Nominum®

resperf

resperf Performance Tool Manual

Version *2.0.0*
Date *March 1, 2012*

1

resperf

Use the *resperf* command to measure the resolution performance of caching Domain Name Service (DNS) servers. The *resperf* command is a caching-specific DNS testing companion to the *dnsperf* command (the Nominum authoritative DNS testing tool).

While the *dnsperf* command was designed for benchmarking authoritative DNS servers, the *resperf* command addresses the requirements inherent in the testing of caching name servers.

A key difference between caching and authoritative DNS servers is that caching servers may need to work on thousands of queries in parallel to achieve maximum throughput, while authoritative servers process queries in order and one-at-a-time.

For authoritative servers, the *dnsperf* command sends a small burst of back-to-back queries to fill up network buffers (keeping the server at 100% utilization); a new query is sent when a response is received. This approach works well for authoritative and caching servers in a closed laboratory environment in which the server is “talking” to a simulated Internet on a single LAN. For caching servers, however, this approach fails to accurately test performance when “talking” to the actual Internet—in real-world conditions, a caching server may need to process thousands of queries in parallel to achieve maximum throughput.

Command Synopsis

```
resperf [-a local_addr] [-b bufsize] [-c constant_traffic_time]
   [-d datafile] [-D] [-e] [-f family] [-h] [-i plot_interval]
   [-L max_loss] [-m max_qps] [-p port][-P plotfile] [-r rampup_time]
   [-s server_addr] [-t timeout] [-x local_port]
   [-y [algorithm:]name:secret]
```

respf P

Option	Description
`-a` *`local_addr`*	Specifies the local address form which to send requests. The default is the wildcard address.
`-b` *`bufsize`*	Sets the socket send/receive buffer size in kilobytes.
`-c` *`constant_traffic_time`*	Specifies the length of time, in seconds, for which to send a constant stream of traffic. The default is 0 seconds.
`-d` *`datafile`*	Specifies an input data file. The default is **`stdin`**.
`-D`	Sets the DNSSEC OK bit (implying EDNS).
`-e`	Enables EDNS 0.
`-f` *`family`*	Specifies the address *`family`* of DNS transport. Options are: □ `any`—Uses the address type of the address as specified in `-s`. • `inet`—IPv4 transport. • `inet6`—IPv6 transport. The default is `any`.
`-h`	Prints a usage message and exits.
`-i` *`plot_interval`*	Specifies the time interval between plot data points, in seconds. The default is 0.
`-L` *`max_loss`*	Specifies the maximum acceptable query loss, as a percentage. The default is 100.
`-m` *`max_qps`*	Specifies the maximum number of queries per second. The default is 100000.
`-p` *`port`*	Sets the port on which to query the server. The default is 53.

Table 1-1 resperf options

Option	**Description**
`-P` *`plotfile`*	Specifies the name of the plot data file. The default is *resperf.gnuplot*.
`-r` *`rampup_time`*	Specifies the ramp-up time in seconds. The default is 60.
`-s` *`server_addr`*	Sets the server to query. The default is `127.0.0.1`.
`-t` *`timeout`*	Specifies the request timeout value, in seconds. After *`timeout`* is reached, *resperf* will no longer wait for a response to a particular request after this many seconds have elapsed. Default is 45 seconds.
`-x` *`local_port`*	Specifies the local port from which to send requests. Default is the wildcard port (0).
`-y [`*`algorithm`*`:]`*`name`*`:`*`secret`*	Adds a TSIG record (as per RFC 2845) to all packets sent, using the specified TSIG key *`name`*, *`secret`*, and, optionally, *`algorithm`*. The *`secret`* is expressed as a base-64 encoded string. If you do not specify an algorithm, the algorithm defaults to hmac-md5.

Table 1-1 resperf options *(continued)*

Understanding resperf

resperf, unlike *dnsperf*, sends DNS queries at a controlled, steadily increasing rate—by default, *resperf* sends traffic for 60 seconds, linearly increasing the amount of traffic from zero to 100,000 queries per second (qps).

During testing, *resperf* tracks responses from the server as well as response rates, failure rates, and latencies. After *resperf* stops sending traffic, it continues to “listen” for responses for an additional 40 seconds to give the server time to respond to the last queries sent.

NOTE The 40-second time limit is longer than the overall query timeout of both Nominum Vantio and current BIND versions.

Defining Success

A successful test is complete when the query rate exceeds the server capacity and queries are dropped, causing the response rate to either stop or decrease as the query rate increases.

