dnsperf

DNS Performance Tool Manual

Version *2.1.0*
Date *May 2, 2014*

Copyright ©2002-2014 Nominum, Inc. - All Rights Reserved

This software and documentation is subject to and made available pursuant to the terms of the Nominum License Agreement, and may be used or copied only in accordance with the terms of that Agreement. This manual, in whole or in part, may not be reproduced, translated or reduced to any machine-readable form without prior written approval from Nominum, Incorporated.

Nominum, Incorporated
2000 Seaport Boulevard
Suite 400
Redwood City, CA, 94063
USA
http://www.nominum.com

Centris, Navitas and Vantio are trademarks of Nominum, Incorporated.

All other trademarks are the property of their respective companies.

dnsperf

dnsperf is an authoritative-server-specific Domain Name Service (DNS) performance testing tool. It is primarily intended for measuring the performance of authoritative DNS servers, but it can also be used for measuring caching server performance in a closed laboratory environment. For testing caching servers resolving against the live Internet, the *resperf* program is preferred.

We recommend that *dnsperf* and the name server under test be run on separate machines, so that the CPU usage of *dnsperf* itself 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.

Command Synopsis

```
dnsperf [ -a local_address][ -b bufsize ] [ -c clients ] [ -d datafile ]
[ -D ] [ -e ]  [ -f family ] [ -h ] [ -l  limit ] [ -n runs_through_file ]
[ -p port ] [ -q num_queries ] [ -Q max_qps ] [ -s server_addr ]
[ -S stats_interval ][ -t timeout ] [ -T threads ][ -u ] [ -v ]
[ -x local_port ] [ -y [algorithm:]name:secret ]
```

dnsperf Options

Option	Description
`-a local_address`	Specifies the local address form which to send requests. The default is the wildcard address.
`-b bufsize`	Sets the size of the socket's send and receive buffers, in kilobytes.
`-c clients`	Enables the local server to act as multiple clients and specifies the number of clients represented by this server. The server sends requests from multiple sockets. By default, the local server acts as a single client.
`-d datafile`	Specifies the input data file. If not specified, *dnsperf* reads from standard input.
`-D`	Sets the DO (DNSSEC OK) bit in all packets sent, as per RFC 3225. This also enables EDNS 0, which is required for DNSSEC.
`-e`	Enables EDNS0, as per RFC 2671, by adding an OPT record to all packets sent.
`-f family`	Specifies the address family used for sending DNS packets. The possible values are: • inet—Use IPv4. • inet6—Use IPv6. • any—Either IPv4 or IPv6 as appropriate. If “any” (the default value) is specified, dnsperf uses whichever address family is appropriate for the server to which it is sending packets.
`-h`	Prints a usage statement and exit.
`-l limit`	Specifies a time limit for the run, in seconds.
`-n runs_through_file`	Specifies a maximum number of runs through the input file. If no time limit is set, the file is read exactly this many times; if a time limit is set, the file may be read fewer times.
`-p port`	Sets the port on which the DNS packets are sent. If not specified, the standard DNS port (53) is used.

Table 1-1 dnsperf options

Option	Description
`-q num_queries`	Sets the maximum number of outstanding requests. When this value is reached, *dnsperf* stops sending requests until either responses are received or its requests time out. The default value is 100.
`-Q max_qps`	Limits the number of requests per second.
`-s server`	Specifies the name or address of the server to which requests will be sent. The default is the loopback address, 127.0.0.1.
`-t timeout`	Specifies the request timeout value, in seconds. After `timeout` is reached, *dnsperf* will no longer wait for a response to a particular request after this many seconds have elapsed. Default is five seconds.
`-T threads`	Run multiple client threads. By default, *dnsperf* uses one thread for sending requests and one thread for receiving responses. If this option is specified, *dnsperf* will instead use N pairs of send/receive threads. NOTE: Because this process impacts CPU and memory, we recommend limiting the number of threads.
`-u`	Instructs *dnsperf* to send DNS dynamic update messages, rather than queries. NOTE: The format of the input file is different in this case—see *"Constructing a Dynamic Update Input File"* on page 5.
`-v`	Enables verbose mode, where the command output includes latency measurements and the DNS RCODE of each response. In the output, query time outs are reported with the string "T" (instead of a normal DNS RCODE). An "I" string in the output indicates the query was interrupted.
`-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, `agorithm`. 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 dnsperf options *(continued)*

Operational Considerations

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 *dnsperf* on *its own machine*, ensuring that:

- The machine running *dnsperf* 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 *dnsperf*.

