Nettimer Manual

Kevin Lai

laik@cs.stanford.edu

This is a manual for the nettimer program. Nettimer is a tool for measuring network bandwidth. It can measure all the link bandwidths along a path or just the bottleneck link. It can measuring existing traffic passively or do active probing. It can measure at one host or at multiple hosts in concert. In this manual I describe the purpose, functionality, installation, parameters, and output of the nettimer program.

Table of Contents

1. Introduction

Nettimer is a program for measuring network link bandwidth. Link bandwidth is the maximum number of bits that a link can transfer per unit time. For example, an Ethernet link usually has a bandwidth of 10Mb/s or 100Mb/s, while a modem link has a bandwidth of 33.6Kb/s or 56Kb/s. Nettimer is a tool for measuring network bandwidth in the same way that ping is a tool for measuring network round trip time.

Knowing the bandwidths of links allows systems and people to be much smarter about the decisions that they make. Some examples of applications:

- A user can determine if that new network equipment/service is really delivering the claimed bandwidth.
- A client can select a higher bandwidth proxy/replicated server over a lower bandwidth proxy/replica.
- A server can adapt to different client bandwidths by scaling the size and quality of its content.
- A multi-homed host can route traffic through the highest bandwidth available interface.

Nettimer is flexible enough that you vary several parameters in how you want it to run: 1) you can measure all link bandwidths along a path or just the bottleneck link bandwidth, 2) you can measure passively or actively, 3) you can measure at 1 or more hosts, and 4) you take measurements from traces or measure in real time.

Nettimer can measure the bandwidths of all the links along a path or just the bandwidth of the minimum bandwidth link along a path (the bottleneck link bandwidth). The disadvantage of measuring all the link bandwidths is that it is much slower and more intrusive than measuring just the bottleneck link bandwidth.

When measuring just the bottleneck link bandwidth along a path, nettimer can either 1) passively listen to other applications' traffic for measurement, or 2) generate its own probe traffic. Passive measurement doesn't put additional load on the network. Active measurement is useful when there is no other traffic to measure. When measuring all the link bandwidths along a path, nettimer must use active probing.

In addition, nettimer can take measurements at one or more hosts. Measuring at more hosts allows accurate results at the cost of more effort to deploy software.

Finally, when measuring the bottleneck link bandwidth, nettimer can read tcpdump traces or measure in real time. The advantage of taking traces is that you can reproduce the bandwidth calculation at some later point for debugging or experimentation. The advantage of measuring in real time is that applications can get the results faster.

The requirement for running nettimer is root access on a Linux >= 2.2.0 host. In addition, the development, testing, and packaging was done on a Redhat 7.1 machines, so it may be more difficult to install on other distributions.

1.1. More Information

For more (and the freshest) information, see the Nettimer (http://mosquitonet.stanford.edu/~laik/projects/nettimer) web page. In addition, we wrote several academic papers on the theory and experiments behind nettimer:

- Kevin Lai and Mary Baker, "Measuring Bandwidth", Proceedings of IEEE INFOCOM '99, March 1999. [html] (http://mosquitonet.stanford.edu/~laik/projects/nettimer/publications/infocom1999/html/index.html) [ps.gz] (http://mosquitonet.stanford.edu/~laik/projects/nettimer/publications/infocom1999/nettimer.ps.gz)
- Kevin Lai and Mary Baker, "Measuring Link Bandwidths Using a Deterministic Model of Packet Delay", Proceedings of ACM SIGCOMM 2000, August 2000. [pdf] (http://mosquitonet.stanford.edu/~laik/projects/nettimer/publications/sigcomm2000/deterministic.pdf)
- Kevin Lai and Mary Baker, "Nettimer: A Tool for Measuring Bottleneck Link Bandwidth", Proceedings of the USENIX Symposium on Internet Technologies and Systems, March 2001. [pdf] (http://mosquitonet.stanford.edu/~laik/projects/nettimer/publications/usits2001/a_tool_for_measuring_bottleneck_link_bandwidth.pd [ps] (http://mosquitonet.stanford.edu/~laik/projects/nettimer/publications/usits2001/a_tool_for_measuring_bottleneck_link_bandwidth.ps [html] (http://mosquitonet.stanford.edu/~laik/projects/nettimer/publications/usits2001/index.html)

1.2. Conventions

This section shows the formatting conventions used in this manual.

Standard text.