Determining Maximum Throughput

Maximum throughput is determined from the plot as either:

- The highest response rate on the plot.
- The response rate at the point where a significant number of queries begin to be dropped.

Test Results

Test results are written to a file in a tabular format as a set of measurements of the query rate, the response rate, the failure response rate and the average query latency as functions of time, and may be plotted using *gnuplot* or a similar plotting tool. See "*The Plot Data File*" on page 10.

resperf, when constructing plot data, always places each query at the point in time at which the query was sent, *not* the point in time at which the response (if any) was received. This permits easy comparison of query and response rates.

Operational Considerations

Benchmarking a live caching server can have serious operational ramifications.

Query Volume and Bandwidth Saturation

A fast caching server such as Nominum Vantio, running on a XEON® server and resolving a mix of cacheable and non-cacheable queries typical of an ISP's customer traffic, is entirely capable of resolving over 100,000 qps. In the process of resolving those queries, the server will send more than 40,000 qps to authoritative Internet servers, and receive answers to most of those queries.

If you assume an average request size of 50 bytes and a response size of 150 bytes, this adds up to approximately 16Mb per second (Mbps) of outbound traffic and approximately 48 Mbps of inbound traffic.

Ensure that your internet connection can handle this amount of bandwidth with room to spare. If you fail to do so:

1. You may saturate your connection, causing a service degradation for your users.
2. More seriously (in the context of testing accuracy), you may wind up measuring the speed of the *connection* instead of the speed of the *server*.

Firewalls

Ensure that no stateful firewalls exist between the caching server and the Internet. As a rule, stateful firewalls can't handle the amount of UDP traffic generated by *resperf*, and may skew test results by:

- Dropping packets.
- Locking up.
- Crashing.

Separate But Equal

It is recommended that *resperf* and the name server under test be run on separate machines, so that the CPU usage of *resperf* does not slow down the name server. The two machines should be connected with a fast network, preferably a dedicated Gigabit Ethernet segment. Testing through a router or firewall is not advisable.

Performance testing at the traffic levels involved is essentially a hard real-time application. At a query rate of 100,000 qps, a 1/100s delay translates into 1000 incoming UDP packets—this is far more than most operating systems can buffer.

Therefore—on the same LAN as the server under test—run *resperf* on *its own machine*, ensuring that:

- The machine running *resperf* is *at least* as fast as the server being tested—otherwise, it may become a performance bottleneck.
- There are no other applications running on the machine running *resperf*.

Timer Granularity and CPU

Most operating system timers are of too coarse a granularity to schedule packet transmissions at sub-millisecond intervals. As a result, *resperf* busy-waits between packet transmissions, constantly polling for responses. It's therefore normal for *resperf* to consume 100% CPU during the whole test run, even during periods where query rates are relatively low.

What You Will Need

A Query Input File

For *resperf,* you need to construct a *dnsperf* input file containing a large and realistic set of queries, on the order of ten thousand to a million. This can be the same file you use for testing *dnsperf.* The input file contains one line per query, consisting of a domain name and an RR type name separated by a space. The class of the query is implicitly IN.

The latest query file is available for download at *ftp://ftp.nominum.com/pub/nominum/dnsperf/data/*.

When measuring the performance serving non-terminal zones such as the root zone or TLDs, note that such servers spend most of their time providing referral responses, not authoritative answers. Therefore, a realistic input file might consist mostly of queries for type A for names *below*, not at, the delegations present in the zone. For example, when testing the performance of a server configured to be authoritative for the top-level domain *fi*, which contains delegations for domains like
helsinki.fi and *turku.fi*, the input file could contain lines like

```
www.turku.fi A
www.helsinki.fi A
```

where the *www* prefix ensures that the server will respond with a referral. Ideally, a realistic proportion of queries for nonexistent domains should be mixed in with those for existing ones, and the lines of the input file should be in a random order.

Configuration

In *resperf* plots, limits on the number of simultaneous resolutions (like the `max-recursive-clients` statement in Nominum Vantio or the `recursive-clients` option in BIND 9) show up as increases in the number of failure responses.

To avoid this, increase these limits.

Nominum’s recommended settings are:

- Vantio—Set `max-recursive-clients` to 10000 (ten thousand).
- BIND 9—Set `recursive-clients` to 100000 (one hundred thousand).

