units: Width of interval as percentage of reported CPU utilization.

`REMOTE_CPU_CONFID`
: This will display the width of the confidence interval actually achieved for overall CPU utilization on the system running netserver (`REMOTE_CPU_UTIL`) during the test, if CPU utilization measurement was enabled. Units: Width of interval as percentage of reported CPU utilization.

`TRANSACTION_RATE`
: This will display the transaction rate in transactions per second for a request/response test even if the user has requested a throughput in units of bits or bytes per second via the global '`-f`' option. It is undefined for a non-request/response test. Units: Transactions per second.

`RT_LATENCY`
: This will display the average round-trip latency for a request/response test, accounting for number of transactions in flight at one time. It is undefined for a non-request/response test. Units: Microseconds per transaction

`BURST_SIZE`
: This will display the "burst size" or added transactions in flight in a request/response test as requested via a test-specific '`-b`' option. The number of transactions in flight at one time will be one greater than this value. It is undefined for a non-request/response test. Units: added Transactions in flight.

`LOCAL_TRANSPORT_RETRANS`
: This will display the number of retransmissions experienced on the data connection during the test as determined by netperf. A value of -1 means the attempt

to determine the number of retransmissions failed or the concept was not valid for the given protocol or the mechanism is not known for the platform. A value of -2 means it was not attempted. As of version 2.5.0 the meaning of values are in flux and subject to change. Units: number of retransmissions.

`REMOTE_TRANSPORT_RETRANS`
: This will display the number of retransmissions experienced on the data connection during the test as determined by netserver. A value of -1 means the attempt to determine the number of retransmissions failed or the concept was not valid for the given protocol or the mechanism is not known for the platform. A value of -2 means it was not attempted. As of version 2.5.0 the meaning of values are in flux and subject to change. Units: number of retransmissions.

`TRANSPORT_MSS`
: This will display the Maximum Segment Size (aka MSS) or its equivalent for the protocol being used during the test. A value of -1 means either the concept of an MSS did not apply to the protocol being used, or there was an error in retrieving it. Units: Bytes.

`LOCAL_SEND_THROUGHPUT`
: The throughput as measured by netperf for the successful "send" calls it made on the data connection. Units: as requested via the global '`-f`' option and displayed via the `THROUGHPUT_UNITS` output selector.

`LOCAL_RECV_THROUGHPUT`
: The throughput as measured by netperf for the successful "receive" calls it made on the data connection. Units: as requested via the global '`-f`' option and displayed via the `THROUGHPUT_UNITS` output selector.

`REMOTE_SEND_THROUGHPUT`
: The throughput as measured by netserver for the successful "send" calls it made on the data connection. Units: as requested via the global '`-f`' option and displayed via the `THROUGHPUT_UNITS` output selector.

`REMOTE_RECV_THROUGHPUT`
: The throughput as measured by netserver for the successful "receive" calls it made on the data connection. Units: as requested via the global '`-f`' option and displayed via the `THROUGHPUT_UNITS` output selector.

`LOCAL_CPU_BIND`
: The CPU to which netperf was bound, if at all, during the test. A value of -1 means that netperf was not explicitly bound to a CPU during the test. Units: CPU ID

`LOCAL_CPU_COUNT`
: The number of CPUs (cores, threads) detected by netperf. Units: CPU count.

`LOCAL_CPU_PEAK_UTIL`
: The utilization of the CPU most heavily utilized during the test, as measured by netperf. This can be used to see if any one CPU of a multi-CPU system was saturated even though the overall CPU utilization as reported by `LOCAL_CPU_UTIL` was low. Units: 0 to 100%

LOCAL_CPU_PEAK_ID
: The id of the CPU most heavily utilized during the test as determined by netperf. Units: CPU ID.

