SIDN Server Measurements

Yuri Schaeffer[1], *NLnet Labs*

NLnet Labs document 2010-003 July 19, 2010

1 Introduction

For future capacity planning SIDN would like to have an insight on the required resources for hosting the *.nl* zone. We performed a series of simulations to test memory, CPU and bandwidth consumption on a limited set of parameters.

The tests are repeated for an increasing percentage of delegations with a DS record in the zone (0%, 10%, and 20%). These parameters are tested with 1, 10, and 25 times the current query load on the *.nl* server.

SIDN provided three zonefiles, a packet trace, and a server. The experiments ran on NLnet Labs's test environment. A description of this setup is given in Section 1.4. Sections 2 to 5 describe the experiment and end with a conclusion in Section 6.

Revision 2: In addition, Section 7 describes differences between Linux' and FreeBSD's network performance for DNS.

1.1 Zonefiles

SIDN constructed three zonefiles based on the current *.nl* zone. It has originally about 4 million delegations, and is increased with 2 million synthetic delegations. This is the expected size for 2013. To prevent skewed results, the added delegations have the same average label and name length as the original delegations. In the three versions of the zone 0, 10, or 20 percent of the delegations get one or more DS records. Which delegation gets DS records is picked random for each file. These delegations get between 1 and 4 DS records from an uniform distribution.

All versions are signed using a 2048 bit KSK and a 1024 bit ZSK. NSEC3 will be used with 5 hash iterations and a salt length of 8.

1.2 Packet Trace

The packet trace is captured from one of SIDNs servers in February 2010. It contains 75 minutes of DNS traffic. We will use this to simulate a realistic load. Layer 2 and 3 addresses are rewritten to suit our test setup, the rest of the messages remain unmodified.

The query rate as found in the trace is 2496 qps on average, about 1.7 mbps.

[1] Yuri@NLnetLabs.nl

1.3 Server

The provided server (HP ProLiant DL360G6) has 2 Intel Xeon CPUs (E5504) with 4 MB cache. Both CPUs have 4 cores and are clocked at 2 GHz. It carries 16 GB of RAM. The operating system is Ubuntu 10.04 with a 64 bit kernel: *2.6.32-22-server*. The server contains 2 on board Broadcom BCM5709 gigabit Ethernet adapters.

1.4 Test Setup

Apart from the SIDN provided server the test setup contains 2 more machines. Both running an up to date 64 bit Debian Testing Distribution. One of these machines will be used to replay the packet trace (henceforth Player), the other will only listen for responses (Listener). These machines are tested to be fast enough to perform sustained replay or capture at the speeds required for our experiments.

Network All machines are connected with one interface to a gigabit Ethernet switch dedicated to these machines only. The tests run on this network to prevent regular network traffic from interfering with the experiments. Each packet in the trace is modified to have the correct source and destination MAC address, and destination IP address. The source IP address however is set to that of Listener, so that Player does not have to take care of handling the responses. A firewall rule on Listener is added to prevent it from sending "icmp-port-unreachable" messages back to the server.

Software The server has both BIND as NSD installed. The BIND version is 9.7.0-P2 (2010-05-19), it is configured with `--enable-threads` and all other options left at their default. NSD is taken from the SVN trunk at r3003 (2010-06-15) with all options default. Libssl 0.9.8 is installed from the Ubuntu repository.

2 CPU Usage

The server (Section 1.3) has a total of 8 cores at its disposal for running the DNS software. Using all cores for the daemon is not necessarily the most optimal solution. Before we do our actual testing we determine the amount of processes or threads we will use for each daemon. This optimum is found by searching for the highest throughput for each number of cores used (1 to 8) in terms of bytes per second.

NSD has the highest throughput using 3 server processes, although with 2 server processes it is just slightly slower. BIND profits from each added thread, it reaches it's maximum throughput at 8 threads. There is no limitation for the amount of processes NSD can handle. When adding more processes the usage of each (by NSD used) core will decrease. This is likely caused by the network stack of the operating system where only one single process can use a socket at once. It should be noted that both NSD and BIND have about the same maximum

throughput at this point, about 250 Mbps responses output with about 60 Mbps queries as input.

Additionally we ran the same test with an echo daemon which got forked after the bind to port 53. The echo daemon runs in a tight loop where the received message is directly sent back to the source, no computation is done. Using a single process gave us the highest throughput, 130 Mbps input and 130 Mbps output. For each added process the performance of the echo daemon decreases.

For all further tests we configured NSD to use 3 processes and BIND to use 8 threads.