NOTE Ensure the server cache is empty before starting a test. If the cache contains data from a previous test that used the same queries, the server answers all queries from the cache, yielding inflated performance numbers.

The resperf-report Script

The `resperf-report` script invokes *resperf,* directing the command output to a file that creates an HTML report.

To use the ***`resperf-report`*** script, you must install gnuplot on your server. The installed version of gnuplot must support the png terminal driver. If gnuplot supports gif but does not support png, open the ***`resperf-report`*** script and change the line **`terminal=png`** to **`terminal=gif`**.

Running Tests

To run *resperf*, providing the minimal requirements of a server IP address and the query data file, issue a command like:

```
# resperf -s 10.0.0.2 -d queryfile
```

As *resperf* runs, some status messages and summary statistics are written to **stdout**, and plot file data is written to *resperf.gnuplot* in the current directory (unless another plot file name has been specified with the `-P` option). The following example shows sample output from the **`resperf`** command:

```
DNS Resolution Performance Testing Tool
Nominum Version 2.0.0.0.d
[Status] Command line: resperf -p 12345 -d in
[Status] Sending
[Status] Fell behind by 1039 queries, ending test at 39331 qps
[Status] Waiting for more responses
[Status] Testing complete
Statistics:
Queries sent: 463036
Queries completed: 463036
Queries lost: 0
Run time (s): 100.000000
Maximum throughput: 36250.000000 qps
Lost at that point: 0.00%
```

Test Duration

A test run, using the default settings, takes 100 seconds at most, comprised of 60 seconds of traffic ramp-up followed by 40 seconds of waiting for responses. However, in practice, the 60-second traffic phase is usually curtailed.

There are several different conditions under which *resperf* will transition from the traffic-sending phase to the waiting-for-responses phase:

- *resperf exceeds 65,536 outstanding queries*—This is the most frequent reason *resperf* stops sending queries before the 60 seconds has finished, and this occurs because *resperf* has exceeded the capacity of the server being tested.

  The limit of 65,536 queries originates with the number of possible ID field values in the DNS packet—*resperf* allocates a unique ID for each outstanding query, and is therefore unable to send further queries if the set of possible IDs is exhausted.

- *resperf is unable to send queries fast enough*—*resperf* may fall behind in query transmission because it cannot send queries quickly enough. If so, once the backlog reaches 1000 queries, *resperf* prints a message describing the number of backlogged queries and stops sending traffic.

  If this message appears, ensure that the machine running *resperf* is sufficiently fast and has no other applications running, and monitor the CPU usage of the server under test.

- *resperf successfully reaches the maximum query rate*—*resperf* may run for its full allotted time and successfully reach the maximum query rate (by default, 60 seconds and 100,000 qps).

Troubleshooting

Server response rates, regardless of what causes the test to end, should flatten towards the end of the test. If this trend does not show in the result plot, the server load is not heavily enough.

If the CPU usage of the server under test doesn’t reach 100% (or close to it) at the point of maximum traffic, you likely have a bottleneck in some other part of the test harness. For example, your external Internet connection may be saturated.

As previously mentioned, if *resperf* is unable to send queries quickly enough, ensure that the machine upon which it is running is sufficient for the purposes of testing—ensure that it’s fast enough, and that there are no other applications running on the machine.

The Plot Data File

Test runs are divided by default into 0.5-second time intervals for the purposes of generating the plot data file (alternative intervals may be specified using `-i`). Each line in the plot data file corresponds to one interval, and contains the following values specified as floating-point integers:

Time

The midpoint of this time interval in elapsed seconds since the beginning of the test run.

Target queries per second

The number of queries per second *scheduled* to be sent in this time interval.

Actual queries per second

The number of queries per second *actually* sent in this time interval.

Responses per second

For this time interval, the number of responses received to queries sent, divided by the interval length.

Failures per second

For this time interval, the number of non-NXDOMAIN and non-NOERROR responses received to queries sent, divided by the interval length.

Average latency

For queries sent in this time interval, the average time between the sending of a query and the receiving of a response.

Measurements for any given query always apply to the time interval within which the query was sent, not the time interval within which the response (if any) was received. For example, if no queries are dropped, the query and response curves are identical. However, if a plot shows 10% failure responses at t=5 seconds, it means that 10% of the *queries sent* at t=5 seconds eventually failed, not that 10% of the *responses received* at t=5 seconds were failures.

Plotting Test Results

The *resperf-report* shell script runs *resperf* with its output redirected to a file, from which an illustrated report in HTML format is automatically generated. *resperf-report* accepts the command-line arguments in Table 1-1, passing those arguments unchanged to *resperf*.

