# Care and Feeding of Netperf

Versions 2.4.0 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
- OpenVMS
- 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. 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 (`netperf-feedback@netperf.org`) for possible inclusion into subsequent versions of netperf.

The netperf-talk (`netperf-talk@netperf.org`) 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 and you must first subscribe by sending email to netperf-talk-request (`netperf-talk-request@netperf.org`).

## 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 (`ftp://ftp.cup.hp.com/dist/networking/benchmarks/netperf/`) for "released" versions of the bits. Pre-release versions of the bits can be retrieved via anonymous FTP from the experimental (`ftp://ftp.cup.hp.com/dist/networking/benchmarks/netperf/experimental/`) subdirectory.

For convenience and ease of remembering, a link to the download site is provided via the NetperfPage (`http://www.netperf.org/`)

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.cup.hp.com. From time to time a kind soul or souls has packaged netperf as a Debian package available via the apt-get mechanism. I would be most interested in learning how to enhance the makefiles to make that easier for people, and perhaps to generate RPM's and HP-UX swinstall"depots."

## 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.

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|unix|dlpi|sctp]=yes` options to the configure command. As of this writing, the configure script will not include those tests automagically.

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 automagical 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 (15.4.89.214) port 0 AF_INET : histogram
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
```

Long-time users of netperf will notice the expansion of the main test header. This stems from the merging-in of IPv6 with the standard IPv4 tests and the addition of code to specify addressing information for both sides of the data connection.

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 –enable-demo mode output:

```
src/netperf -D 1.35 -H lag -f M
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) port 0 AF_INET : demo
Interim result:    9.66 MBytes/s over 1.67 seconds
Interim result:    9.64 MBytes/s over 1.35 seconds
Interim result:    9.58 MBytes/s over 1.36 seconds
Interim result:    9.51 MBytes/s over 1.36 seconds
Interim result:    9.71 MBytes/s over 1.35 seconds
Interim result:    9.66 MBytes/s over 1.36 seconds
Interim result:    9.61 MBytes/s over 1.36 seconds
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    MBytes/sec

 32768  16384  16384    10.00       9.61
```

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 without relying on interval timers, but by calculating how many units of work must be performed to take at least the desired interval.

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 (127.0.0.1) port 0 AF_INET
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 equivalent. When you execute netperf, the first that that will happen is the establishment of 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 as many systems offer mechanisms that are at best il-suited to measuring CPU utilization in high interrupt rate (eg networking) situations.

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.

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 11.23 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 where by 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 consumtion, it has been suggested that this mechanism may become slightly confsed, in which case using BIOS 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 11iv1 and later are known 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.

# 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).

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."

`-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`
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. [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 transfered 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 optional for most tests, this option is required for a test utilizing the sendfile() or related calls because sendfile tests need a name of a file to reference.

`-h` This option causes netperf to display its 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 "tardy" 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 their data connections, 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.]

`-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.]

`-I <optionspec>`

This option enables the calculation of confidence intervals and sets the confidence and width parameters with the first have 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 netperf 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 (15.244.44.58) port 0 AF_INET : +/-2.5%  99% conf.
!!! 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
```

Where we see that netperf did not meet the desired convidence 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.

**-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.

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.

If netperf determines that the desired confidence intervals have not been met, it emits a noticeable warning.

**-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.

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.

`-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.

`-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 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 by on the remote 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]

`-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 20, Section 5.2.2 [TCP_MAERTS], page 21, Section 5.2.3 [TCP_SENDFILE], page 21, Section 6.2.1 [TCP_RR], page 28, Section 6.2.3 [TCP_CRR], page 29, Section 6.2.2 [TCP_CC], page 29
- Section 5.2.4 [UDP_STREAM], page 22, Section 6.2.4 [UDP_RR], page 30
- Section 5.2.5 [XTI_TCP_STREAM], page 23, Section 6.2.5 [XTI_TCP_RR], page 30, Section 6.2.7 [XTI_TCP_CRR], page 31, Section 6.2.6 [XTI_TCP_CC], page 31
- Section 5.2.6 [XTI_UDP_STREAM], page 23, Section 6.2.8 [XTI_UDP_RR], page 31
- Section 5.2.7 [SCTP_STREAM], page 23, Section 6.2.11 [SCTP_RR], page 31
- Section 5.2.8 [DLCO_STREAM], page 24, Section 6.2.10 [DLCO_RR], page 31, Section 5.2.9 [DLCL_STREAM], page 24, Section 6.2.9 [DLCL_RR], page 31
- Chapter 7 [Other Netperf Tests], page 32, Chapter 7 [Other Netperf Tests], page 32