LOCAL_CPU_MODEL
: Model information for the processor(s) present on the system running netperf. Assumes all processors in the system (as perceived by netperf) on which netperf is running are the same model. Units: Text

LOCAL_CPU_FREQUENCY
: The frequency of the processor(s) on the system running netperf, at the time netperf made the call. Assumes that all processors present in the system running netperf are running at the same frequency. Units: MHz

REMOTE_CPU_BIND
: The CPU to which netserver was bound, if at all, during the test. A value of -1 means that netperf was not explicitly bound to a CPU during the test. Units: CPU ID

REMOTE_CPU_COUNT
: The number of CPUs (cores, threads) detected by netserver. Units: CPU count.

REMOTE_CPU_PEAK_UTIL
: The utilization of the CPU most heavily utilized during the test, as measured by netserver. This can be used to see if any one CPU of a multi-CPU system was saturated even though the overall CPU utilization as reported by REMOTE_CPU_UTIL was low. Units: 0 to 100%

REMOTE_CPU_PEAK_ID
: The id of the CPU most heavily utilized during the test as determined by netserver. Units: CPU ID.

REMOTE_CPU_MODEL
: Model information for the processor(s) present on the system running netserver. Assumes all processors in the system (as perceived by netserver) on which netserver is running are the same model. Units: Text

REMOTE_CPU_FREQUENCY
: The frequency of the processor(s) on the system running netserver, at the time netserver made the call. Assumes that all processors present in the system running netserver are running at the same frequency. Units: MHz

SOURCE_PORT
: The port ID/service name to which the data socket created by netperf was bound. A value of 0 means the data socket was not explicitly bound to a port number. Units: ASCII text.

SOURCE_ADDR
: The name/address to which the data socket created by netperf was bound. A value of 0.0.0.0 means the data socket was not explicitly bound to an address. Units: ASCII text.

SOURCE_FAMILY
: The address family to which the data socket created by netperf was bound. A value of 0 means the data socket was not explicitly bound to a given address family. Units: ASCII text.

DEST_PORT
: The port ID to which the data socket created by netserver was bound. A value of 0 means the data socket was not explicitly bound to a port number. Units: ASCII text.

DEST_ADDR
: The name/address of the data socket created by netserver. Units: ASCII text.

DEST_FAMILY
: The address family to which the data socket created by netserver was bound. A value of 0 means the data socket was not explicitly bound to a given address family. Units: ASCII text.

LOCAL_SEND_CALLS
: The number of successful "send" calls made by netperf against its data socket. Units: Calls.

LOCAL_RECV_CALLS
: The number of successful "receive" calls made by netperf against its data socket. Units: Calls.

LOCAL_BYTES_PER_RECV
: The average number of bytes per "receive" call made by netperf against its data socket. Units: Bytes.

LOCAL_BYTES_PER_SEND
: The average number of bytes per "send" call made by netperf against its data socket. Units: Bytes.

LOCAL_BYTES_SENT
: The number of bytes successfully sent by netperf through its data socket. Units: Bytes.

LOCAL_BYTES_RECVD
: The number of bytes successfully received by netperf through its data socket. Units: Bytes.

LOCAL_BYTES_XFERD
: The sum of bytes sent and received by netperf through its data socket. Units: Bytes.

LOCAL_SEND_OFFSET
: The offset from the alignment of the buffers passed by netperf in its "send" calls. Specified via the global '`-o`' option and defaults to 0. Units: Bytes.

LOCAL_RECV_OFFSET
: The offset from the alignment of the buffers passed by netperf in its "receive" calls. Specified via the global '`-o`' option and defaults to 0. Units: Bytes.

LOCAL_SEND_ALIGN
: The alignment of the buffers passed by netperf in its "send" calls as specified via the global '-a' option. Defaults to 8. Units: Bytes.

LOCAL_RECV_ALIGN
: The alignment of the buffers passed by netperf in its "receive" calls as specified via the global '-a' option. Defaults to 8. Units: Bytes.

