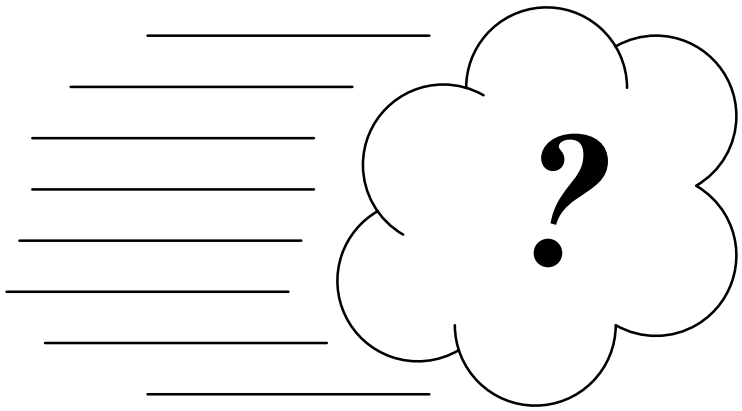
Netperf:

A Network Performance Benchmark

Revision 2.1

Information Networks Division
Hewlett−Packard Company
February 15, 1996



Section 0. The Legal Stuff

Section 1. Introduction

Netperf is a benchmark that can be used to measure various aspects of networking performance. Its primary focus is on bulk data transfer and request/response performance using either TCP or UDP and the Berkeley Sockets interface. There are optional tests available to measure the performance of DLPI, Unix Domain Sockets, the Fore ATM API and the HP HiPPI LLA interface.

This tool is maintained and informally supported by the IND Networking Performance Team. It is **NOT** supported via any of the normal Hewlett−Packard support channels. You are free to make enhancements and modifications to this tool.

This document is organized (loosely) into several sections as follows:

- Section 1. is what you are reading right now.
- Section 2. describes how to get the netperf bits and how to set−up your system to run netperf. It also describes a simple way to verify that the installation has been successful.
- Section 3. describes the design of netperf.
- Section 4. describes netperf's bulk data transfer tests and their command line options.
- Section 5. describes netperf's request−response tests and their command options.
- Section 6. describes some of the supporting test types of netperf and their command line options.
- Section 7. provides a description of the global command−line options for netperf.
- Section 8. provides some examples of netperf usage.
- Section 9. lists the changes and fixes in this revision of netperf.
- Section 10. lists several known problems with this revision of netperf.
- Section 11. provides some troubleshooting assistance.

We thank you in advance for your comments, and hope that you find this tool useful.

The maintainers of netperf.

"How fast is it? It's so fast, that ..." ;−)

Conventions and Definitions

You may not be familiar with some of the conventions and definitions used by this document. Generally, items of particular importance, command line options, and commands will be in **boldface** type. Filenames and command line items requiring user substitution will appear in *italicized* type.

A *sizespec* is a one or two item list passed with a command line option that can set the value of one or two netperf parameters. If you wish to set both parameters to separate values, items should be separated by a comma − Eg "parm1,parm2". If you wish to set the first parameter without altering the value of the second, you should follow the first item with a comma − Eg "parm1,". Likewise, precede the item with a comma if you wish to set only the second parameter − Eg ",parm2". An item without a comma will set both parameters. This last mode is the one most frequently used.

Netperf has two types of command line options. The first are global command line options. They are essentially any option that is not tied to a particular test, or group of tests. An example of a global command line option is the test type. The second options are test specific options. These are options which are only applicable to a particular test. 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, test specific second. They must be separated from each other by a "−−" (two dashes). If you wish to give test specific options only, they must be preceded by "−−". (EG ./netperf −− −m 1024)

Section 2. Installing Netperf

Netperf primary form of distribution is source code. This is to allow installation on systems other than those to which the authors have access and thus the ability to create binaries. There are two ways to install netperf. The first runs the netperf server program, netserver, as a child of inetd, which requires that the installer of netperf be able to edit the files */etc/services* and */etc/inetd.conf* (or their equivalent). The second is to run netserver as a standalone daemon. This second method does not require edit capabilities on */etc/services* and */etc/inetd.conf*. but does mean that you must remember to run the netserver program explicitly. The second method is required for Windows NT.

This manual assumes that those wishing to measure networking performance already know how to use anonymous FTP.

Getting the netperf bits from the Internet

For those people connected to the Internet, netperf is available via WWW. If you are not connected to the Internet such that you can use WWW, then you may be able to retrieve netperf via FTP or an FTP mail server. If all else fails, you can send email to Netperf Request <netperf−request@netperf.cup.hp.com>.

If you have a WWW browser, you can retrieve netperf via the Netperf Page. It is located at The Netperf Page . Follow the links from that page.

Netperf source bits are also available via anonymous FTP from **ftp.cup.hp.com** in the directory *dist/networking/benchmarks*. You should use **binary mode** transfers when bringing over the bits as you will be grabbing the latest copy of this document along with the netperf C source files.

NOTE: Older versions of netperf were available via anonymous FTP from **col.hp.com** under the directory *dist/networking/benchmarks/*. Other servers on the Internet may have copies. The “primary” place to go for netperf bits is ftp.cup.hp.com.

While the netperf source bits can be placed anywhere on the system, this manual will assume that the source bits are placed in the directory */opt/netperf/src*. Previous revisions of netperf were assumed to be placed in */usr/etc/net_perf/src*, but this has been changed to better emphasize that netperf is not an official Hewlett−Packard product. You are free to place netperf wherever you like, provided that you make the necessary modifications to the scripts.

Installing the bits

Once you have placed the netperf source bits onto the system, it is necessary to compile them and perform some editing tasks. This section assumes that you have elected to install the benchmark server, netserver, as a child of inetd.

The netperf distribution includes a makefile which assumes the existence of the directory */opt/netperf*. If you do not wish to have netperf installed there, it will be necessary for you to edit the makefile. To assist in this process, obvious markers have been placed in the makefile to indicate what must be changed. Also, some systems require different compile switches and libraries. For those systems where the requirements were known to the authors, comments have been added to the makefile.

Once the makefile is customized as needed, simply enter the command:

```
$ make install
```

from within the netperf source directory. The netperf executables will be compiled and copied into */opt/netperf* or the location specified in the makefile. Make will also copy the sample script files into the same place and verify that they are set to be executable.

If you do not have root access, or do not wish to install netperf as a child of inetd, skip to the subsection titled "Running netserver as a standalone Daemon."

Now that the executables have been created, it is necessary to edit the */etc/services* and */etc/inetd.conf* files. If you have decided to keep the netperf executables someplace other than */opt/netperf*, alter these lines accordingly. This editing step generally requires root access.