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

`-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]

`-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 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.

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 (ftp://ftp.cup.hp.com/dist/networking/tools/) 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'. While it was written with HP-UX's netstat in mind, the annotated netstat (ftp://ftp.cup.hp.com/dist/networking/briefs/annotated_netstat.txt) 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 send and receive socket buffer sizes for the control 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. [Default: 0 - use the system's default socket buffer sizes]

`-S <sizespec>`
This option sets the remote send and/or receive socket buffer sizes for the control 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. [Default: 0 - 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 18, 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 an 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 28 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) port 0 AF_INET
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. Througput 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 20 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 20 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) port 0 AF_INET
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 20 test except netperf calls the platform's equivalent to HP-UX's `sendfile()` 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.

The Section 4.2 [Global Options], page 10 is required for this test and it must specify a file of at least the size of the send ring (See Section 4.2 [Global Options], page 10.) multiplied by the send size (See Section 5.2 [Options common to TCP UDP and SCTP tests], page 18.). All other TCP-specific options are 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) port 0 AF_INET
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:

```
$ ../src/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) port 0 AF_INET
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 20 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 (192.168.2.125) port 0 AF_INET
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 (192.168.2.125) port 0 AF_INET
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 (192.168.2.125) port 0 AF_INET
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
```

### 5.2.5 XTI_TCP_STREAM

An XTI_TCP_STREAM test is simply a Section 5.2.1 [TCP_STREAM], page 20 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 22 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 20 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 '-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 20 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 DPLI 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 22 test in that both make use of unreliable/best-effort, connection-less transports. The DLCL_STREAM test differs from the Section 5.2.4 [UDP_STREAM], page 22 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 24 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 20 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-unix=yes`. The remote netserver must have also been configured with `--enable-unix=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 20 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 24 test are applicable to a DG_STREAM test.

The DG_STREAM test is only present if netperf was configured with `--enable-unix=yes`. The remote netserver must have also been configured with `--enable-unix=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 can cover a multitude of latency and even path-length sins, they 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 at time test such as those found in netperf, is particularly sensitive to the path-length of the networking stack. An _RR test can also uncover those platforms where the NIC's 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 transfered per second, 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. Netperf does not do either of these on its own but leaves them as exercises to the benchmarker.

## 6.1 Issues in Reqeust/Response

Most if not all the Section 5.1 [Issues in Bulk Transfer], page 17 apply to request/response. The issue of round-trip latency is even more important as netperf only has one transaction outstanding at a time.

A single instance of an _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.

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 retrnamission timer expiring.

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 distinguised 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 drvier 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 send and receive socket buffer sizes for the control 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 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. [Default: 0 - use the system's default socket buffer sizes]

`-S <sizespec>`
: This option sets the remote send and/or receive socket buffer sizes for the control 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 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. [Default: 0 - 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 though-of as a user-space to user-space `ping` with no think time - it is 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 TCP_CC or TCP_CRR 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 (192.168.2.125) port 0 AF_INET
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 :)

### 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 is 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. 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_RR test is like a merger of a TCP_RR and TCP_CC 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.

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 (192.168.1.213) port 0 AF_INET
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 28 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

### 6.2.7 XTI_TCP_CRR

### 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 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).

## 7.1 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 ystems, 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.

# 8 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.

# 9 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 apropriate '--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

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 concensus 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 :)

One of these days, it is hoped that some sort of public version control system will be setup - perhaps on netperf.org - to make this whole process easier.

# Index