LOCAL_SEND_WIDTH
: The "width" of the ring of buffers through which netperf cycles as it makes its "send" calls. Defaults to one more than the local send socket buffer size divided by the send size as determined at the time the data socket is created. Can be used to make netperf more processor data cache unfriendly. Units: number of buffers.

LOCAL_RECV_WIDTH
: The "width" of the ring of buffers through which netperf cycles as it makes its "receive" calls. Defaults to one more than the local receive socket buffer size divided by the receive size as determined at the time the data socket is created. Can be used to make netperf more processor data cache unfriendly. Units: number of buffers.

LOCAL_SEND_DIRTY_COUNT
: The number of bytes to "dirty" (write to) before netperf makes a "send" call. Specified via the global '-k' option, which requires that –enable-dirty=yes was specified with the configure command prior to building netperf. Units: Bytes.

LOCAL_RECV_DIRTY_COUNT
: The number of bytes to "dirty" (write to) before netperf makes a "recv" call. Specified via the global '-k' option which requires that –enable-dirty was specified with the configure command prior to building netperf. Units: Bytes.

LOCAL_RECV_CLEAN_COUNT
: The number of bytes netperf should read "cleanly" before making a "receive" call. Specified via the global '-k' option which requires that –enable-dirty was specified with configure command prior to building netperf. Clean reads start were dirty writes ended. Units: Bytes.

LOCAL_NODELAY
: Indicates whether or not setting the test protocol-specific "no delay" (eg TCP_NODELAY) option on the data socket used by netperf was requested by the test-specific '-D' option and successful. Units: 0 means no, 1 means yes.

LOCAL_CORK
: Indicates whether or not TCP_CORK was set on the data socket used by netperf as requested via the test-specific '-C' option. 1 means yes, 0 means no/not applicable.

REMOTE_SEND_CALLS
REMOTE_RECV_CALLS
REMOTE_BYTES_PER_RECV
REMOTE_BYTES_PER_SEND
REMOTE_BYTES_SENT
REMOTE_BYTES_RECVD
REMOTE_BYTES_XFERD
REMOTE_SEND_OFFSET
REMOTE_RECV_OFFSET
REMOTE_SEND_ALIGN
REMOTE_RECV_ALIGN
REMOTE_SEND_WIDTH
REMOTE_RECV_WIDTH
REMOTE_SEND_DIRTY_COUNT
REMOTE_RECV_DIRTY_COUNT
REMOTE_RECV_CLEAN_COUNT
REMOTE_NODELAY
REMOTE_CORK
: These are all like their "LOCAL_" counterparts only for the netserver rather than netperf.

LOCAL_SYSNAME
: The name of the OS (eg "Linux") running on the system on which netperf was running. Units: ASCII Text

LOCAL_SYSTEM_MODEL
: The model name of the system on which netperf was running. Units: ASCII Text.

LOCAL_RELEASE
: The release name/number of the OS running on the system on which netperf was running. Units: ASCII Text

LOCAL_VERSION
: The version number of the OS running on the system on which netperf was running. Units: ASCII Text

LOCAL_MACHINE
: The machine architecture of the machine on which netperf was running. Units: ASCII Text.

REMOTE_SYSNAME
REMOTE_SYSTEM_MODEL
REMOTE_RELEASE
REMOTE_VERSION
REMOTE_MACHINE
: These are all like their "LOCAL_" counterparts only for the netserver rather than netperf.

LOCAL_INTERFACE_NAME
: The name of the probable egress interface through which the data connection went on the system running netperf. Example: eth0. Units: ASCII Text.

LOCAL_INTERFACE_VENDOR
: The vendor ID of the probable egress interface through which traffic on the data connection went on the system running netperf. Units: Hexadecimal IDs as might be found in a 'pci.ids' file or at the PCI ID Repository.

LOCAL_INTERFACE_DEVICE
: The device ID of the probable egress interface through which traffic on the data connection went on the system running netperf. Units: Hexadecimal IDs as might be found in a 'pci.ids' file or at the PCI ID Repository.