A command that a user types at a prompt:

```
nettimer --run_dpcap_server
```

A required argument for a command:

nettimer {—run_dpcap_server}

An optional argument for a command:

nettimer [—version]

A parameter that is supplied by the user:

nettimer [—dpcap_servers *servers*]

1.3. New Versions

Versions

Date : 7/23/2001
Changes : Document active probing functionality

Date : 3/6/2001
Changes : Did Bottleneck Bandwidth Log Results section

Date : 2/13/2001
Changes : Describes nettimer version 2.2.0

Date : 11/10/2000
Changes : Initial Version. Describes nettimer version 2.0.1

1.4. To Do

To Do

Date : 10/25/2000
Task : Do Link Bandwidth Measurement section

Date : 10/26/2000
Task : Section on Security

Date : 10/26/2000
Task : Finish client calculation options

1.5. Feedback

Please send your comments, suggestions, and criticisms to the following email address : <laik@cs.stanford.edu>.

1.6. Credits

The following people were very helpful with the research behind nettimer: Mary Baker, Mema Roussopoulos, T.J. Giuli, Vern Paxson.

The following people have given invaluable feedback on nettimer: David G. Anderson, Mary Baker, Petros Maniatis, Robert Morris, Stefan Saroiu, Ed Swierk, Xinhua Zhao.

1.7. Copyright Information

1.8. Disclaimer

No liability for the contents of this document can be accepted. Use the concepts, examples and other content at your own risk. As this is a new edition of this document, there may be errors and inaccuracies, that may of course be damaging to your system. Proceed with caution, and although this is highly unlikely, the author(s) do not take any responsibility for that.

All copyrights are held by their by their respective owners, unless specifically noted otherwise. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark.

Naming of particular products or brands should not be seen as endorsements.

You are strongly recommended to take a backup of your system before major installation and backups at regular intervals.

2. Quick Start

This section describes how to get started quickly.

First, install nettimer as described in Section 3.

This example only covers the common case of actively measuring the bottleneck link:

```
nettimer --active_probing_addresses remote_host
```

where *remote_host* is the name or address of the remote host that you want to measure to.

The output looks something like:

```
BW: 7995536.719157 Error: 0.554183 Samples: 12
Active probing statistics:
  packets sent: 16
  packets received: 16
  timeouts: 0
KLException caught in program: "nettimer"
 at file: "nettimer.c" function: "main" line: XXX
  Passed at file: "packet_capture.c" function: "packet_capture_event_handler" line: XXX
    meaning: "Client exception."
  Passed at file: "dpcap_flow.c" function: "flow_set_new_packet" line: XXX
    meaning: "client exception"
  Thrown at file: "active_probing.c" function: "active_probing_packet_new" line: XXX
    meaning: "Exceeded active probing sent packets maximum."
nettimer exitting
```

The first line of the output gives the bandwidth estimate, error estimate, and the number of samples used to generate the estimate.

3. Installation

This section describes how to install nettimer.

You can install a dynamically linked or a statically linked version of nettimer. The dynamically linked version saves some disk space and can be installed/uninstalled more easily because it uses RPM, but it only works on Redhat >= 7.0 systems. The statically linked version works on any Linux system with a 2.2.x kernel, but it consumes more disk space, and may not install/uninstall cleanly. The other option is to compile it yourself, but this isn't recommended unless you know what you're doing.

3.1. Installing the Dynamically Linked Version

The procedure is to download the required RPM packages and install them. For i386 machines, the required standard packages:

- blas-3.0 (ftp://speakeasy.rpmfind.net/linux/redhat/7.1/en/os/i386/RedHat/RPMS/blas-3.0-9.i386.rpm): This library provides the basic linear algebra routines which are necessary for some of nettimer's statistical code.
- lapack-3.0 (ftp://speakeasy.rpmfind.net/linux/redhat/7.1/en/os/i386/RedHat/RPMS/lapack-3.0-9.i386.rpm): This library provides some matrix manipulation routines which are necessary for some of nettimer's statistical code.

Download the latest versions of the nettimer specific packages from the nettimer home page (http://mosquitonet.stanford.edu/~laik/projects/nettimer/index.html):

- libkl: This library provides statistical and other utility functionality.
- libdpcap: This library provides the distributed packet capture functionality that nettimer uses to capture packets at multiple hosts in concert.
- nettimer: This package contains the main nettimer application.