Reports are stored with a unique filename based on the current date and time. For example:

```
20060812-1550.html
```

PNG images of the plots and other auxiliary files are stored in separate files beginning with the same date-time string.

For example, to benchmark a server running on `10.0.0.2`, you could run

```
# resperf-report -s 10.0.0.2 -d queryfile
```

and then open the resulting *20060812-1550.html* file in a web browser.

NOTE *resperf-report* uses *gnuplot* to generate plots—ensure that *gnuplot* is installed, and that it supports the *gif* terminal driver. *gnuplot* may be obtained at *http://www.gnuplot.info*.

To copy the report to a separate machine for viewing, copy the *.gif* files along with the *.html* file, or copy all of the files using

```
# scp 20060812-1550.* host:directory/
```

Resperf Plots

The *resperf-report* contains two plots generated by gnuplot:

- Query/response/failure rate
- Latency

Query/response/failure Rate Plot

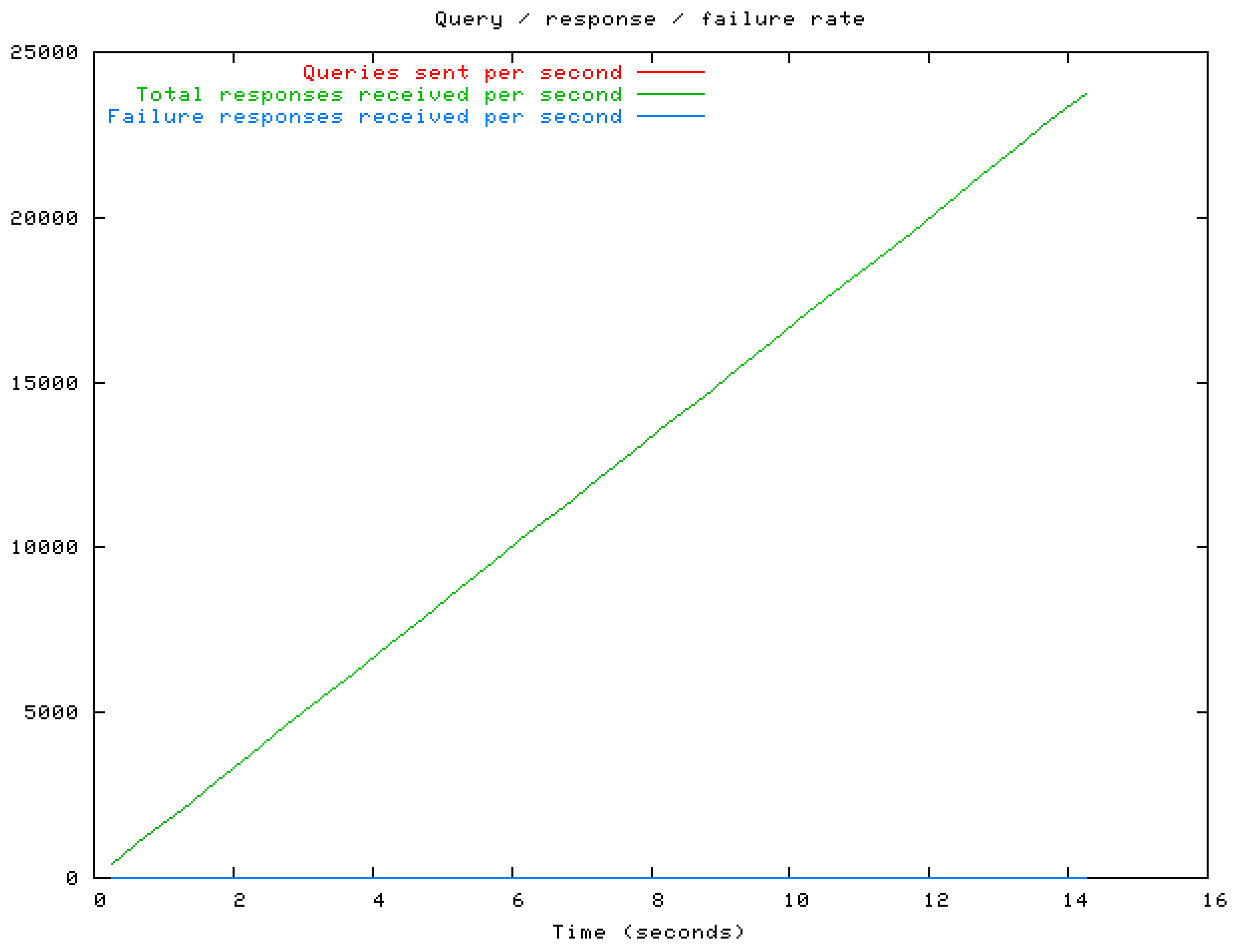
The Query/response/failure rate plot contains three graphs. Use the graphs these determine how the server behaves under an increasing traffic load:

- Queries sent per second—Displays the amount of traffic being sent to the server. A straight diagonal line reflects the linear ramp-up of traffic.
- Total responses received per second—Displays how many of the queries received a response from the server. All responses are counted, whether successful (when a NOERROR or NXDOMAIN message is returned) or not (when a SERVFAIL message is returned).
- Failure responses received per second—Displays how many of the queries received a failure response. A response is considered to be a failure if its RCODE is not NOERROR nor NXDOMAIN.
  This graph, which appears near the bottom of the plot, typically shows a linear ramp with fluctuation where failing queries are interspersed with successful queries. The number of failures increases in proportion to the traffic load. A sharp spike in this graph indicates the load has exceeded the capacity of the server. This occurs if the server reacts to an overload by sending SERVFAIL responses (instead of by dropping queries). A sudden increase in the number of failures indicates you need to increase the exceeded limit.

NOTE Initially, the “Total responses received per second” graph may overlap the “Queries sent per second” graph. As the load exceeds the server capacity, the “Total responses received per second” graph flattens out and diverges from the “Queries sent per second” graph, indicating that some queries are being dropped.

Figure 1-1 shows an example of a query/response/failure rate plot.

Figure 1-1 Query/Response/Failure Rate Plot



Latency Plot

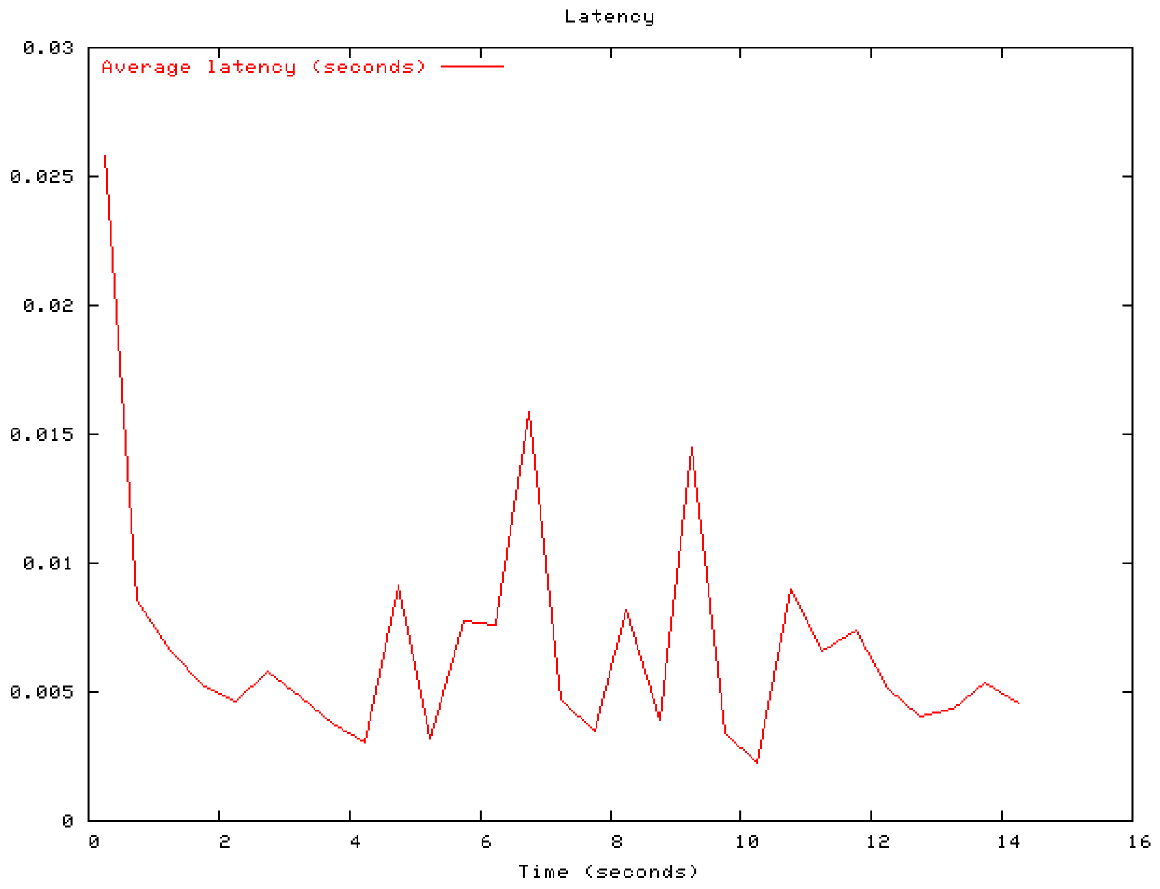
The Latency plot contains an “Average latency” graph that shows latency variations during the course of a test. The “Average latency” graph typically exhibits a downward trend because, during the test, the cache hit rate improves as the number of responses cached increases (and the latency for a cache hit is smaller than for a cache miss). A sharp spike in the graph indicates the point at which the server becomes overloaded.