LOCAL_INTERFACE_SUBVENDOR
: The sub-vendor ID of the probable egress interface through which traffic on the data connection went on the system running netperf. Units: Hexadecimal IDs as might be found in a 'pci.ids' file or at the PCI ID Repository.

LOCAL_INTERFACE_SUBDEVICE
: The sub-device ID of the probable egress interface through which traffic on the data connection went on the system running netperf. Units: Hexadecimal IDs as might be found in a 'pci.ids' file or at the PCI ID Repository.

LOCAL_DRIVER_NAME
: The name of the driver used for the probable egress interface through which traffic on the data connection went on the system running netperf. Units: ASCII Text.

LOCAL_DRIVER_VERSION
: The version string for the driver used for the probable egress interface through which traffic on the data connection went on the system running netperf. Units: ASCII Text.

LOCAL_DRIVER_FIRMWARE
: The firmware version for the driver used for the probable egress interface through which traffic on the data connection went on the system running netperf. Units: ASCII Text.

LOCAL_DRIVER_BUS
: The bus address of the probable egress interface through which traffic on the data connection went on the system running netperf. Units: ASCII Text.

LOCAL_INTERFACE_SLOT
: The slot ID of the probable egress interface through which traffic on the data connection went on the system running netperf. Units: ASCII Text.

`REMOTE_INTERFACE_NAME`
`REMOTE_INTERFACE_VENDOR`
`REMOTE_INTERFACE_DEVICE`
`REMOTE_INTERFACE_SUBVENDOR`
`REMOTE_INTERFACE_SUBDEVICE`
`REMOTE_DRIVER_NAME`
`REMOTE_DRIVER_VERSION`
`REMOTE_DRIVER_FIRMWARE`
`REMOTE_DRIVER_BUS`
`REMOTE_INTERFACE_SLOT`

These are all like their "LOCAL_" counterparts only for the netserver rather than netperf.

`LOCAL_INTERVAL_USECS`

The interval at which bursts of operations (sends, receives, transactions) were attempted by netperf. Specified by the global '`-w`' option which requires –enable-intervals to have been specified with the configure command prior to building netperf. Units: Microseconds (though specified by default in milliseconds on the command line)

`LOCAL_INTERVAL_BURST`

The number of operations (sends, receives, transactions depending on the test) which were attempted by netperf each LOCAL_INTERVAL_USECS units of time. Specified by the global '`-b`' option which requires –enable-intervals to have been specified with the configure command prior to building netperf. Units: number of operations per burst.

`REMOTE_INTERVAL_USECS`

The interval at which bursts of operations (sends, receives, transactions) were attempted by netserver. Specified by the global '`-w`' option which requires –enable-intervals to have been specified with the configure command prior to building netperf. Units: Microseconds (though specified by default in milliseconds on the command line)

`REMOTE_INTERVAL_BURST`

The number of operations (sends, receives, transactions depending on the test) which were attempted by netperf each LOCAL_INTERVAL_USECS units of time. Specified by the global '`-b`' option which requires –enable-intervals to have been specified with the configure command prior to building netperf. Units: number of operations per burst.

LOCAL_SECURITY_TYPE_ID
LOCAL_SECURITY_TYPE
LOCAL_SECURITY_ENABLED_NUM
LOCAL_SECURITY_ENABLED
LOCAL_SECURITY_SPECIFIC
REMOTE_SECURITY_TYPE_ID
REMOTE_SECURITY_TYPE
REMOTE_SECURITY_ENABLED_NUM
REMOTE_SECURITY_ENABLED
REMOTE_SECURITY_SPECIFIC
: A bunch of stuff related to what sort of security mechanisms (eg SELINUX) were enabled on the systems during the test.

RESULT_BRAND
: The string specified by the user with the global '-B' option. Units: ASCII Text.