These are optional packages:

- libpcap-0.6.2 (ftp://speakeasy.rpmfind.net/linux/rawhide/1.0/i386/RedHat/RPMS/libpcap-0.6.2-7.i386.rpm): This library provides the portable packet capture functionality which is necessary for nettimer to do live packet capture and read tcpdump traces. You only need this library if you are compiling nettimer yourself.
- tcpdump (http://mosquitonet.stanford.edu/~laik/projects/nettimer/tcpdump-3.4-10.i386.rpm): This application allows you to take traces which can later be analyzed by nettimer off-line.
- dmalloc (ftp://rpmfind.net/linux/sourceforge/Dmalloc/dmalloc-4.7.1-1.i386.rpm): This library allows you to debug the memory allocations of nettimer. You only need this package if you want to compile and debug nettimer yourself.

Once you have all the packages, become root and use rpm to install them:

```
rpm -U *.rpm
```

These packages are compiled for Intel 386 architecture machines. You can recompile them for Redhat hosts of other architectures by downloading the appropriate SRPM package and then building the binary package (see the RPM documentation). I have not tested this, so there may be byte-ordering or other bugs.

You may be able to use later versions of these packages without any problems. However, later versions of the libpcap library and tcpdump application that Redhat provides use an incompatible trace file format. If you gather traces using the new format, you cannot use older versions of tcpdump and copies of nettimer that are linked with older versions of libpcap to read those traces.

3.2. Installing Statically Linked

Installing statically linked is recommended for RedHat 6.x, Debian, SuSE, Slackware, other Linux kernel 2.2.x systems. You must have root access on the machine you are installing on.

To install, get the statically linked binary from the nettimer home page (http://mosquitonet.stanford.edu/~laik/projects/nettimer/index.html). Un-tar and un-zip. Copy the nettimer binary "nettimer/src/nettimer" wherever you like, and if you want non-root users to be able to use it, make root own it and set the setuid bit:

```
cp nettimer/src/nettimer /usr/local/sbin
chown root /usr/local/sbin/nettimer
chmod +s /usr/local/sbin/nettimer
```

3.3. Compiling Nettimer Yourself

For installing on non-Linux hosts, you are pretty much on your own. I give some tips in this section, but you have to know what you are doing and there are no guarantees. The general procedure is for each of the packages mentioned above, download the source code, un-tar and un-zip, run configure, compile, and install.

First, you must have installed the external packages (blas, lapack, libpcap) on your system and downloaded the nettimer specific source archives. Once you have the archives, un-tar and un-zip them:

```
tar -xzvf *.tar.gz
```

Go into each of the directories and run the configure program, specifying where to find the other components, compile, and install:

```
cd libkl-x1.y1.z1
./configure
make
make install
cd ../libdpcap-x2.y2.z3
./configure --with-libkl="../libkl-x1.y1.z1"
make
make install
cd ../nettimer-x.y.z
./configure --with-libkl="../libkl-x1.y1.z1"
--with-libdpcap="../libdpcap-x2.y2.z2"
make
make install
```

For Windows NT and 2000, you have to do all of the above and find and fix the code that depends on non-POSIX UNIX-specific functionality.

For everything else, and also for the above systems where porting is too much trouble, you can run nettimer on a Redhat machine and use the results on another machine.

4. Measuring the Bottleneck Link Bandwidth Actively

This section describes how to actively measure the bottleneck link bandwidth along a path.

In this section I describe how to run nettimer for actively probing the bottleneck link bandwidth. This is useful if you don’t have any existing traffic that you want to measure.

However, the active probing mode has two limitations: 1) it can’t measure asymmetric links, and 2) it puts a possibly significant load on the network.

The active probing mode currently cannot measure asymmetric links because it can only measure the bandwidth from the measurement to host to another host. This is because it’s difficult to cause arbitrary Internet hosts to respond with two packets back-to-back. Consequently, you can only measure the bandwidth going in one direction of an asymmetric link unless you can run nettimer on both sides of the link (but not necessarily adjacent to it).

The active probing mode puts a possibly significant load on the network. In general, you should remember that the owner of the remote host has better things to do with his/her network bandwidth than carry your measurement traffic. Therefore, you should try to make sure that the measurement traffic isn't putting a noticeable burden on anyone's network. Otherwise, someone is going to think you are lauching a denial-of-service attack against them.