Add this line to the */etc/services* file:

```
netperf         12865/tcp
```

Then add this line to the */etc/inetd.conf* file:

```
netperf stream tcp nowait root /opt/netperf/netserver netserver
```

Running as root is probably no longer required (It was used on older versions of netperf which read from /dev/kmem) so if you are uncomfortable running netserver as root, you can pick another id. Once the files have been edited, it is necessary to have inetd re-configure itself. On an HP-UX system, this is accomplished with the command:

```
$ /etc/inetd -c
```

On some systems it is possible to get inetd to re-configure itself by sending it a SIGHUP with the kill(2) command:

```
$ kill -HUP <pid of inetd>
```

On other systems it might be necessary to kill and re-start the inet daemon. At this point, root access is no longer needed and you can proceed to the verification step.

Verifying the bits

To verify the installation of netperf, simply execute the command

```
/opt/netperf/netperf
```

A TCP_STREAM test of 10 seconds duration should be performed over the loopback interface.

Running netserver as a standalone Daemon

If you cannot install netperf as a child of inetd, you can run the netserver as a standalone daemon. Simply execute netserver with the "-p <port number>" option and it will happily start accepting requests on the port number you specify. If you specify a port number other than the normal netperf port number, you should remember to also specify "-p <portnum>" as a global command line option of netperf.

Final Customization

The scripts provided with the netperf distribution are written with the assumption that netperf is installed in */opt/netperf*. If you have decided to install netperf in a different location, you will need to edit each of the script files and alter this line:

NETHOME=/opt/netperf

or one like it, to be something like:

NETHOME=/my/netperf/location

Section 3. The Design of Netperf

Design Basics

Netperf is designed around the basic client−server model. There are two executables − netperf and netserver. Generally you will only execute the netperf program − the netserver program will be invoked by the other system's inetd.

When you execute netperf, the first thing 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 being run, the control connection will be a TCP connection using BSD sockets.

Once the control connection is up and the configuration information has been passed, a separate connection will be opened for the measurement itself using the APIs and protocols appropriate for the test. The test will be performed, and the results will be displayed.

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 system's default, may put packets out on the control connection.

CPU Utilization

CPU utilization is a frequently requested metric of networking performance. Unfortunately, it can also be one of the most difficult metrics to measure accurately. Netperf is designed to use one of several (perhaps platform dependent) CPU utilization measurement schemes. Depending on the CPU utilization measurement technique used, a unique single−letter code will be included in the CPU portion of the test banner for both the local and remote systems.

The default CPU measurement technique is based on the use of "loopers" which will sit in tight little loops consuming any CPU left over by the networking. This method is not without its added overhead, but wherever possible, care has been taken to keep that overhead to a minimum. If you would like to get an estimate of the overhead, run one test with CPU utilization, and one test without, and compare the throughputs. Use of loopers in measuring CPU utilization is indicated by the letter code "L."

NOTE: For accurate CPU utilization on MP systems, it is *crucial* that netperf and netserver know the number of processors on the system. For some systems (HP−UX) this can be determined programmatically. Other systems require the use of the "−n" global command line argument.

HP−UX 10.X offers a zero additional overhead, very accurate CPU utilization mechanism based on the pstat() system call. If you are compiling on HP−UX 10, you should replace the "−DUSE_LOOPER" in the makefile with "−DUSE_PSTAT" and recompile. When this method is being used, the letter code "I" will be displayed.

Other codes may be included in later versions of netperf. When the CPU utilization mechanism is unknown, either a "U" or a "?" will be displayed.

Great care should be exercised when looking at CPU utilization. Be certain you are familiar with the technique being used, and its implications. For example, a mechanism that is based solely on CPU charged to the netperf (netserver) process alone will likely under−report the real CPU utilization SIGNIFICANTLY. Much network processing takes place away from the user process context. Caveat Benchmarker!

Section 4. Using Netperf to measure bulk data transfer performance

The most common use of netperf is measuring bulk data transfer performance. This is also referred to as “stream” or “unidirectional stream” performance. Essentially, these tests will measure how fast one system can send data to another and/or how fast that other system can receive it.

TCP Stream Performance

The TCP stream performance test is the default test type for the netperf program. The simplest test is performed by entering the command:

/opt/netperf/netperf −H *remotehost*

which will perform a 10 second test between the local system and the system identified by *remotehost*. The socket buffers on either end will be sized according to the systems' default and all TCP options (e.g. TCP_NODELAY) will be at their default settings.

To assist in measuring TCP stream performance, two script files are provided with the netperf distribution. They are *tcp_stream_script* and *tcp_range_script*. *Tcp_stream_script* will invoke netperf based on the setting of script variables controlling socket and send sizes. *Tcp_range_script* will perform a similar set of tests, with the difference being that where *tcp_stream_script* tests specific datapoints, *tcp_range_script* will perform tests at points within a specified range.

If you would like to perform tests other than those done by the scripts, you can invoke netperf manually. Some of the options you will likely want to experiment with are:

−s *sizespec*	which will set the local send and receive socket buffer sizes to the value(s) specified. [Default: system default socket buffer sizes]
−S *sizespec*	which behaves just like −s but for the remote system
−m *value*	set the local send size to *value* bytes. [Default: local socket buffer size]
−M *value*	which behaves like −m, setting the receive size for the remote system. [Default: remote receive socket buffer size]
−l *value*	set the test length to *value* seconds when value is > 0 and to $\|value\|$ bytes when value is < 0
−D	set the TCP_NODELAY option to true on both systems

This is not a complete list of options that can affect TCP stream performance, but it does cover those options that are used most often. A complete list of netperf options can be found in Section 7.

XTI TCP Stream Performance

The XTI TCP stream performance test quite similar to the TCP_STREAM test. XTI requires a device file be opened – as the device file is placed in different locations on different systems, it generally must be specified. The simplest XTI TCP stream test on HP–UX is performed by entering the command:

/opt/netperf/netperf –H *remotehost* **–t** XTI_TCP_STREAM **–– –X** /dev/inet_cots

which will perform a 10 second test between the local system and the system identified by *remotehost*. The socket buffers on either end will be sized according to the systems' default and all TCP options (e.g. TCP_NODELAY) will be at their default settings.

The test parameters for an XTI_TCP_STREAM test are the same as for a TCP_STREAM test with the addition of:

–X *devspec* set the local/remote XTI device file name from *devspec*.

UDP Stream Performance

A UDP stream performance test is very similar to a TCP stream test. One difference is that the send size cannot be larger than the smaller of the local and remote socket buffer sizes. What this means is that you must make certain that when you specify the –m option, you use a value that is less than or equal to the socket buffer sizes (–s and –S). Also, since the UDP Stream test is not the default test, the –t *testname* option must be specified, with the testname set to UDP_STREAM. So, a simple UDP stream test command might look something like this:

$ **/opt/netperf/netperf –H** *remotehost* **–t** UDP_STREAM **–– –m** 1024

There is a script provided that performs various UDP stream performance tests. It is called *udp_stream_script*. As with TCP stream performance, you can use the script provided, or perform tests yourself to get datapoints not covered by the script.