UUID
: The universally unique identifier associated with this test, either generated automagically by netperf, or passed to netperf via an omni test-specific '-u' option. Note: Future versions may make this a global command-line option. Units: ASCII Text.

MIN_LATENCY
: The minimum "latency" or operation time (send, receive or request/response exchange depending on the test) as measured on the netperf side when the global '-j' option was specified. Units: Microseconds.

MAX_LATENCY
: The maximum "latency" or operation time (send, receive or request/response exchange depending on the test) as measured on the netperf side when the global '-j' option was specified. Units: Microseconds.

P50_LATENCY
: The 50th percentile value of "latency" or operation time (send, receive or request/response exchange depending on the test) as measured on the netperf side when the global '-j' option was specified. Units: Microseconds.

P90_LATENCY
: The 90th percentile value of "latency" or operation time (send, receive or request/response exchange depending on the test) as measured on the netperf side when the global '-j' option was specified. Units: Microseconds.

P99_LATENCY
: The 99th percentile value of "latency" or operation time (send, receive or request/response exchange depending on the test) as measured on the netperf side when the global '-j' option was specified. Units: Microseconds.

MEAN_LATENCY
: The average "latency" or operation time (send, receive or request/response exchange depending on the test) as measured on the netperf side when the global '-j' option was specified. Units: Microseconds.

`STDDEV_LATENCY`
: The standard deviation of "latency" or operation time (send, receive or request/response exchange depending on the test) as measured on the netperf side when the global '`-j`' option was specified. Units: Microseconds.

`COMMAND_LINE`
: The full command line used when invoking netperf. Units: ASCII Text.

`OUTPUT_END`
: While emitted with the list of output selectors, it is ignored when specified as an output selector.

10 Other Netperf Tests

Apart from the typical performance tests, netperf contains some tests which can be used to streamline measurements and reporting. These include CPU rate calibration (present) and host identification (future enhancement).

10.1 CPU rate calibration

Some of the CPU utilization measurement mechanisms of netperf work by comparing the rate at which some counter increments when the system is idle with the rate at which that same counter increments when the system is running a netperf test. The ratio of those rates is used to arrive at a CPU utilization percentage.

This means that netperf must know the rate at which the counter increments when the system is presumed to be "idle." If it does not know the rate, netperf will measure it before starting a data transfer test. This calibration step takes 40 seconds for each of the local or remote systems, and if repeated for each netperf test would make taking repeated measurements rather slow.

Thus, the netperf CPU utilization options '`-c`' and and '`-C`' can take an optional calibration value. This value is used as the "idle rate" and the calibration step is not performed. To determine the idle rate, netperf can be used to run special tests which only report the value of the calibration - they are the LOC_CPU and REM_CPU tests. These return the calibration value for the local and remote system respectively. A common way to use these tests is to store their results into an environment variable and use that in subsequent netperf commands:

```
LOC_RATE=`netperf -t LOC_CPU`
REM_RATE=`netperf -H <remote> -t REM_CPU`
netperf -H <remote> -c $LOC_RATE -C $REM_RATE ... -- ...
...
netperf -H <remote> -c $LOC_RATE -C $REM_RATE ... -- ...
```

If you are going to use netperf to measure aggregate results, it is important to use the LOC_CPU and REM_CPU tests to get the calibration values first to avoid issues with some of the aggregate netperf tests transferring data while others are "idle" and getting bogus calibration values. When running aggregate tests, it is very important to remember that any one instance of netperf does not know about the other instances of netperf. It will report global CPU utilization and will calculate service demand believing it was the only thing causing that CPU utilization. So, you can use the CPU utilization reported by netperf in an aggregate test, but you have to calculate service demands by hand.