Using the default parameters given below, the traffic that the active mode generates is at most 64,000 bits. This is only a problem for very low bandwidth links (less than 28Kb/s) unless you are running the program very frequently. As a result, you are ok most of the time using the default parameters. However, the default parameters may not give you the most accurate results (as detailed below), so you may want to change them.

4.1. Running in Active Probing Mode

In this mode, the program will send some number of TCP FIN packets to ports on the destination host. The port is incremented for each packet. These packets should cause TCP RESET packets to be sent back to the measurement host. The program captures these packets, makes calculations, and decides whether to repeat the experiment. The following arguments control nettimer's behavior:

nettimer { —active_probing_addresses *[src_addr[:src_service]]dst_addr[:dst_service]* } [—active_probing_burst_size *burst_size*] [—active_probing_packet_size *packet_size*] [—active_probing_min_samples *min_samples*] [—active_probing_max_packets_sent *max_packets_sent*] [—active_probing_max_error *max_error*] [—active_probing_timeout *timeout*]

The *dst_addr* specifies the IP address or hostname of the destination. It must be specified. The other parts of the active_probing_adddresses are optional. The *dst_service* specifies the destination port to start with (default: 10100). The *src_addr* and *src_service* specify the source port and service, respectively.

The *packet_size* parameter specifies the size in bytes of the packets to be sent. Increase this value (max: 1500 bytes) when measuring high bandwidth links on slow hosts. Larger packets reduce the effect of client-based sources of error like poor packet capture timing precision (which usually occurs on slow hosts). This is necessary for high bandwidth bottleneck links. On the other hand, larger packets are more likely to encounter interference from other packets and put a larger load on the network.

The *burst_size* parameter specifies the number of packets to sent back-to-back. The larger this number is, the more accurate result you will get, until you send enough packets to cause packets to be dropped from a queue. Also, increases this number increases the load on the network.

The *timeout* parameter specifies the time to wait for the reception of all the expected packets of one burst until another burst is sent. If this is too low, then bursts will be sent before the last burst has cleared from the queue. If this is too high, then measurement will take a long time.

The *min_samples*, *max_packets_sent*, and *max_error* parameters specify the conditions under which the active probing will complete. If the program has collected at least *min_samples* samples and the error is below *max_error*, then the program will exit successfully. Otherwise, it will continue sending bursts until *max_packets_sent* packets have been sent and the program will exit with an exception.

4.2. Interpreting Active Bottleneck Bandwidth Results

In this section, I describe how to interpret the results that nettimer generates.

The first line of the output gives the bandwidth estimate, error estimate, and the number of samples used to generate the estimate. The following lines give the number of packets sent, the number of packets received, and the number of timeouts during measurement. Finally, an exception is printed if the final error exceeds the specified maximum error.

An example of the output:

```
BW: 7995536.719157 Error: 0.554183 Samples: 12
Active probing statistics:
  packets sent: 16
  packets received: 16
  timeouts: 0
KLException caught in program: "nettimer"
 at file: "nettimer.c" function: "main" line: XXX
  Passed at file: "packet_capture.c" function: "packet_capture_event_handler" line: XXX
    meaning: "Client exception."
  Passed at file: "dpcap_flow.c" function: "flow_set_new_packet" line: XXX
    meaning: "client exception"
  Thrown at file: "active_probing.c" function: "active_probing_packet_new" line: XXX
    meaning: "Exceeded active probing sent packets maximum."
nettimer exitting
```

This example shows that the estimated bottleneck link bandwidth is 7.995536 Mb/s, the estimated error is 55%, and the number of samples used was 12. This measurement was done on a busy 10Mb/s Ethernet which is why the error was so high. 16 packets were sent and 16 were received. Since the error was so high, the maximum number of packets were sent, and the program exits with an exception.

You may see that the results vary significantly between runs. The sources of error for the active probing are 1) cross traffic, 2) poor packet capture timing resolution and high bandwidth links, and 3) a short round trip time.

Cross traffic will cause arbitrary error in the measurement. Try measuring at a time when you know the there is little cross traffic (e.g. early in the morning).

Poor packet capture timing precision will prevent you from measuring a high bandwidth link. Poor packet capture timing precision is usually caused by a slow host and bugs in the packet capture library or OS kernel. The solutions are to run nettimer on a faster host, upgrade the packet capture library and/or OS kernel, or increase the active probing packet size.