2.1 Measurement

For both NSD and BIND we replay the original query trace at 1, 10, and 25 times the original speed. We have logged the accumulated idle percentage of all processors and derived CPU usage from this. This way we have user and system usage captured in a single value.

All tests ran for about 3 minutes (we have 75 minutes of data and a 25× speedup) and are sampled about 50 times. The percentages given are for all eight cores, this means that for NSD there is less room for growth than seems to be at first glance.

	Replay speed		
Percentage DS	1×	10×	25×
0%	5.1 (0.8)	14.4 (1.4)	28.8 (1.8)
10%	5.1 (0.8)	14.9 (1.4)	29.0 (2.2)
20%	5.7 (1.1)	15.2 (1.3)	29.7 (1.8)

Table 1: CPU usage for NSD in percent, standard deviation between parentheses.

	Replay speed		
Percentage DS	1×	10×	25×
0%	5.8 (0.6)	36.0 (2.0)	73.4 (1.6)
10%	5.5 (0.3)	36.2 (2.4)	73.7 (1.8)
20%	5.5 (0.5)	36.0 (2.6)	72.8 (1.4)

Table 2: CPU usage for BIND in percent, standard deviation between parentheses.

Table 1 and Table 2 show CPU usage for both programs. The figures are skewed as the programs use a different amount of cores (3 and 8) with the CPU usage given for all 8 cores. When correcting[1] for this, the values are very similar. Both programs seem to scale the same way on this hardware.

[1]NSD is configured to use 3 out of 8 cores. To get a realistic value one should multiply the NSD values by $\frac{8}{3}$

3 Memory Usage

The memory usage is taken from `/proc/<PID>/smaps`[2]. Table 3 shows the memory usage for both programs as well as the disk size of the zonefile. Note that since both programs are authoritative only and no caching is done, memory usage remains constant over time. Also, the amount of threads or processes has little impact.

Percentage DS	NSD	BIND	Zonefile
0%	2.9 GB	1.5 GB	0.5 GB
10%	4.4 GB	2.1 GB	1.0 GB
20%	5.9 GB	2.7 GB	1.5 GB

Table 3: Memory usage for NSD and BIND, plus the size of the zonefile.

When extrapolating these values to 100 percent signed DS records we would see a memory requirement of 18 GB for NSD and 7.5 GB for BIND.Our test system has 16 GB of memory installed at present. SIDN estimates that in 2013 the zone will be no larger than 6 million delegations and at most 20 percent of those will be signed. Given these estimations and increasing memory capacity over the years. This memory consumption does not have to become a limiting factor in the near future. In certain unlucky conditions NSD can take briefly up to 4 times the normal memory consumption. We do not know how the BIND daemon behaves in the worst case scenario.

4 Bandwidth Usage

For increasing amounts of DS records we measured the bytes on wire. The data is read from `/proc/net/dev`[3]. Not all queries where answered but on average 99.99% succeeded for each test. With each measurement the complete query trace was replayed.

		MB out	
Percentage DS	MB in	NSD	BIND
0%	919 MB	4330 MB	4199 MB
10%	919 MB	4456 MB	4325 MB
20%	919 MB	4371 MB	4240 MB

Table 4: Network usage by replaying whole trace.

[2]We used *ps_mem* which finds the most accurate method available: http://www-.pixelbeat.org/scripts/ps_mem.py

[3]This is the same information *netstat -s* uses, but netstat has a bug not displaying this information correctly.

We have 2 observations about the data in Table 4. The first one is that with 10 percent DS records for both NSD and BIND the total reply size is much larger than for 0 and 20 percent. Which delegations are signed was chosen randomly during generation of the zonefiles and differ between files (The set DS records in the 10% file is not a subset of the 20% file). The delegations might be chosen unfortunately in one of the two cases for the given packet trace.

The second observation is that BIND consistently outputs 131 MB less than NSD (about 3 percent). A sample of 100 responses shows that both daemons return the same answer each time but BIND compresses slightly better.

With these 6 measurements there is no clear trend visible in the reply size. The average response grows but not by much. A possible reason for this is the low amount of NXDomain responses, about 2.9 percent. This type of response will increase the most in size when serving DNSSEC data. Of all the incoming queries in our trace 50 percent had EDNS0 with the DO bit set. Past measurements at the K-root server has shown that NXDomain queries are less likely to have the DO bit set than queries for existing domains[1].

Although one of our zonefiles has 0 percent signed delegations the zone is still signed. Each existing but unsigned delegation will be delivered including a proof that no signatures for that name exist. Those responses will generally be larger than signed delegations, but smaller than NXDomain responses. We can see this in the SIDN provided trace which includes queries as well as responses, the trace is made without DNSSEC enabled on the server. The total response size of the replies in the unsigned variant is 2.0 GB.