Do not use the latency graph for absolute latency meaFsurements or comparisons between servers; the latencies shown in the graph do not represent production latencies due to an initially empty cache, and the deliberate overloading of the server occurs towards the end of the test.

NOTE All latency measurements are displayed on the plot at the horizontal position that corresponds to the time when the query was sent (and not when the response, if any, was received). This enables the comparison of query and response rates; for example, if no queries are dropped, the query and response graphs are identical. If the plot shows 10% failure responses at 5 seconds, this indicates that 10% of the queries sent at 5 seconds failed (and not that 10% of the responses received at 5 seconds were failures).

Figure 1-2 shows an example of a latency plot.

Figure 1-2 Latency Plot



Interpreting Results

Summary statistics are printed to standard output at the end of the test, and include the server's measured maximum throughput.

By default, the maximum throughput is the highest point on the response rate plot, without regard to the number of queries dropped or failing at that point. If you want *resperf* to report throughput at the point in the test where the percentage of queries dropped exceeds a given limit, which may be a more realistic indication of how much the server can be loaded while still providing an acceptable level of service, use the `-L` command-line option.

For example, specifying the following command forces *resperf* to report the highest throughput reached before the server starts dropping more than 10% of queries received:

```
# resperf -s 10.0.0.2 -L 10 -d queryfile
```

When a server is driven into overload, the service it provides may deteriorate gradually, and this deterioration can manifest itself in any of the following ways:

- queries being dropped
- an increase in the number of SERVFAIL responses
- an increase in latency

A Note On Failed Queries

All plots should be manually inspected to ensure that they don’t contain an abnormal number of failed queries. It is not possible to automatically constrain results based upon the number of failed queries, because failed queries (unlike dropped queries) occur even when the server is not overloaded. Additionally, the number of failed queries is heavily dependent upon both query data and network conditions.

Generating Constant Traffic

Generate a constant stream of traffic by using the “-c” (constant) and “-m” (maximum number of queries per second) options, as in the following example:

```
# resperf -d input_file -s server -m 10000 -c 3600
```

To avoid the initial 30-second gradual ramp-up of traffic at the beginning of the test (which can overwhelm a server that is starting with an empty cache), include “-r 0” option to instantly start the stream of traffic, as shown in the following example:

```
# resperf -d input_file -s server -m 10000 -c 0 -r 0
```

NOTE *resperf* does a linear ramp-up of traffic from 0 to "-m" queries per second over a period of **`-r`** seconds, followed by a plateau of steady traffic at "-m" queries per second (lasting for **`-c`** second), followed by a 40 second wait for responses. To suppress the ramp-up or plateau, include "-r 0"and "-c 0" in the *resperf* command string. "-c 0" is the default.

Be aware that sending traffic at high rates for a long time requires large amounts of input data. A long-running test generates a large amount of plot data that is kept in memory for the duration of the test. To reduce the memory usage and the size of the plot file, include the "- i" option as follows to increase the interval between measurements from the default of 0 seconds:

```
# resperf -d input_file -s server -m 10000 -c 0 -r 0 -i 5
```

NOTE When using *resperf* for long-running tests, ensure the traffic rate specified with the "-m" option can be sustained by *resperf* and the server being tested. Otherwise, the test will fails if query IDs run out (because of large numbers of dropped queries) or if *resperf* falls behind its transmission schedule.