NOTE: UDP is an unreliable protocol. It is important that you examine the results carefully as the reported send rate can be much higher than the actual receive rate. Great care should be taken when reporting UDP_STREAM test results to make sure they are not misleading. For example, one should **always** report both send and receive rates **together** for a UDP_STREAM test. If you are going to report a single number, you should report the receive rate.

NOTE: If you would like to "pace" the send rate of the UDP_STREAM test, add a –DINTERVALS to the makefile, do a "make clean" and re–compile. You can then use the –b and –w global options to set the burst size (sends) and wait time (milliseconds) respectively.

XTI UDP Stream Performance

The XTI UDP stream performance test quite similar to the UDP_STREAM test. XTI requires a device file be opened. As the device file is placed in different locations on different systems, it generally must be specified. The simplest XTI UDP stream test on HP–UX is performed by entering the command:

/opt/netperf/netperf –H *remotehost* **–t** XTI_UDP_STREAM **–– –X** /dev/inet_clts

The test parameters for an XTI_UDP_STREAM test are the same as for a UDP_STREAM test with the addition of:

−X *devspec* set the local/remote XTI device file name from *devspec*.

NOTE: UDP is an unreliable protocol. It is important that you examine the results carefully as the reported send rate can be much higher than the actual receive rate. Great care should be taken when reporting XTI_UDP_STREAM test results to make sure they are not misleading. For example, one should **always** report both send and receive rates **together** for a XTI_UDP_STREAM test. If you are going to report a single number, you should report the receive rate.

NOTE: If you would like to "pace" the send rate of the XTI_UDP_STREAM test, add a −DINTERVALS to the makefile, do a "make clean" and re−compile. You can then use the −b and −w global options to set the burst size (sends) and wait time (milliseconds) respectively.

DLPI Connection Oriented Stream Performance

NOTE: DLPI tests are not compiled−in by default with netperf. If you wish to measure performance over DLPI, you will need to add a −DDO_DLPI to the makefile and perhaps add to the "LIBS=" and re−compile netperf and netserver.

A DLPI Connection Oriented Stream test (DLCO_STREAM) looks very similar to a TCP Stream test − they both use reliable, connection oriented protocols. The DLPI test differs from the TCP test in that the message size must always be less than or equal to the local interface's MTU − DLPI does not provide TCP−style segmentation and reassembly.

The simplest DLPI Connection Oriented Stream test would look something like this:

$ **/opt/netperf/netperf −H** *remotehost* **−t** DLCO_STREAM −− **−m** 1024

Here are some of the DLPI−specific command line options:

−D *devspec*	specify the local and/or remote DLPI device file name(s) (fully−qualified). Syntax is the same as that of a *sizespec*.
−m *value*	specify the send size, in bytes, for the local system. This must be less than or equal to the link MTU.
−M *value*	which behaves like −m, setting the receive size for the remote system.
−p *ppaspec*	set the local and/or remote DLPI PPA(s). Syntax is the same as that of a *sizespec*.
−r *value*	specify the request size, in bytes, for the test.
−R *value*	specify the response size, in bytes, for the test.
−s *value*	specify the 802.2 SAP for the test. This should not conflict with any assigned SAP's.
−w *sizespec*	specify the local send/recv window sizes in frames (where available).
−W *sizespec*	specify the remote send/recv window sizes in frames (where available).

DLPI Connectionless Stream

NOTE: DLPI tests are not compiled−in by default with netperf. If you wish to measure performance over DLPI, you will need to add a −DDO_DLPI to the makefile and perhaps add to the "LIBS=" and re−compile netperf and netserver.

A DLPI Connectionless Stream test (DLCL_STREAM) is analogous to a UDP_STREAM test. They both make use of unreliable, connectionless transports. The DLPI test differs from the UDP test in that the message size must always be less than or equal to the link MTU − DLPI does not provide IP−like segmentation and reassembly functionality, and the netperf benchmark does not presume to provide one.

The simplest DLPI Connectionless Stream test command line would look something like this:

$ **/opt/netperf/netperf −H** *remotehost* **−t** DLCL_STREAM −− **−m** 1024

Here are some of the DLPI−specific command line options for the DLCL_STREAM test:

−D *devspec*	specify the local and/or remote DLPI device file name(s) (fully−qualified). Syntax is the same as that of a *sizespec*.
−m *value*	specify the send size, in bytes, for the local system. This must be less than or equal to the link MTU.
−M *value*	which behaves like −m, setting the receive size for the remote system.
−p *ppaspec*	set the local and/or remote DLPI PPA(s). Syntax is the same as that of a *sizespec*.
−s *value*	specify the 802.2 SAP for the test. This should not conflict with any assigned SAP's.
−w *sizespec*	specify the local send/recv window sizes in frames (where available).
−W *sizespec*	specify the remote send/recv window sizes in frames (where available).

Unix Domain Stream Sockets

NOTE: Unix Domain Socket tests are not compiled into netperf by default. If you wish to measure the performance of Unix Domain Sockets, you must recompile netperf and netserver with −DDO_UNIX added to the makefile.

A Unix Domain Stream Socket Stream test (STREAM_STREAM) is very much like a TCP_STREAM test.

The Simplest Unix Domain Stream Socket Stream test command line would look something like this:

$ **/opt/netperf/netperf −t** STREAM_STREAM

The −H global command line Option is not valid for a Unix Domain Socket test and should not be specified.

Here are some of the Unix Domain−specific command line options for the STREAM_STREAM test:

−m *value*	set the local send size to *value* bytes. [Default: local socket buffer size]
−M *value*	which behaves like −m, setting the receive size for the remote system. [Default: remote receive socket buffer size]
−p *dirspec*	set the directory where pipes will be created. [Default: system default for the tempnam() call]
−s *sizespec*	which will set the local send and receive socket buffer sizes to the value(s) specified. [Default: system default socket buffer sizes]
−S *sizespec*	which behaves just like −s but for the remote system

Unix Domain Datagram Sockets

NOTE: Unix Domain Socket tests are not compiled into netperf by default. If you wish to measure the performance of Unix Domain Sockets, you must recompile netperf and netserver with −DDO_UNIX added to the makefile.

A Unix Domain Datagram Socket Stream test (DG_STREAM) is very much like a TCP_STREAM test except that message boundaries are preserved.

The Simplest Unix Domain Datagram Socket Stream test command line would look something like this:

$ **/opt/netperf/netperf −t** DG_STREAM

The −H global command line option is not valid for a Unix Domain Socket test and should not be specified. Here are some of the test specific command line options available in a DG_STREAM test.

−m *value*	set the local send size to *value* bytes. [Default: local socket buffer size]
−M *value*	which behaves like −m, setting the receive size for the remote system. [Default: remote receive socket buffer size]
−p *dirspec*	set the directory where pipes will be created. [Default: system default for the tempnam() call]
−s *sizespec*	which will set the local send and receive socket buffer sizes to the value(s) specified. [Default: system default socket buffer sizes]
−S *sizespec*	which behaves just like −s but for the remote system

Fore ATM API Stream

NOTE: Fore ATM API tests are not compiled into netperf by default. If you wish to measure the performance of connections over the Fore ATM API, you must recompile netperf and netserver with −DDO_FORE added to the makefile.