5 Spamrun

In addition to a normal increase of traffic at the SIDN server we tested three additional scenarios. Sometimes SIDN observes an unusual amount of NXDomain responses from their servers. This is likely to be a side effect of someone sending large quantities of spam to or from non existing domains. Peaks are seen with 4 times more NXDomain responses than regular responses. With DNSSEC enabled this could potentially increase demand on resources. For non-existing names an authoritative server using NSEC3 needs to perform some additional hashing operations in order to find the *closest encloser* and *next closer*.

This test involves reordering our query trace which causes the timing of the packets to be shuffled as well. We must tell tcpreplay to replay packets at a specific rate (Mbps) to not confuse the program with the mixed timing. The average rate of the original trace for incoming packets is 1.7 Mbps. The three scenarios are:

1. The current bandwidth of 'good' queries and 4 times that bandwidth with 'bad' (NX) queries. The situation which is often seen now (8.5 Mbps, 5× current bandwidth).

2. The current bandwidth of 'good' queries and 24 times that bandwidth with 'bad' queries. (42.5 Mbps, 25× current bandwidth). The same situation as

1., but with a more intense spamrun.

3. A spamrun as seen in the present, like 1., but with 'good' and 'bad' queries equally amplified (42.5 Mbps, 25× current bandwidth).

All scenarios are tested with NSD and the zonefile with 20 percent of the delegations signed. For comparison we replayed the same amount of queries as in Section 4.

scenario	CPU Usage	Traffic out
1	10.4 (1.4)	4338 MB
2	30.1 (2.0)	4364 MB
3	30.6 (1.9)	4338 MB

Table 5: CPU usage and transmitted bytes for the 3 spamrun scenarios using NSD.

For this test the configuration and the zonefile is the same as in previous tests. Thus memory consumption will not change. The CPU usage in these scenarios (Table 5) is similar to Table 1. Again 3 of 8 cores are used for this test. The ratio of NXDomain responses seems to have no influence on performance.

6 Conclusion

We tested the throughput of BIND with up to 8 threads. The throughput increases with every added thread and equals NSD's throughput with 3 processes. Adding more processes for NSD does in fact decrease the throughput. The network stack of the operating system is likely to be the bottleneck for this system.

We have not seen a clear bandwidth increase for an increasing number of signed delegations. However, enabling DNSSEC without signing any delegations does double the required bandwidth.

The ratio of NXDomain responses does not influence the bandwidth in the three tested scenarios. Responses with unsigned delegations will have a similar size increase as NXDomain responses when enabling DNSSEC with opt-out.

Both daemons show no additional CPU load when increasing the number of DS records. Also, in the 'spamrun' scenarios CPU load is similar to normal operation.

Memory usage of both daemons is not likely to form a problem in the foreseeable future. Even for NSD which requires a lot more memory than BIND.

Further Study It seems that the current performance is limited by the network stack. The operating system can not handle packets fast enough to keep NSD (and perhaps BIND) busy. For future testing it might be interesting to try and push performance on this hardware further. We could use an other operating system such as a BSD variant to compare the influence of the network stack.

One could also try to improve performance by using multiple network interfaces at the same time.

7 Epilogue

During the experiments we observed that both BIND as NSD had the same maximum throughput on this hardware but with a different amount of used cores. BIND reached its maximum at 8 cores, NSD reached it using 3 cores. Adding more processes for NSD did not increase performance. This indicates a common bottleneck other than CPU for NSD and BIND.

7.1 Concurrency

On the same hardware an installation of FreeBSD 8.0 was added. Figure 1 Shows NSD with the same *.nl* zone as used in the other tests. NSD is configured with 1 to 8 processes on both operating systems. By replaying the packet trace at various speeds the maximum output was estimated. *note: for FreeBSD, at 7 and 8 cores, the output was very unstable.*