A short round trip time can cause errors because the TCP RESET packet can arrive before the last TCP FIN packet in a burst is sent. Probe to a host that is farther away, decrease the burst size, or increase the packet size.

5. Measuring the Bottleneck Link Bandwidth Passively

This section describes how to passively measure the bottleneck link bandwidth along a path.

To measure bottleneck link bandwidth, you must specify to nettimer to source of the measurements and the destination of the results. The source of the measurements can either be packet capture servers or trace files. The destination of the results is a file. In the following sections, I will describe how these parameters are specified.

5.1. Measurement Sources

In this section I describe how to set up the measurement sources. The two kinds of measurement sources are nettimer run in distributed packet capture server mode and tcpdump traces. Use the distributed packet capture servers if you want to use the calculations in real time. Use the traces if you want to be able to repeat calculations for testing or debugging.

5.1.1. Running in Distributed Packet Capture Server Mode

You must first decide which hosts you want to take measurements at and then which host will do the actual bandwidth calculation. The bandwidth calculation can consume many CPU cycles, so it may be better to do so on a machine that isn't being measureed. On the other hand, the result of the calculation is likely to be most useful one of the hosts being measured, so it may be better do the the calculation on one of the measured machines. One scenario would be to take measurements at a web client and a web server and then do the calculation at the server.

If you are going to take measurements and do calculation at the same machine, then you don't need to start up a separate distributed packet capture server at that machine. Nettimer run in bandwidth calculation client mode can start one up automatically (see Section 5.2).

On the measurement machines, start up nettimer in packet capture server mode:

nettimer {—run_dpcap_server} [—interface *interface*] [—dpcap_filter *expression*] [—dpcap_server_port *port_num*] [—dpcap_server_packet_buffer *packet_buffer_size*] [—dpcap_server_send_timeout *timeout*] [—dpcap_cap_len *cap_len*]

This will start up a program which will capture all packets that it hears on the *interface* network interface (e.g. "eth0"). If this option isn't specified, then the default interface is used, so this is most useful on multi-homed hosts.

The captured packets are filtered using the tcpdump filtering *expression*. Filtering the packets cuts down on the amount of data sent to each of the clients.

At the same time as it captures packets, the server will listen for incoming TCP connections from clients on *portnum*. The default port is 9090. If the server can't get this port number , then it will pick a random port number and print it out. Clients that want to connect to this server must specify this non-default port number (see Section 5.2.1).

After a client connects, the server will transmit the first *cap_len* bytes of each captured packet to that client. The default value of 60 should be sufficient to get the TCP and IP header of most packets on most link layer technologies. However, some link layers and/or packets with many IP options may push the TCP header to far into the packet, so this parameter may need to be increased. On the other hand, the larger this value, is the more data is sent to each client.

The server buffers up captured headers until *packet_buffer_size* bytes have been captured or *timeout* seconds have passed since the last buffer was sent to the client. The tradeoff here is between the efficiency and timeliness of communication between the server and its clients. The *packet_buffer_size* can safely be set to be much larger than the TCP maximum segment size. However, setting the *timeout* to a low values will cause timely but possibly small packets to be sent, while setting it to a large values will cause less timely, but likely larger packets to be sent.

The server can handle any number of clients. The clients can open and close connections at any time, but they will only receive packets that were captured during the time they were connected.

5.1.2. Gathering Traces

The traces must be gathered using a version of tcpdump with the same trace file format as the version of libpcap that is linked with nettimer. The name of the trace file must be of the form `IPAddress_pcap` or `IPAddress_*_pcap` where IPAdress is the IP address of the host were the trace was taken. In the absence of synchronized clocks, Nettimer needs this information to be able to determine whether a packet is coming or going in a trace.

5.2. Running in Bandwidth Calculation Client Mode

In this section I describe how to run nettimer in bandwidth calculation client mode. In this mode, nettimer gathers measurements from measurement sources, calculates the bandwidth, and outputs the result. Before running in this mode, you must set up the measurement sources (see Section 5.1). There are so many options for running the bandwidth calculation client that I have divided them into the general options and the calculation options.

