1

dhcperf

DHCP performance testing provides a means of predicting server behavior under load and verifying server performance. The most important elements in a DHCP performance test tool are:

- *Accuracy*—The tool's results should correlate strongly with the server's "real-world" performance.
- *Reproducibility*—Successive testing runs with the tool should produce strongly similar results.
- *Simplicity*—Good results should not be dependent upon configuration options, and the tool should be easy to use and understand.

Definitions

dhcperf uses the following definitions of *latency*, *throughput*, *load* and *failure*.

Latency and Throughput

Latency is the amount of time required to service a DHCP transaction.

Throughput is the number of transactions that may be serviced during a unit of time.

Throughput is a far more accurate measure of DHCP performance than is latency. Accurate DHCP performance tools measure throughput.

Load

Transaction load is defined as *transactions per second*.

For testing, transactions should be created with an event time distribution—that is, at a load of 10 transactions per second, a transaction should be created every 0.1 seconds.

For multi-step transactions like a four-way handshake, only the initial packet in the transaction should be rate-limited—subsequent packets should be sent as soon as an appropriate server response arrives.

Failure

Failure is defined as a server failing to respond to *0.1 percent or more of all messages it receives*.

In the real world, DHCP clients resend messages to which no response is received, and a DHCP server can fail to respond to some transactions without adversely affecting network operations. However, a dropped transaction results in an impaired user experience, and an accurate DHCP performance testing tool must take that into account.

dhcperf

dhcperf is a tool for testing DHCP server performance.

Command Synopsis

```
dhcperf [ --client-creation-load transactions ] [ --clients clients ]
   [ --discover ] [ -h | --help ] [ --one-discover ] [ -p | --port port ]
   [ --progress ] [ -q | --quiet ] [ --renew ] [ --retain-leases ]
   [ -s | --server server ] [ --test-duration seconds ]
   [ --test-load transactions ] [ --usage ]
```

Options

client-creation-load

```
--client-creation-load transactions/second
```

Specifies the load to use when creating clients for DHCPRENEW tests. By default *dhcperf* uses an adaptive load detection algorithm when creating clients. Specifying a constant load with `--client-creation-load` can speed up client creation.

If you are unsure whether or not you need to use this option, it's best to leave it unset.

clients

```
--clients clients
```

Specifies the number of clients to create when running the *peak-renew* test. This parameter has no effect when running other tests.

The default is 20,000 clients.

discover

```
--discover
```

Selects the DHCPDISCOVER test mode. In this mode, *dhcperf* performs load testing using four-way DHCP handshakes.

help

```
-h, --help
```

Prints detailed usage information.

one-discover

```
--one-discover
```

Selects the one-discover test mode. In this mode, *dhcperf* performs a single four-way handshake.

port

```
-p, --port port
```

Sets the UDP port upon which *dhcperf* sends and receives DHCP messages.

The default is 67—*dhcperf* emulates a DHCP relay agent and uses the "DHCP server" port.

progress

```
--progress
```

Prints test progress information every second.

quiet

```
-q, --quiet
```

Instructs *dhcperf* to print minimal information while running.

renew

```
--renew
```

Selects the DHCPRENEW test mode. In this mode, *dhcperf* performs load testing using client DHCPRENEW requests.

retain-leases

```
--retain-leases
```

Prevents *dhcperf*, when running the *one-discover* and *peak-discover* tests, from issuing a DHCPRELEASE for each acquired address.

server

```
--server server
```

Selects the server(s) to which synthetic broadcast traffic should be sent. This option may be specified multiple times to select multiple servers.

Broadcast packets from synthetic clients are sent to all servers, while unicast packets are sent only to one server.

ggtest-duration

```
--test-duration seconds
```

Specifies the number of seconds for which *dhcperf* is to run test probes.

Note that this is *not* the total duration of the *dhcperf* run (which will be substantially higher), but the *duration of the test run* at each tested transaction load.

The default test duration is two minutes (120 seconds). The minimum test duration is 15 seconds. Longer tests will produce more accurate results.

When combined with `--test-load`, `--test-duration` sets the total length of time for which *dhcperf* should maintain the given load.

test-load

```
--test-load transactions/second
```

Specifies a constant load to use for testing. By default, *dhcperf* attempts to determine the peak sustainable transaction load of the tested server(s).