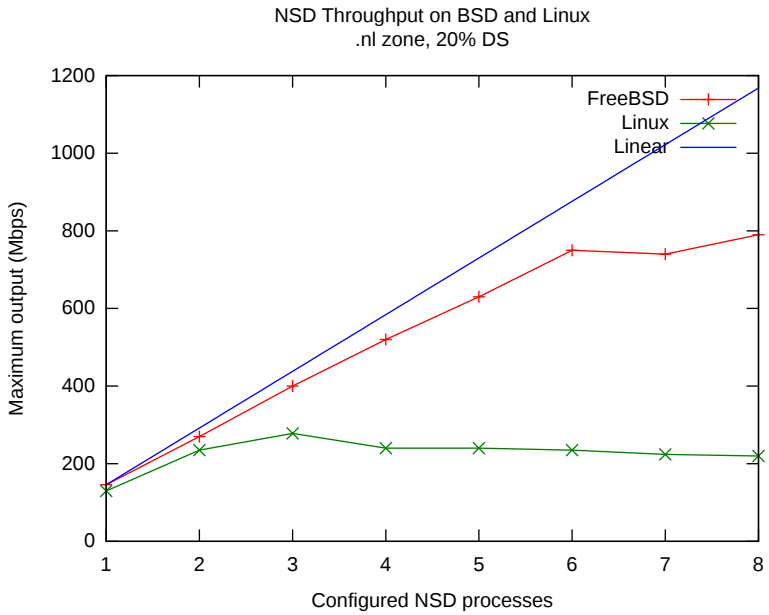

Figure 1: NSD performance on Linux and FreeBSD by used cores. Linear plot for reference on scaling.

With enough cores available NSD's performance triples when switching from Linux to FreeBSD.

For further testing we installed Iperf[4]. The default packet size of Iperf is 1480 Bytes. Using that size we see a bandwidth utilization of around 860 Mbps independent of the amount of cores used by Iperf. This is not very realistic since

[4]http:// iperf.sourceforge.net/

DNS queries are much smaller, usually around 100 Bytes. With 100 Byte packets 280 Mbps is reached at maximum, which is close to NSD's performance on Linux in Figure 1. FreeBSD is better than Linux at handling a large amount of tiny packets.

7.2 Performance Degradation.

In the beginning of the research we determined to maximum output for NSD and BIND on Linux. In this process we noticed that, when offering more traffic than the DNS software could handle, the performance would decrease. FreeBSD showed such behavior to a far lesser extend. In Figure 2 NSD's output for given input on both OS's. Note that for this setup and packet trace the responses are about 5 times larger than the queries, for 30 Mbps input 150 Mbps output is expected.

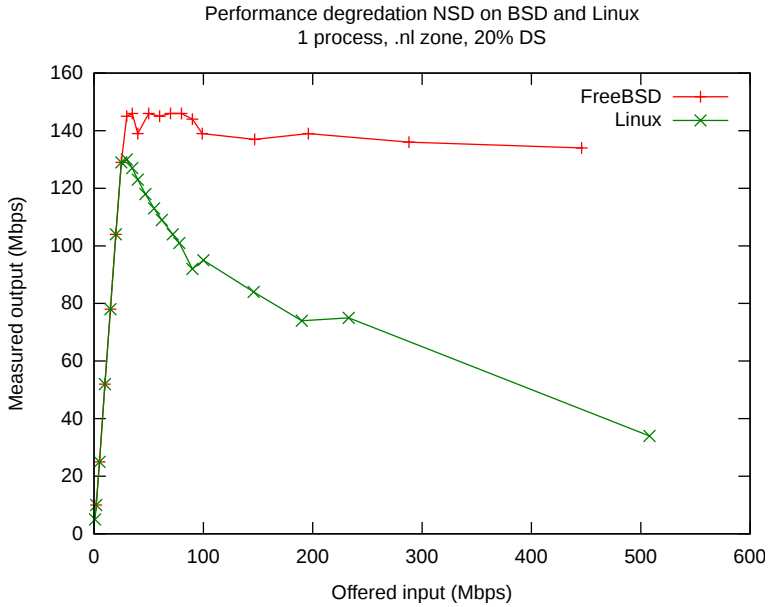

Figure 2: NSD output when offered an excessive amount of input.

NSD is configured to use just 1 process to make sure the CPU bottleneck is hit early. The maximum output using 1 process is 130 to 146 Mbps. Figure 1 shows that on both operating systems the output can be higher given enough cores, and thus the CPU is the bottleneck here. Yet for higher offered bandwidths the performance on Linux decreases. As soon as there are not enough processes ready to accept incoming packets, Linux is to busy handling the excess load or to busy to handle NSD's responses. Either way, letting the bandwidth grow towards the maximum capacity on Linux is a risk. When the maximum capacity is exceeded on Linux, the capacity will decrease. This amplifies the damage done by an attack where the server is overloaded with queries. FreeBSD shows this behavior just slightly.

In retrospect, on this particular machine configured with 3 processes, this

effect will probably not be observed before 29 times the current incoming bandwidth is reached.

References

[1] NLnet Labs, *K-root TCP load measurements*, http://www.nlnetlabs.nl/-downloads/publications/k-root_tcp_measurements.pdf, February 2010.