5.2.1. Client General Options

The general client options cover how to specify the verbosity, how to specify the measurement sources, how to get results, and how to control memory usage.

nettimer [—version] [—verbose *verbosity_level*] { —dpcap_servers *address_or_hostname:portnum* | —read_trace *file_name* } [—interface *interface*] [—dpcap_filter *expression*] [—log_file_name *log_file_name*] [—update_file_name *update_file_name*] [—update_interval *update_interval*] [—no_convert_addressess] [—qualify_host_names] [—maximum_number_of_flows *max_num_flows*] [—maximum_flow_length *max_flow_len*]

The `--version` option causes the version number to be printed to stdout.

The `--verbose` option controls how verbose the output is. The default is 0.

If you are using live distributed packet capture servers, then specify the `--dpcap_servers` option followed by the addresses or hostnames of the servers as a single command line argument. For example, to take measurements from distributed packet capture servers running on the localhost, listening on the default port on host1, and listening on port 10001 on host2:

```
nettimer --dpcap_servers "localhost host1 host2:10001"
```

.

Nettimer will automatically start up a local distributed packet capture server if "localhost" is specified as one of the servers, so you almost never need to explicitly start up a distributed packet capture server on the same host as the bandwidth calculation client. The `--interface` option specifies the *interface* for this local server to capture on.

On the other hand, if you are reading trace(s), then use must specify the `--read_trace` option followed by the filename(s) of the traces as a single command line argument. For example, to read trace from the files `127.0.0.2_pcap` and `127.0.0.3_pcap`:

```
nettimer --read_trace "127.0.0.2_pcap 127.0.0.3_pcap"
```

The traces must also follow the restrictions described in Section 5.1.2.

Regardless of the type of the measurement sources, you can specify a filter *`expression`* using the `--dpcap_filter` option to reduce the amount of data that has to be transferred from the servers to the client and/or the amount of computation the client has to do. If you are using live servers, then the *`expression`* is sent to all the servers when the client connects. Note that the server uses both its own filtering *`expression`* and the client's *`expression`* when determining what packet reports are sent to that client. If you are reading from traces, then the *`expression`* is applied to all of the filters.

Nettimer can generate results in three ways: to the screen, as a file updated with only the latest results, and to a log file containing all the results. By default the results are written to the screen. The user can also redirect stdout to a file, which will then be updated continuously with the latest results. The user can also specify the update file through the `--update_file_name` option. The flows are sorted from most-recently used to least-recently used. Use the `--update_interval` option to control how frequently the screen or this file is updated. To exit this mode, press 'q'. Any other key causes the file to be updated. Nettimer uses `flock()` to lock the update file while it is being updated, so programs that need to read a consistent updated file should `flock()` it before reading. To generate the log file, specify the `--log_file_name` option. This option can generate very large files if nettimer runs for a long time.

By default, nettimer output uses an unqualified hostname to identify hosts. The `--no_convert_addresses` option causes nettimer to use IP addresses instead of hostnames. Nettimer will also use IP addresses if it cannot resolve addresses to names while it is calculating (e.g. no network connection or no reverse mapping in the DNS server). The `--qualify_host_name` option causes nettimer to write fully domain-qualified hostnames.

Use the `--maximum_number_of_flows` option (default: 100) to control the maximum number of flows that the client keeps track of. If this limit is exceeded, then the oldest flow and all its packets are discarded. You might have to increase this limit if you want to keep track of many flows or decrease it if you are running out of memory. Use the `--maximum_flow_length` option (default: 1000) to control the maximum number of packets each flows can have. You might have to increase this limit if you want to use information more than 1000 packets ago or decrease it if you are running out of memory.

5.2.2. Client Calculation Options

The client calculation options control how the client does the bandwidth calculation. These settings exist mostly to make experimenting with the algorithms easier, so you shouldn't need to tweak them.

nettimer [—calc_restrict_packet_pair *`packet_pair_algorithm`*] [—calc_window_size *`max_flow_len`*] [—calc_kernel_bound_ratio *`kernel_bound_ratio`*] [—calc_kernel_resolution *`kernel_resolution`*] [—calc_kernel_function *`kernel_function`*] [—calc_min_bunch_size *`min_bunch_size`*] [—calc_max_bunch_size *`max_bunch_size`*] [—calc_potential_bw *`potential_bw`*] [—calc_min_delta_rtt] [—calc_max_delta_rtt] [—calc_no_bw_ratio *`max_flow_len`*]

The `--calc_restrict_packet_pair` option (default: auto) restricts the client to only calculating one of SBPP (Sender Based Packet Pair), RBPP (Receiver Based Packet Pair), ROPP (Receiver Only Packet Pair), or auto (automatically select the optimal algorithm). This option is only useful to compare the effectiveness of the different algorithms.