A Fore ATM API Stream test (FORE_STREAM) is very much like a UDP_STREAM test.

NOTE: The Fore ATM API exports an unreliable protocol. It is important that you examine the results carefully as the reported send rate can be much higher than the actual receive rate. Great care should be taken when reporting FORE_STREAM test results to make sure they are not misleading. For example, one should **always** report both send and receive rates **together** for a FORE_STREAM test. If you are going to report a single number, you should report the receive rate.

The simplest Fore ATM API Stream test command line would look something like this:

$ **/opt/netperf/netperf −t** FORE_STREAM **−H** *remotehost*

Here are some of the test specific command line options applicable to a FORE_STREAM test.

−a *AAL*	use the ATM Adaptation Layer number aal to encapsulate packets. Specifying 3 or 4 will yield AAL3/4, and 5 will yield AAL5. [Default: 5 −> AAL5]
−b *sizespec*	set the mean burst target and/or minimum in units of kilobit packets. The first value is target and the second is minimum. [Default: 0,0]
−d *devspec*	set the name of the ATM device file to be opened. [Default: /dev/atm]
−m *value*	set the local send size to *value* bytes. This must not be larger than the ATM MTU. [Default: ATM MTU]
−M *value*	which behaves like −m, setting the receive size for the remote system. [Default: ATM MTU]
−p *sizespec*	set the peak bandwidth target and/or minimum in units of kilobits/s. The first value is target and the second it minimum. [Default: 0,0 −> network assigned]
−P *sizespec*	set the mean bandwidth target and/or minimum in units of kilobits/s. The first value is target and the second is minimum. [Default: 0,0 −> network assigned]

Section 5. Using Netperf to measure request/response performance

Request/response performance is the second area that can be investigated with netperf. Generally speaking, netperf request/response performance is quoted as "transactions/s" for a given request and response size. A transaction is defined as the exchange of a single request and a single response. From a transaction rate, one can infer one way and round−trip average latency.

TCP Request/Response Performance

The TCP request/response test can be invoked with netperf though the use of the −t option with an argument of TCP_RR. So, a "default" request/response command would look something like this:

$ **/opt/netperf/netperf −H** *remotehost* **−t** TCP_RR

and will use the system default socket buffer sizes, a default request size of 1 byte, and a default response size of 1 byte.

As with the stream performance tests, a script is available to assist you in generating TCP request/response performance numbers. It is called *tcp_rr_script*. However, if you should need to generate numbers at points of you own choosing, these command line options will be of use:

−r	*sizespec*	set the request and/or response sizes based on *sizespec*.
−l	*value*	set the test duration based on *value*. For *value* > 0, test duration will be *value* seconds. Otherwise, test duration will be |*value*| transactions.
−s	*sizespec*	which will set the local send and receive socket buffer sizes to the value(s) specified. [Default: system default socket buffer sizes]
−S	*sizespec*	which behaves just like −s but for the remote system
−D		set the TCP_NODELAY option to true on both systems

The request and response sizes will be the buffer sizes posted to send and receive. The −m and −M options are not meaningful for a TCP_RR test.. As TCP is a stream protocol and not a message protocol, it is necessary to loop on receives until the entire message is delivered. The buffer pointer passed to the first receive for an individual transaction will be aligned and offset as requested by the user. It will be incremented by the number of bytes received each time until the entire request/response is received. The buffer pointer will be re−aligned and offset for the next transaction.

TCP Connect/Request/Response

The TCP_CRR test is a test which mimics the http protocol used by most web servers. Instead of simply measuring the performance of request/response in the same connection, it establishes a new connection for each request/response pair. The test−specific parameters are the same as the TCP_RR test, with one addition:

−p	*max*[,*min*]	set min/max port numbers used by the client side.

It is important that this test run for a reasonable length of time — at least two minutes. This is related to the behavior of various TCP implementations. If you run the test for shorter periods of time, the results could be higher than seen in a steady-state condition. So, a good TCP_CRR command line to simulate a web-server might look like:

$ **/opt/netperf/netperf -t** TCP_CRR **-l** 120 **-H** *remotehost* **-- -r** 32,1024

XTI TCP Request/Response Performance

The XTI TCP request/response test can be invoked with netperf though the use of the -t option with an argument of XTI_TCP_RR. Not all systems put the requisite device files in the same location, so, a "default" request/response command on HP-UX would look something like this:

$ **/opt/netperf/netperf -H** *remotehost* **-t** XTI_TCP_RR **-- -X** /dev/inet_cots

and will use the system default socket buffer sizes, a default request size of 1 byte, and a default response size of 1 byte.

The command-line options for the XTI_TCP_RR test are the same as the TCP_RR test, with the following additions:

-X *devspec* set the local/remote XTI device file name from *devspec*.

The request and response sizes will be the buffer sizes posted to send and receive. As TCP is a stream protocol and not a message protocol, it is necessary to loop on receives until the entire message is delivered. The buffer pointer passed to the first receive for an individual transaction will be aligned and offset as requested by the user. It will be incremented by the number of bytes received each time until the entire request/response is received. The buffer pointer will be re-aligned and offset for the next transaction.

UDP Request/Response Performance

UDP request/response performance works just like TCP request/response performance. All the options available there are present here with the exception of the -D option; TCP_NODELAY has no meaning for a UDP test. To invoke a UDP request/response test, use an argument of UDP_RR with the -t option to produce a command like something like this:

$ **/opt/netperf/netperf -H** *remotehost* **-t** UDP_RR

Again, a script is provided which will generate results for some of the more common datapoints. It is named *udp_rr_script*.

XTI UDP Request/Response Performance

The XTI UDP request/response test can be invoked with netperf though the use of the -t option with an argument of XTI_UDP_RR. Not all systems put the requisite device files in the same location, so, a "default" request/response command on HP-UX would look something like this:

$ **/opt/netperf/netperf -H** *remotehost* **-t** XTI_UDP_RR **-- -X** /dev/inet_clts

and will use the system default socket buffer sizes, a default request size of 1 byte, and a default response size of 1 byte.

The command–line options for the XTI_UDP_RR test are the same as the UDP_RR test, with the following additions:

–X *devspec*	set the local/remote XTI device file name from *devspec*.

The request and response sizes will be the buffer sizes posted to send and receive.

DLPI Connection Oriented Request/Response Performance

NOTE: DLPI tests are not compiled into netperf by default. If you wish to measure the performance of DLPI, you must recompile netperf and netserver with –DDO_DLPI added to the makefile.