When `--test-load` is given, *dhcperf* instead runs a single test at a fixed load for a fixed furation (specified by `--test-duration`) and reports the percentage of successful transactions.

usage

```
--usage
```

Instructs *dhcperf* to print a brief usage message.

Test methodology

dhcperf creates synthetic DHCP load by emulating clients operating through a DHCP relay agent.

The synthetic clients created by *dhcperf* include a `parameter-request-list` option (DHCP option 55) in their requests for the `subnet-mask`, `router`, `domain-name-server`, and `domain-name` options (DHCP options 1, 3, 6, and 15).

dhcperf attempts to find the peak sustained load at which a DHCP server or servers can operate.

"Load" is defined in transactions per second, where transactions are created in an even time distribution. For example, at a load of 10 transactions per second, *dhcperf* will create a new transaction every 100 milliseconds.

When testing multi-step transactions (e.g., four-way handshakes), only the initial packet in the transaction is rate-limited. Subsequent packets are sent as soon as the appropriate server response arrives.

The "peak load" is defined as the maximum load at which no more than 0.1% transactions fail to complete successfully. While a DHCP server may operate successfully even at far higher failure rates (since DHCP clients will resend packets that receive no response), packet loss will result in an impaired user experience. As such, *dhcperf* is relatively unforgiving of dropped transactions.

A transaction is considered to have failed when a packet sent by one of *dhcperf*'s synthetic clients goes for four seconds without a response. No attempt is made to resend the packet.

In order to test sustained load, *dhcperf* will run load tests for a configurable length of time, controlled by the --test-duration option. Note that the performance of some DHCP servers is limited by intermittent behavior, such as garbage collection or database synchronization. As such, care should be taken to run tests for long enough to accurately measure sustained performance. The default of two minutes is a reasonable minimum.

Peak performance is determined by performing a number of successive test runs at different loads, increasing the load after each successful test and decreasing it after each failed test. The exact procedure is as follows:

1. Begin with a minimal load of one transaction/second. Rapidly increase load until failures are detected. The failure point is defined as the initial "high water mark" for the test. The high water mark is the lowest transaction load known to produce failures.

2. Define the initial "low water mark" to be 0 transactions/second. The low water mark is the highest transaction load known to produce less than a 0.1% failure rate.

3. Repeatedly perform test runs of a fixed duration (set by the `--test-duration` option) with a transaction load set to the middle of the range between the high and low water marks.

   Transactions are created with an even distribution. For example, if the transaction load is 1000 transactions per second, one transaction will be created every millisecond. If a test run fails, the high water mark is adjusted to the run's transaction load. If a test run succeeds, the low water mark is adjusted.

4. When the low water mark is within 5% of the value of high water mark, complete testing. The final low water mark is the peak performance (highest transaction load known to be sustainable without errors) of the server.

Tests

Test Modes

dhcperf provides two main test modes:

- The `peak-discover` test analyzes performance of four-way handshakes, and `peak-renew` that of address renewals.

- The `one-discover` mode performs a single four-way handshake, as a quick means of testing server availability.

Select a test mode using the `--discover`, `--renew`, and `--one-discover` options.

NOTE All tests require that a server be specified with the `--server` option. Multiple servers may be specified with multiple instances of the `--server` option, for testing failover configurations.

---

peak-discover

The peak-discover test performs a peak performance test (as described above) for clients performing initial four-way handshakes. (A four-way handshake consists of a DHCPDISCOVER message from the client, a DHCPOFFER from the server, a DHCPREQUEST from the client, and a final DHCPACK from the server.) After each successful four-way handshake, clients will issue a DHCPRELEASE for the acquired address.

Issuing a DHCPRELEASE for each lease acquired requires the server to handle three requests per client rather than two, and will consequently result in a lower reported peak performance. The `--retain-leases` option will suppress the DHCPRELEASE. Using this option will require that the DHCP server be configured with sufficient leases to serve all the clients created over the duration of the test.

```
# dhcperf --server host --discover
```

peak-renew

The `peak-renew` test performs a peak performance test (as described above) for clients renewing addresses. This test begins by creating a number of clients (configurable with the `--clients` option) and acquiring addresses for them. It then proceeds with the peak test, rotating renewal requests among this pool of clients.