The `--calc_window_size` option controls how many samples of history to use in the calculation of bandwidth. Larger values allow a more stable calculation while smaller values allow faster reaction to bandwidth change. This value cannot exceed the maximum flow length(see Section 5.2.1).

The `--calc_kernel_bound_ratio`, `--calc_kernel_resolution`, and `--calc_kernel_function` options control how the kernel density algorithm works. The `--calc_kernel_bound_ratio` (default: .05) option controls the ratio of the kernel width to the kernel's x value. A larger value gives a smoother distribution function, while smaller values are faster to calculate. The `--calc_kernel_resolution` (default: 100) option controls the minimal parameter resolution of the kernel density algorithm. This means that different samples as much as *`kernel_resolution`* apart may generate the same kernel value. A larger value will make the algorithm faster, while a smaller value will make the algorithm more accurate. The `--calc_kernel_function` option (default: triangular) selects what kernel function to use. The choices are "uniform" and "triangular". In almost all cases, the default is sufficient.

5.3. Interpreting Passive Bottleneck Bandwidth Results

In this section, I describe how to interpret the results that nettimer generates.

Nettimer organizes its measurements by flow. A flow is defined as all packets that travel from a specific IP address to another specific IP address. For each flow, nettimer calculates several bandwidth metrics, like Sender Based Packet Pair or Receiver Based Packet Pair. The types and meanings of these metrics depends on the orientation of the distributed packet servers to the flow. A flow could originate from a distributed packet server, terminate at a distributed packet server, both arrive at and leave from a distributed packet server (e.g. the libdpcap server is a router), or bypass a libdpcap server completely.

Nettimer uses some rules to determine which metric it should calculate. If it only has the arrival timings for packets in a flow at a libdpcap server, then it computes Receiver Only Packet Pair to that server. If it has both transmission timings and round trip timings for packets in a flow, then it computes Sender Based Packet Pair from that server. If it has the transmission timings for a flow from one libdpcap server and the arrival timings at another server, then it computes Receiver Based Packet Pair from the transmission timings server to the arrival timings server. For any particular flow, more than one of these rules may apply, and for each rule, there may be multiple combinations of libdpcap servers. As a result, nettimer generates multiple bandwidth metrics for each flow.

As described in Section 5.2.1, nettimer can generate results by updating a file with only the latest results or appending to a log file. Both are described in the following sections.

5.3.1. Updated Results File

The updated results file consists of a line of column headings followed by a variable number of lines, each describing a metric of a particular flow. The columns are space delimited.

The meaning of the columns are as described below. Only the non-verbose columns are documented.

Updated Results File Columns

Heading : FlowSource
Meaning : The hostname or IP address of the source of the flow.

Heading : FlowDest
Meaning : The hostname or IP address of the destination of the flow.

Heading : FI
Meaning : The index (relative to the command line specification) of the distributed packet capture server or trace file where the transmission timings for the packets of this flow are taken. An index of -1 indicates that nettimer has no information about the transmission timings for this flow.

Heading : TI
Meaning : The index (relative to the command line specification) of the distributed packet capture server or trace file these measurements are taken to.

Heading : Metr
Meaning : The name of the metric that is calculated. "SBPP" = Sender Based Packet Pair, "RBPP" = Receiver Based Packet Pair, "ROPP" = Receiver Only Packet Pair.

Heading : Bandwidth
Meaning : The bandwidth measured in bits/second.

An example of non-verbose output:

```
Cmds: (q)uit, any other key to update
FlowSource      FlowDest        FI TI Metr    Bandwidth
192.168.45.1    10.0.0.2        -1  0 ROPP  87972363.38
192.168.45.1    10.0.0.2         0  0 SBPP   4004618.24
192.168.45.1    10.0.0.2         1  0 RBPP  87833710.03
10.0.0.2        192.168.45.1    -1  0 ROPP    139792.80
10.0.0.2        192.168.45.1     0  1 RBPP    137449.52
10.0.0.2        192.168.45.1     1  1 SBPP     43924.45
```

The lines 3-5 describe a flow from 192.168.45.1 to 10.0.0.2. The libdpcap servers specified on the command line were "10.0.0.2 192.168.45.1".