A DLPI Connection Oriented Request/Response test (DLCO_RR) looks much the same as any other request/response test. It performs a request/response test over a reliable connection. As with the other DLPI tests, there is no segmentation and reassembly, so all request and/or response sizes must be less than or equal to the link MTU.

A simple DLCO_RR test invocation would look something like this:

$ **/opt/netperf/netperf –H** *remotehost* **–t** DLCO_RR

Here are some of the DLPI–specific command line options:

–D *devspec*	specify the local and/or remote DLPI device file name(s) (fully–qualified). Syntax is the same as that of a *sizespec*.
–p *ppaspec*	set the local and/or remote DLPI PPA(s). Syntax is the same as that of a *sizespec*.
–r *sizespec*	specify the request and/or response sizes, in bytes, for the test.
–s *value*	specify the 802.2 SAP for the test. This should not conflict with any assigned SAP's.
–w *sizespec*	specify the local send/recv window sizes in frames (where available).
–W *sizespec*	specify the remote send/recv window sizes in frames (where available).

DLPI Connectionless Request/Response Performance

NOTE: DLPI tests are not compiled into netperf by default. If you wish to measure the performance of DLPI, you must recompile netperf and netserver with –DDO_DLPI added to the makefile.

A DLPI Connectionless Request/Response test (DLCL_RR) looks much the same as any other request/response test. It performs a request/response test over an unreliable connection. However, netperf does not have any sort of retransmission mechanism, so packet loss with this test will result in dramatically lowered performance results. As with the other DLPI tests, there is no segmentation and reassembly, so all request and/or response sizes must be less than or equal to the link MTU.

A simple DLCL_RR test invocation would look something like this:

$ **/opt/netperf/netperf –H** *remotehost* **–t** DLCL_RR

Here are some of the DLPI–specific command line options:

−D *devspec*	specify the local and/or remote DLPI device file name(s) (fully−qualified). Syntax is the same as that of a *sizespec*.
−p *ppaspec*	set the local and/or remote DLPI PPA(s). Syntax is the same as that of a *sizespec*.
−r *sizespec*	specify the request and/or response sizes, in bytes, for the test.
−s *value*	specify the 802.2 SAP for the test. This should not conflict with any assigned SAP's.
−w *sizespec*	specify the local send/recv window sizes in frames (where available).
−W *sizespec*	specify the remote send/recv window sizes in frames (where available).

Unix Domain Stream Socket Request/Response Performance

NOTE: Unix Domain Socket tests are not compiled into netperf by default. If you wish to measure the performance of Unix Domain Sockets, you must recompile netperf and netserver with −DDO_UNIX added to the makefile.

A Unix Domain Stream Socket Request/Response test (STREAM_RR) is very much like a TCP_RR test.

The STREAM_RR test command line would look something like this:

$ **/opt/netperf/netperf −t** STREAM_RR

The −H global command line option is not valid for a Unix Domain Socket test and should not be specified.

Here are some of the Unix Domain−specific command line options for the STREAM_STREAM test:

−p *dirspec*	set the directory where pipes will be created. [Default: system default for the tempnam() call]
−r *sizespec*	which will set the request and response sizes to the value(s) specified. [Default: 1 byte]

Unix Domain Datagram Socket Request/Response Performance

NOTE: Unix Domain Socket tests are not compiled into netperf by default. If you wish to measure the performance of Unix Domain Sockets, you must recompile netperf and netserver with −DDO_UNIX added to the makefile.

The Simplest Unix Domain Datagram Socket Request/Response (DG_RR) test command line would look something like this:

$ **/opt/netperf/netperf −t** DG_STREAM

The −H global command line option is not valid for a Unix Domain Socket test and should not be specified. Here are some of the test specific command line options available in a DG_STREAM test.

−p *dirspec* set the directory where pipes will be created. [Default: system default for the tempnam() call]

−r *sizespec* set the request and/or response sizes to the value(s) specified. [Default: 1 byte]

Fore ATM API Request/Response Performance

NOTE: Fore ATM API tests are not compiled into netperf by default. If you wish to measure the performance of connections over the Fore ATM API, you must recompile netperf and netserver with −DDO_FORE added to the makefile.

A Fore ATM API Request/Response test (FORE_RR) is very much like a UDP_RR test.

The simplest FORE_RR test command line would look something like this:

$ **/opt/netperf/netperf −t** FORE_RR **−H** *remotehost*

Here are some of the test specific command line options applicable to a FORE_STREAM test.

−a *aal* use the ATM Adaptation Layer number aal to encapsulate packets. Specifying 3 or 4 will yield AAL3/4, and 5 will yield AAL5. [Default: 5 −> AAL5]

−b *sizespec* set the mean burst target and/or minimum in units of kilobit packets. The first value is target and the second is minimum. [Default: 0,0]

−d *devspec* set the name of the ATM device file to be opened. [Default: /dev/atm]

−p *sizespec* set the peak bandwidth target and/or minimum in units of kilobits/s. The first value is target and the second it minimum. [Default: 0,0 −> network assigned]

−P *sizespec* set the mean bandwidth target and/or minimum in units of kilobits/s. The first value is target and the second is minimum. [Default: 0,0 −> network assigned]

−r *sizespec* set the request and/or response sizes to the values specified [Default: 1 byte]

Section 6. Other Netperf tests

Apart from the usual performance tests, netperf contains some tests that can be used to streamline measurements. These tests range from CPU rate calibration (present) to host identification (future enhancement).

CPU rate calibration

NOTE: Previous revisions of the manual may have discussed the HP kernel idle counter. With the release of HP−UX 10.0, this mechanism has been replaced with an equally accurate one based on information returned from the pstat() system call. This accurate pstat−based mechanism is *not* available in HP−UX 9.X, nor in non−HP operating systems. For those systems, a CPU utilization measurement based on the use of "loopers" is employed. Each requires calibration.

In the context of netperf, a CPU rate is expressed not in clock frequencies, MIPS or MFLOPS, but simply how fast the system can count. There are two CPU rate calibrations tests. The first measures and displays the CPU rate for the local system. It is called LOC_CPU. The second test, REM_CPU, is exactly the same, except that it works on the system specified with the −H command line option.

In and of themselves, these two tests are only arcanely interesting. However, they can be used to greatly speed−up test scripts. Remember that for CPU measurements, it is necessary to "calibrate" the CPU or determine how fast it can count. This process takes at least forty (40) seconds for the local system and forty (40) seconds for the remote system. One can save the results of the CPU tests in shell variables and then use them as arguments to the −c and −C command line options. Passing−in a rate with the −c or −C option tells netperf that you already know the CPU rate, so it can skip the calibration steps. For example, the following Unix shell fragment will determine the local CPU rate and use that for subsequent tests:

$ LOC_RATE=‘**/opt/netperf/netperf −t** LOC_CPU‘

$ **/opt/netperf/netperf −H** *somehost* **−c** $LOC_RATE

You should remember that CPU rates will vary from system to system. Generally, the best trade−off between time and accuracy is to perform the calibrations once in each script or session. The default scripts provided will use the LOC_CPU and REM_CPU tests to reduce the time overhead of CPU calibration.