10.2 UUID Generation

Beginning with version 2.5.0 netperf can generate Universally Unique IDentifiers (UUIDs). This can be done explicitly via the "UUID" test:

```
$ netperf -t UUID
2c8561ae-9ebd-11e0-a297-0f5bfa0349d0
```

In and of itself, this is not terribly useful, but used in conjunction with the test-specific '`-u`' option of an "omni" test to set the UUID emitted by the Section 9.3.1 [Omni Output

Selectors], page 51 output selector, it can be used to tie-together the separate instances of an aggregate netperf test. Say, for instance if they were inserted into a database of some sort.

11 Address Resolution

Netperf versions 2.4.0 and later have merged IPv4 and IPv6 tests so the functionality of the tests in 'src/nettest_ipv6.c' has been subsumed into the tests in 'src/nettest_bsd.c' This has been accomplished in part by switching from gethostbyname()to getaddrinfo() exclusively. While it was theoretically possible to get multiple results for a hostname from gethostbyname() it was generally unlikely and netperf's ignoring of the second and later results was not much of an issue.

Now with getaddrinfo and particularly with AF_UNSPEC it is increasingly likely that a given hostname will have multiple associated addresses. The establish_control() routine of 'src/netlib.c' will indeed attempt to chose from among all the matching IP addresses when establishing the control connection. Netperf does not _really_ care if the control connection is IPv4 or IPv6 or even mixed on either end.

However, the individual tests still ass-u-me that the first result in the address list is the one to be used. Whether or not this will turn-out to be an issue has yet to be determined.

If you do run into problems with this, the easiest workaround is to specify IP addresses for the data connection explicitly in the test-specific '-H' and '-L' options. At some point, the netperf tests _may_ try to be more sophisticated in their parsing of returns from getaddrinfo() - straw-man patches to netperf-feedback@netperf.org would of course be most welcome :)

Netperf has leveraged code from other open-source projects with amenable licensing to provide a replacement getaddrinfo() call on those platforms where the configure script believes there is no native getaddrinfo call. As of this writing, the replacement getaddrinfo() as been tested on HP-UX 11.0 and then presumed to run elsewhere.

12 Enhancing Netperf

Netperf is constantly evolving. If you find you want to make enhancements to netperf, by all means do so. If you wish to add a new "suite" of tests to netperf the general idea is to:

1. Add files '`src/nettest_mumble.c`' and '`src/nettest_mumble.h`' where mumble is replaced with something meaningful for the test-suite.
2. Add support for an appropriate '`--enable-mumble`' option in '`configure.ac`'.
3. Edit '`src/netperf.c`', '`netsh.c`', and '`netserver.c`' as required, using #ifdef WANT_MUMBLE.
4. Compile and test

However, with the addition of the "omni" tests in version 2.5.0 it is preferred that one attempt to make the necessary changes to '`src/nettest_omni.c`' rather than adding new source files, unless this would make the omni tests entirely too complicated.

If you wish to submit your changes for possible inclusion into the mainline sources, please try to base your changes on the latest available sources. (See Section 2.1 [Getting Netperf Bits], page 3.) and then send email describing the changes at a high level to `netperf-feedback@netperf.org` or perhaps `netperf-talk@netperf.org`. If the consensus is positive, then sending context `diff` results to `netperf-feedback@netperf.org` is the next step. From that point, it is a matter of pestering the Netperf Contributing Editor until he gets the changes incorporated :)

13 Netperf4

Netperf4 is the shorthand name given to version 4.X.X of netperf. This is really a separate benchmark more than a newer version of netperf, but it is a descendant of netperf so the netperf name is kept. The facetious way to describe netperf4 is to say it is the egg-laying-woolly-milk-pig version of netperf :) The more respectful way to describe it is to say it is the version of netperf with support for synchronized, multiple-thread, multiple-test, multiple-system, network-oriented benchmarking.

Netperf4 is still undergoing evolution. Those wishing to work with or on netperf4 are encouraged to join the netperf-dev mailing list and/or peruse the current sources.

Concept Index

Option Index