Line 3 says that the bandwidth using Receiver Only Packet Pair and the arrival timings taken at the 0 libdpcap server (10.0.0.2) is 87972363.38 b/s. In other words, this is an estimate of the bottleneck bandwidth from 192.168.45.1 to 10.0.0.2.

Line 4 says that the bandwidth using Sender Based Packet Pair and the transmission and round trip timings taken at the 0 libdpcap server (10.0.0.2) is 4004618.24 b/s. Since the packets of this flow are travelling to 10.0.0.2, this is the bottleneck bandwidth from the driver in 10.0.0.2's operating system to its TCP/IP stack and back to the driver.

Line 5 says that the bandwidth using Receiver Based Packet Pair, the transmission times at the 1 libdpcap server (192.168.45.1), and the arrival times at the 0 libdpcap server (10.0.0.2) is 87833710.03 b/s. This is another (and probably most accurate) estimate of the bottleneck bandwidth from 192.168.45.1 to 10.0.0.2.

The last three lines describe the reverse flow (10.0.0.2 to 192.168.45.1) of the one described above. All of the lines describe similar metrics, except in the reverse direction. However, the bandwidths are much smaller. In fact, this path has symmetric bandwidth, but only TCP acknowledgements are flowing in the reverse direction. Small and slowly sent packets like acknowledgements are insufficient to measure the high (100Mb/s) bottleneck bandwidth that exists on this path.

When both the sending and arrival times of packets are available, then nettimer in most cases will not compute the Receiver Only Packet Pair bandwidth between hosts because it is inferior to the Receiver Based Packet Pair bandwidth. However, sometimes nettimer gets the arrival times for packets before it gets the send times (e.g. the client is at the receiver). In this case, nettimer will calculate a ROPP bandwidth because it has no way of knowing when or if the send timings will arrive. When the send timings do arrive, then nettimer will calculate the RBPP bandwidth, but the ROPP bandwidth will remain.

5.3.2. Log File Results

The log file consists of series of lines, one for the transmission or arrival of a packet that could change a bandwidth estimate. This usually happens when a packet is sent or arrives. The columns are space delimited.

The meaning of the columns are as described below. Only the non-verbose columns are documented.

Updated Results File Columns

Column : 1
Meaning : The time at which a packet was sent.

Column : 2
Meaning : The time at which a packet was received.

Column : FlowSource
Meaning : The hostname or IP address of the source of the flow.

Column : FlowDest
Meaning : The hostname or IP address of the destination of the flow.

Column : FI
Meaning : The index (relative to the command line specification) of the distributed packet capture server or trace file where the transmission timings for the packets of this flow are taken. An index of -1 indicates that nettimer has no information about the transmission timings for this flow.

Column : TI
Meaning : The index (relative to the command line specification) of the distributed packet capture server or trace file these measurements are taken to.

Column : ID
Meaning : The IP packet id.

Column : Metr
Meaning : The name of the metric that is calculated. "SBPP" = Sender Based Packet Pair, "RBPP" = Receiver Based Packet Pair, "ROPP" = Receiver Only Packet Pair.

Column : Bandwidth
Meaning : The bandwidth measured in bits/second.

6. Measuring All Link Bandwidths Along a Path

This section describes how to measure all the link bandwidths along a path.

Undocumented

7. Frequently Asked Questions

This section lists the frequently asked questions and their answers.

1. Q: When are you going to port nettimer to Windows NT, Solaris, FreeBSD, OS X, etc.?

   A: I'm not planning to do any ports, but I will accept patches for ports.

2. Q: Why is nettimer dumping core or giving bizarre output?

   A: Send me a tcpdump trace that causes the problem and I will look at it. If you can't generate a tcpdump trace that causes the problem, then it will be very difficult for me to fix the problem.

3. Q: Why is nettimer is giving me little or no output in passive bottleneck bandwidth measurement mode?

   A: Nettimer does not generate its own probe traffic in passive bottleneck bandwidth measurement mode. This is supposed to be feature so that it does not burden the network. If you want to generate some artificial traffic just to measure bandwidth, then run it in active probing mode or use ping while nettimer is running:

   ```
   ping -c 10 -l 10 -s 1400 some.I.P.address
   ```

   This will send 20 1408 byte packets back-to-back to some.I.P.address. You need to be root to use these options to ping. You should reduce the 20 to something smaller if you are on a low bandwidth link. Don't do this too often because it's hard on the network.