NOTE: For accurate CPU utilization on MP systems, it is *crucial* that netperf and netserver know the number of processors on the system. For some systems (HP−UX) this can be determined programmatically. Other systems require the use of the "−n" global command line argument. The authors are always looking for supported calls to find the number of active processors in a system.

Section 7. Netperf Command-line Options Reference

This section describes each of the global command-line options available in the netperf program. Essentially, it is an expanded version of the usage information displayed by netperf when invoked with the -h option in global command line option area.

Command-line Options Syntax

Revision 1.8 of netperf introduced enough new functionality to overrun the English alphabet for mnemonic command line option names. For this reason, command-line options were split in Revision 1.8. This split remains in Revision 1.9alpha. There are two types of netperf command-line options. They are "global" and "test-specific." Both types are entered on the same command line, but they must be separated by a "--" for correct parsing. Global command line options come first, followed by test-specific. If only test-specific options are to be specified, they must be preceded by "--" or the results will be undefined.

Global Options

-a sizespec
: This option allows you to alter the send and receive buffer alignments on the local system. Changing the alignment of the buffers can force the system to use different copying schemes, which can have a measurable impact on performance. If the page size for the system was 4096 bytes, and you wanted to pass page aligned buffers beginning on page boundaries, you could use "-a 4096". The units for this option are whole bytes. [Default: 8 bytes]

-A sizespec
: This option is identical to the -a option with the exception that the alignments are altered for the remote system.

-b size
: This option (-DINTERVALS compilation only) sets the size of a burst of packets in a _STREAM test. This can be used to "pace" the send rate when there is no flow-control provided by the protocol being measured.

-c [rate]
: This option will request CPU utilization and service demand calculations for the local system. If the optional rate parameter is specified, netperf will use that instead of calculating the rate itself. For more information on CPU utilization measurements with netperf, please consult Section 3. [Default: no CPU measurements]

-C [rate]
: This option is identical to the -c option with the exception that it requests CPU utilization for the remote system.

-d
: This option will increase the quantity of debugging output displayed during a test. If debugging is set high enough, it may have a measurable impact on performance. Debugging information for the local system (the one running netperf) is printed to stdout. Debugging information for the remote system (the one running netserver) is sent to the file /tmp/netperf.debug [Default: no debugging]

−f	GMKgmk	This option can be used to change the units of measure for stream tests. The "G", "M", and "K" arguments will set the output units to 2^{30}, 2^{20}, and 2^{10} bytes/s respectively. The "g", "m", and "k" arguments will set the output units to 10^9, 10^6, and 10^3 bits/s respectively. [Default: m − 10^6 bits/s)]
−h		This option causes netperf to display its usage string and exit.
−H	remotehost	This option sets the name of the remote system. It can be specified as either a hostname (e.g. foo.bar.baz) or an IP address (e.g. 1.2.3.4). [Default: localhost]
−l	testlen	With this option you can control the length of the test. If you specify a positive value for testlen, the test will run for that many seconds. If you specify a negative value, the test will run for that many transactions for a request/response test, or that many bytes for a stream test. Some tests can only be timed. [Default: 10 seconds]
−n	value	The value passed−in will be used as the number of CPU's in the system for the purposes of CPU utilization. This is required for MP systems where netperf cannot determine the number of processors programmatically. It is not needed on HP−UX 10.X. [Default: 1 processor]
−o	sizespec	The value passed with this option will be used as an offset from the alignment specified with the −a option. With this option you could, for example, pass buffers to the system that began 3 bytes after the beginning of a 4KB page (−a 4096 −o 3) [Default: 0 bytes]
−O	sizespec	This option behaves just like the −o option but on the remote system. It works in conjunction with the −A option. [Default: 0 bytes]
−p	portnum	You should use this option when the netserver program will be waiting at a port other than the default. This might be the case if you run netserver as a standalone process rather than a child of inetd.
−P	0|1	If you do not want the test banner to be displayed, then use this option with an argument of 0. An situation where this might be useful would be where you repeat the same test configuration several times and do not want the banners cluttering things up. [Default: 1 − display test banners]

−t testname
: You should use this option to specify the test you wish to perform. As of this writing, the valid testnames are TCP_STREAM, TCP_RR, UDP_STREAM, UDP_RR, DLCO_STREAM, DLCO_RR, DLCL_STREAM, DLCL_RR, STREAM_STREAM, STREAM_RR, DG_STREAM, DG_RR, FORE_STREAM, FORE_RR, HIPPI_STREAM, HIPPI_RR LOC_CPU, and REM_CPU. [Default: TCP_STREAM]

−v verbosity
: This option can be used to set the verbosity level for the test. It can be used in conjunction with the −P option. If the verbosity is set to 0, then only the result of the test will be displayed. If CPU utilization numbers are requested for the local system, the result of the test will be local service demand. If remote CPU utilization is requested, then the result will be remote service demand. Otherwise, the result will be the measured thruput.

−V
: This option will attempt to enable the copy−avoidance features of HP−UX 9.0 networking. [Default: no copy−avoidance attempted]

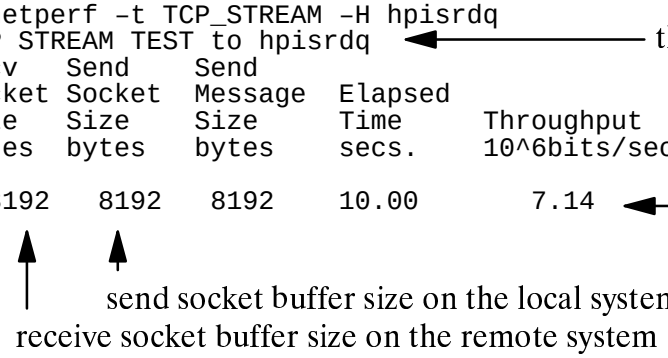
−w time
: This option (−DINTERVALS compilation only) will set the inter−burst time to time milliseconds. The actual wait time may differ depending on the resolution of timers on the system being measured.

Section 8. Netperf examples

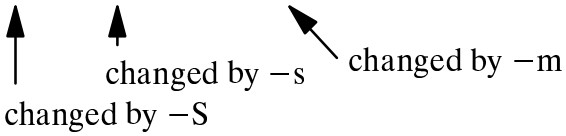
NOTE: These examples are from an older version of netperf and do not represent the split between global and test specific command line options.