Bandwidth

dnsperf makes significant demands on bandwidth. The machine running *dnsperf* and the server under test should be connected by a network segment with Gigabit Ethernet (or greater) capacity.

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 *dnsperf*, and may skew test results by:

- Dropping packets.
- Locking up.
- Crashing.

Configuring the Server

The nameserver under test should be configured as an authoritative server, serving one or more zones that are similar in size (and number) to those the server is expected to serve in production.

Recursion should be disabled for authoritative servers that support it—otherwise, the server's attempts to retrieve glue information from the Internet during testing will slow the server by an unpredictable amount of time. Recursion can be disabled as follows:

- BIND8 and BIND9—Specify `recursion no;` in the options block.
- BIND8 only—Specify `fetch-glue no;` in the options block.

The Query Input File

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. 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.

Nominum provides a sample query file for ease of testing. 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.

Constructing a Dynamic Update Input File

To test dynamic update performance, *dnsperf* is run with the `-u` option, and the input file is constructed of blocks of lines describing dynamic update messages. The first line in a block contains the zone name:

```
example.com
```

Subsequent lines contain prerequisites, if any are required. Prerequisites can specify that a name may or may not exist, an RRset may or may not exist, or an RRset exists and its RDATA matches all specified rdata for that name and type. The keywords “require” and “prohibit” are followed by the appropriate information. All relative names are considered to be relative to the zone name. The following lines show the 5 types of prerequisites:

```
require a
require a A
require a A 1.2.3.4
prohibit x
prohibit x A
```

Subsequent lines contain records to be added, records to be deleted, RRsets to be deleted, or names to be deleted. The keywords “add” or “delete” are followed by the appropriate information. All relative names are considered to be relative to the zone name. The following lines show the 4 types of updates.

```
add x 3600 A 10.1.2.3
delete y A 10.1.2.3
delete z A
delete w
```

Each update message is terminated by a line containing the `send` command:

```
send
```

Running Tests

To run *dnsperf* tests, include the "-d" (data file) and "-s" (server) options in the command string, as shown in the following example:

```
# dnsperf -d input_file -s server
```

Monitoring the Test

The output of *dnsperf* is mostly self-explanatory. Pay attention to the number of dropped packets reported; when running the test over a local Ethernet connection, the number of dropped packets should be zero. If one or more packets has been dropped, there may be a problem with the network connection. In that case, the results should be considered suspect and the test repeated.

The following example shows sample output from the **`dnsperf`** command:

```
DNS Performance Testing Tool
Nominum Version 2.0.0.0.d
[Status] Command line: dnsperf -d in -l 10
[Status] Sending queries (to 127.0.0.1)
[Status] Started at: Thu Jan 26 15:18:06 2012
[Status] Stopping after 10.000000 seconds
[Status] Testing complete (time limit)
Statistics:
Queries sent: 370539
Queries completed: 370539 (100.00%)
Queries lost: 0 (0.00%)
Response codes: NXDOMAIN 370539 (100.00%)
Average packet size: request 27, response 71
Run time (s): 10.000463
Queries per second: 37052.184484
Average Latency (s): 0.001693 (min 0.000048, max 0.036055)
Latency StdDev (s): 0.001181
```

Measuring Latency

The average latency is printed in the statitics output; it takes into account both successes and failures. Note that requests that received no response at all will not be included in the latency graph—this may unfairly

skew the average latency in favor of servers that drop requests (or respond with an error later than the *dnsperf* timeout) over those from which an error response is received.

TSIG Support

When the "`-y`" option is used, TSIG records are added to all requests. The command-line option specifies the key *`name`* and *`secret`*, where the *`secret`* is encoded in base64.

Throttling Support

The "`-Q`" option limits the numbers of requests per second to a specific number, which is computed during the entire run of *dnsperf*.