The number of clients created must be large enough to provide one client for each simultaneous transaction created during the test. The exact number will differ depending on the performance characteristics of the server configuration being tested.

If the pool of clients is too small, the test will not be able to create a sufficient number of simultaneous transactions. In this case, the test will abort with an error message.

Some DHCP servers optimize handling of rapid renewals of the same lease. For these servers, the peak-renew test will report unnaturally high performance values unless the number of clients is large enough that no one client renews twice within the optimization threshold.

```
# dhcperf --server host --renew
```

one-discover

The `one-discover` test creates a single synthetic client which attempts to perform a four-way handshake. After acquiring an address, the client issues a DHCPRELEASE and *dhcperf* exits. This test is useful for checking the health of a DHCP server.

If the `--retain-leases` option is given, the DHCPRELEASE will not be issued.

```
# dhcperf --server host --one-discover
```

Example Testing Scenarios

The following example testing scenarios assume a situation in which DCS is passing out addresses from the fictitious address space of `10.0.0.0/16`.

First, create the following objects in Nominum DCS, using the *nom_tell* Command Channel client in interactive mode:

```
# nom_tell dcs

dcs> ccdb method=create objtype=network name=realNetwork
   netaddr=192.168.0.9/29

dcs> ccdb method=create objtype=network name=fakeNetwork
   netaddr=10.0.0.0/16

dcs> ccdb method=create objtype=sharednet name=sharedNetwork

dcs> ccdb method=add_network objtype=sharednet name=sharedNetwork
   network=realNetwork

dcs> ccdb method=add_network objtype=sharednet name=sharedNetwork
   network=fakeNetwork

dcs> ccdb method=create objtype=pool name=pool1
   ranges=( (10.0.0.1, 10.0.255.254) )
```

After creating these objects, run the tests as described.

one-discover

```
# dhcperf -server 192.168.0.1 --one-discover

Acquired address:10.0.0.1
```

peak-discover

```
# dhcperf --server 128.177.198.19 --discover --clients 253
   --release-leases

Beginning DHCPDISCOVER load test.
Initial probe complete: High-water mark is 209 clients/second.
Preparing for next test run.
Beginning test run: 104 clients/second for 120 seconds.
Succeeded: 0/12480 clients failed.
Preparing for next test run.
Beginning test run: 156 clients/second for 120 seconds.
Stopping run after 19 seconds; 19/2777 clients failed.
```

```
Preparing for next test run.
Beginning test run: 130 clients/second for 120 seconds.
Succeeded: 0/15600 clients failed.
Preparing for next test run.
Beginning test run: 143 clients/second for 120 seconds.
Stopping run after 19 seconds; 20/2594 clients failed.
Preparing for next test run.
Beginning test run: 136 clients/second for 120 seconds.
Succeeded: 0/16320 clients failed.
Preparing for next test run.
Beginning test run: 139 clients/second for 120 seconds.
Succeeded: 0/16680 clients failed.
139 clients/second.
```

peak-renew

```
# dhcperf --server 128.177.198.19 --renew --clients 10000
   --release-leases

Creating 10000 clients for renew test.
10000 clients available
Beginning DHCPRENEW load test.
Initial probe complete: High-water mark is 709 clients/second.
Preparing for next test run.
Beginning test run: 354 clients/second for 120 seconds.
Succeeded: 0/42480 clients failed.
Preparing for next test run.
Beginning test run: 531 clients/second for 120 seconds.
Stopping run after 6 seconds; 68/1844 clients failed.
Preparing for next test run.
Beginning test run: 442 clients/second for 120 seconds.
Stopping run after 14 seconds; 55/5884 clients failed.
Preparing for next test run.
Beginning test run: 398 clients/second for 120 seconds.
Stopping run after 34 seconds; 54/13213 clients failed.
Preparing for next test run.
Beginning test run: 376 clients/second for 120 seconds.
Succeeded: 0/45120 clients failed.
Preparing for next test run.
Beginning test run: 387 clients/second for 120 seconds.
Stopping run after 56 seconds; 51/21413 clients failed.
376 clients/second.
```