The next few pages contain annotated screen dumps of example netperf runs. After examining these examples, you should have a fairly good idea of how to read the output of netperf and what effect certain options have on that output. First, TCP_STREAM tests.

```
$ netperf -t TCP_STREAM -H hpisrdq
TCP STREAM TEST to hpisrdq
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

  8192   8192   8192    10.00       7.14
```

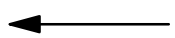



```
$ netperf -t TCP_STREAM -H hpisrdq -- -s 16384 -S 16K -m 1K
TCP STREAM TEST to hpisrdq
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 16384  16384   1024    10.01       7.32
```




```
$ netperf -t TCP_STREAM -H hpisrdq -P 0
  8192   8192   8192    10.00       8.07
```


```
$ netperf -t TCP_STREAM -H hpisrdq -P 0 -v 0
   7.05
```



This next set of examples is taken from some UDP_STREAM tests. You should notice right away that the output format is slightly different as there can be UDP sends that "fail" without there being a connection loss. Also, since every UDP datagram sent is not always "sent" or received, there are more statistics displayed – one line for local statistics and a second line for remote statistics.

```
$ ./netperf -t UDP_STREAM
UDP UNIDIRECTIONAL SEND TEST to localhost
Socket  Message  Elapsed      Messages
Size    Size     Time         Okay Errors   Throughput
bytes   bytes    secs            #      #   10^6bits/sec

  9216    9216   9.99        10500      0      77.47
  9360           9.99        10314             76.10
```

test type and dest

message size

successful calls to send

send performance

```
$ ./netperf -t UDP_STREAM -f K
UDP UNIDIRECTIONAL SEND TEST to localhost
Socket  Message  Elapsed      Messages
Size    Size     Time         Okay Errors   Throughput
bytes   bytes    secs            #      #   KBytes/sec

  9216    9216   10.00        9822      0    8839.06
  9360           10.00        9534           8579.88
```

receive perf

receive socket size on remote

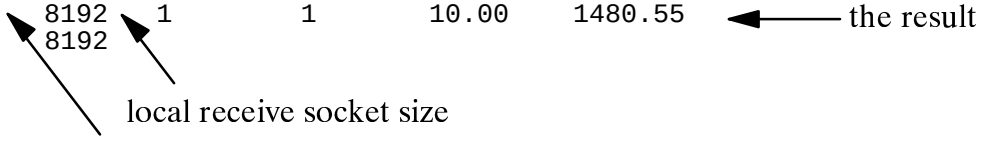
successful calls to recv

```
$ ./netperf -t UDP_STREAM -H hpindio -- -m 1472
UDP UNIDIRECTIONAL SEND TEST to hpindio
Socket  Message  Elapsed      Messages
Size    Size     Time         Okay Errors   Throughput
bytes   bytes    secs            #      #   10^6bits/sec

  9216    1472   10.00        7634  45525       8.99
  9360           10.00        7572              8.92
```

failed send calls (ENOBUFS)

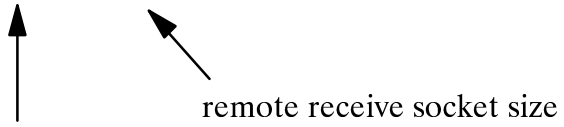
This third set of examples is taken from some *_RR tests. These tests use a two–line results format much like that of the UDP_STREAM tests. The only exception is that there are no provisions made for displaying lost packets, as there are not to be any.

```
$ ./netperf -t TCP_RR
TCP REQUEST/RESPONSE TEST to localhost
Local /Remote
Socket Size   Request  Resp.   Elapsed  Trans.
Send   Recv   Size     Size    Time     Rate
bytes  Bytes  bytes    bytes   secs.    per sec

8192   8192   1        1       10.00    1480.55
8192   8192
```



use 1024 byte requests and 256 bytes responses

```
$ ./netperf -t UDP_RR -- -r 1024,256
UDP REQUEST/RESPONSE TEST to localhost
Local /Remote
Socket Size   Request  Resp.   Elapsed  Trans.
Send   Recv   Size     Size    Time     Rate
bytes  Bytes  bytes    bytes   secs.    per sec

9216   9360   1024     256     10.00    1421.35
9216   9360
```



If you have compiled netperf with –DHISTOGRAM, it is possible to get a histogram of individual response times for a request/response test. To enable the histogram code, you need to specify a verbosity level of two or more with the "–v" global command.

```
$ ./netperf –l 60 –t TCP_RR –H hpindio –v 2
TCP REQUEST/RESPONSE TEST to hpindio : histogram
Local /Remote
Socket Size   Request  Resp.   Elapsed  Trans.
Send   Recv   Size     Size    Time     Rate
bytes  Bytes  bytes    bytes   secs.    per sec

8192   8192   1        1       60.00     779.55
8192   8192
Alignment      Offset
Local  Remote  Local  Remote
Send   Recv    Send   Recv
    8      0       0      0
Histogram of request/response times
TENTH_MSEC    :    0:    0:    0:    0:    0:    0:    0:    0:   0:    0
UNIT_MSEC     :    0: 45771:  582:  141:   42:   48:   48:   38:   24:   10
TEN_MSEC      :    0:   51:    8:    1:    1:    0:    0:    1:    1:    0
HUNDRED_MSEC  :    0:    2:    0:    0:    0:    1:    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:      46770
```

Section 9. Changes and Fixes in this Release of Netperf

Revision 2.1 of netperf contains the following changes and fixes:

- Netperf now supports revision 3.51 of the Windows NT operating system (TCP and UDP socket tests only). The same code/binaries may actually run under Windows95, but that is untested as of this writing.
- The XTI test suite is now complete and supported (at least on HP−UX). This is for XTI Version 2. (−DDO_XTI compilation only)
- Fixes that can affect −DUSE_LOOPER CPU utilization.
- Support for TCP and UDP over IPv6 (IPng) using the BSD sockets interface (−DDO_IPV6 compilation only)
- Compiler support for a "long long" data type is only required for −DUSE_PSTAT compilation on HP−UX.
- The annotated examples in the manual are no longer implemented with screen dumps, making the manual *much* smaller.

Section 10. Known Bugs and Misfeatures

Revision 2.1 of netperf contains the following known bugs and misfeatures:

- Verbose Output – the logic for greater than standard verbosity is flawed in all tests except the TCP_STREAM test. The verbose output for the TCP_STREAM test could use some formatting help. When confidence intervals are requested, the verbose output will likely be flawed.
- On some systems, the UDP_RR test will fail with with the message "send_udp_rr: data recv error: Connection Refused." It is unknown if this is a bug in those systems for connected UDP sockets, or a bug in netperf.
- Some test suites support for neither histograms nor confidence intervals.
- The benchmark is not written to ANSI C. However, an ANSI C compiler is required for –DHISTOGRAM compilation. Future versions of the benchmark may require ANSI C for all modes of compilation. –DUSE_PSTAT compilation on HP–UX requires support for the "long long" data type.
- The manual does not show examples for tests other than UDP or TCP.
- The DLPI Tests are not fully debugged for multivendor or non–Ethernet environments.
- The Fore API Tests need better headers.
- The errors reported for remote netperf errors in a Fore API test will almost certainly be wrong as netperf does not distinguish between an atm_errno and a standard errno.
- The manual needs additional troubleshooting examples.
- CPU utilization measurements in Win32 (–DUSE_LOOPER) need calibration.

If you are feeling adventurous, fixes for these problems would be greatly appreciated ;–)

Section 11. Troubleshooting

Netperf is a rather simple tool, so its error conditions will be relatively simple and few in number. This section will cover some of the more common error messages and situations you might see and offers advice as to what might be wrong and how to correct it. The error messages displayed on non−HP−UX systems might not be the same.

establish_control: control socket connect failed: Connection refused

When this message is displayed, it can mean one of the following:

- Netperf has not been installed on the remote system, or you forgot to run the netserver program explicitly. Either install netperf following the instructions at the beginning of this document, or run netserver explicitly on the remote system.
- You made a typo in the files */etc/services*, or */etc/inetd.conf*. Check your entries against those in the installation section. If changes are needed, make them and then reconfigure inetd as described in the installation section.
- You forgot to reconfigure inetd. Reconfigure inetd following the instructions in the installation section.
- You specified a port number with the −p option to netperf that is not the same as the port number being used by the netserver program on the remote system.
- Inetd security on the remote system is not configured to allow connections from the local system. Follow the instructions in the manpage for inetd.sec.
- Netperf hates you. It has always hated you. Consult the job listings in misc.jobs.offered and start your life over :)

udp_send: data send error: Message too long
−or−
send_udp_rr: data send error: Message too long

These messages indicate the following:

- You have requested a send size (−m option) or request size (−r option) that is larger than the local socket send buffer size (−s option) with a UDP_STREAM or UDP_RR test (−t option). You should either increase the size of the socket buffer, or decrease the size of the send/request.

put_control: acknowledgement error wanted 6 got 5
netperf: dl_open: could not sent control message, errno = 0
netperf: send_dlpi_co_stream: dlpi_stream data descriptor: Error 0

This stream of messages indicates the following:

- You have specified a DLPI PPA that is not valid. Verify the correct PPA and try again. Variations on this stream of messages can be seen for each of the DLPI tests.

netperf: dl_open: open of /dev/foo failed, errno = 2
netperf: send_dlpi_co_stream: dlpi stream data descriptor: No such file or directory

This stream of messages indicates the following:

- You have specified a DLPI device file that is not valid. Verify the device file name and try again. Defaults for DLPI device files will vary from OS to OS. Variations on this stream of messages can be seen for each of the DLPI tests.

netperf: receive_response: no response received

This message indicates that netperf was expecting to receive a response on its control connection, but none was received within the timeout period. In some cases, this could be the result of the operating system restarting system calls that netperf/netserver expects to exit with an errno of EINTR.

Of course, there may be other problems that are not covered by this section. For further support, you can direct email questions to the author:

raj@cup.hp.com

There is also an Internet mailing list devoted to the discussion of netperf. It is called netperf–talk and is hosted on netperf.cup.hp.com. To subscribe, send an email message to netperf–talk–request@netperf.cup.hp.com. Please do *not* send subscription requests to netperf–talk itself.

Alternatively, the authors of netperf do try to read the Internet newsgroups comp.sys.hp.*, comp.protocols.tcp–ip, comp.benchmarks and the HP internal newsgroups pertaining to networking and performance. As likely as not, they will also see posts about netperf in other groups as well.

Section 12. Glossary

DLPI	**D**ata **L**ink **P**rovider **I**nterface. This is a standardized Link–Level Access (Level 2 of the OSI Reference Model) API. This interface is discussed in many books on Streams programming. The DLPI Version 2 Reference can be retrieved via anonymous FTP from col.hp.com.
TCP	**T**ransmission **C**ontrol **P**rotocol. This defines a reliable, byte–stream protocol for exchanging information between two hosts.
UDP	**U**ser **D**atagram **P**rotocol. This defines an unreliable, message–oriented protocol for exchanging information between two hosts.
TLI	**T**ransport **L**ayer **I**nterface. This is a standardized Transport level (Level 4 of the OSI Reference Model) API. This interface can be used in conjunction with many different transport protocols, including TCP, and the OSI Transports (TP0 through TP4). Often found in Streams implementations.
XTI	X/Open Transport Interface. This is a TLI–like (very TLI–like actually) Transport level API. The definition of XTI is maintained by X/Open.
IP	**I**nternet **P**rotocol. This protocol is the "glue" between TCP/UDP and the Link–Level. It provides the services of routing and packet segmentation and reassembly to export the illusion of a single homogenous network to the higher protocols.

Section 13. The Netperf Database

An online database of netperf results is available to anyone with a forms−capable WWW browser and access to the Internet. The URL to follow is:

http://www.cup.hp.com/netperf/NetperfPage.html

From there you can search or browse the database, or better still, submit numbers! There are additional links which will allow you to download copies of the benchmark or find other sources of network performance information.

Section 14. Netperf Futures

The netperf benchmark is becoming very large. In fact, it is almost large enough to be untenable – it certainly appears daunting to anyone wishing to port it to new platforms. So, unless overwhelming response is received in support of the following test suites, revision 2.1 will be the last release of netperf with the following tests:

- Unix Domain Sockets
- HP HiPPI LLA

Further, the following are also mildly under consideration for removal in the future:

- Fore ATM API

A good way to show overwhelming support for any of the test suites listed above would be to submit results using those suites to the netperf results database.

Appendix A. Netperf on NT

Netperf is not run quite the same on NT as it is under Unix. These differences stem from a (perceived) lack of certain services of Unix − namely the inetd. The lack of inetd support means that netserver must be run explicitly for each invocation of netperf. One simple mechanism to save time is to use the "for" functionality of the MS−DOS command shell:

$ **for %i in (1 2 3 ... N) do netserver −p 12865**

which will run N netservers in succession, one after the other. You can then run N successive instances of netperf on other systems before having to re−run netserver manually. If you are using scripts, you may wish to introduce some delays between individual invocations of netperf. Experimentation has shown that NT may not be able to get the next netserver running before the next netperf comes along. A pause of two seconds or so should be sufficient.

If you wish to run concurrent instances of netperf, it is necessary to combine the "for" and "start" commands of the MS−DOS command shell. On the system running netserver you would enter;

$ **for %i in (12865 12866 ... 12865+N) do start /B netserver −p %i**

which will run N parallel instances of the netserver, each waiting at their own port number. The port numbers you use are up to you, but they must match those you use in the invocation of netperf. For example, if you also wanted to run N parallel netperfs (perhaps to measure aggregate loopback performance) you would execute the following in another MS−DOS command shell:

$ **for %i in (12865 12866 ... 12865+N) do start /B netperf −p %i −P 0 −v 0 −l 240**

and then manually tabulate the results − after being certain that all tests finished closely to one another.

None of the scripts provided with netperf have been ported to MS−DOS command files for the 2.1 release. Any assistance you might be able to provide in that area would be greatly appreciated by the authors.