K-root TCP load measurements

Yuri Schaeffer[1], *NLnet Labs*

NLnet Labs document 2010-001 February 17, 2010

Note: This document is still a draft, the research is not completed.

Abstract

Enabling DNSSEC at the root will affect traffic patterns such as fallback to TCP. This study aims at finding how the current K-root instances are influenced and what they can handle. Two different loadbalancers are used and UDP and TCP query load is measured. We also look at the expected response sizes and bandwidth usage for a signed root zone.

1 Test setup

The test setup consists of four identical Dell R200 Poweredge machines , a router, and a switch. The machines have a single core Intel Celeron 430 CPU clocked at 1.8GHz, with 512KB cache.

Two different routers are used, a Cisco 73xx and a Juniper M7i. Each experiment one of them functions as load-balancer between two of the R200's (referred to as Server1 and Server2) running CentOS 4.8 (2.6.9-89.0.11.ELsmp) and NSD 3.2.3. The servers together with the router represent a K-root instance. The other two machines are running the same operating system and can be used for different roles such as simulate clients from the Internet. Dependent on their role these are called Player or Listener. All machines are connected to an 100Mb/s full duplex Ethernet switch (HP ProCurve 2524) dividing the network in two segments using VLANs. Figure 1 shows the topology of the setup. P1 and P2 represent the players, K1 and K2 the servers, R the router and the small circles the Ethernet switch. The router balances traffic to address `193.0.14.129` between both servers.

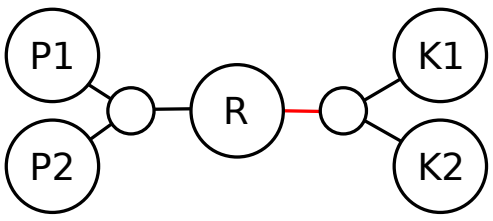


Figure 1: Logical Topology of test setup.

[1]Yuri@NLnetLabs.nl

The servers host the `root`, `arpa`. and `root-servers.net.` zones. We signed the zonefiles using a 2048 bit RSASHA256 KSK and a 1024 bit RSASHA256 ZSK and NSEC.

To simulate real world DNS traffic we used a previously recorded pcap capture file with more than 9 million queries (about 900MB, 1 hour) supplied by RIPE. *tcprewrite*[1] is used to rewrite the captures for our test setup and finally *tcpreplay*[2] version 3.4.3 to resend the queries towards the servers.

2 UDP performance

We are interested to see how many DNS queries a k-root instance can handle, the result can serve as a baseline for further experiments.

2.1 Method

For this test a *pcap* file with UDP IPv4 queries is prepared. The source ethernet address is rewritten to that of Player and the destination address to the router's interface. The destination IP address is rewritten to the address of the servers. The source IP address is set to that of Listener. This causes the responses to flow back to Listener and offloads Player. Furthermore a black hole is introduced to avoid Listener sending back ICMP messages for unexpected traffic: `route add -net 193.0.14.129 netmask 255.255.255.255 reject`

Since a single source address is used all communication is only with one of the servers. Just to be sure, the other server is monitored for packets ending up at the wrong machine. `tcpreplay` is used for replaying the UDP traffic, rate, total amount of packets and the *get time of day* timing method is specified. The tests are performed for the Juniper as well as the Cisco router.

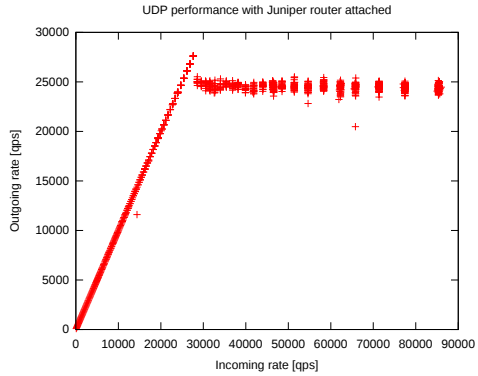
2.2 Juniper results

Figure 2(a) shows at the X-axis the rate of UDP DNS queries send towards the server in queries per second, the Y-axis represents the rate of answered queries in the same unit. Figure 2(b) is the same data but the Y-axis now represents the fraction of queries being answered.

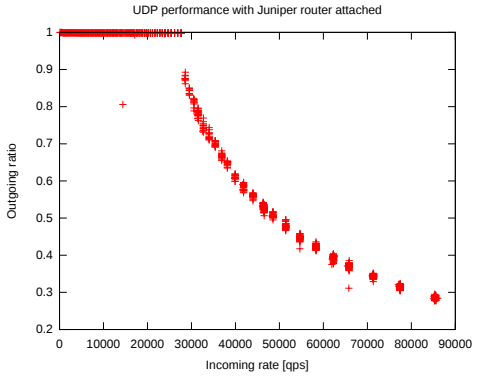
Each plot point is a measurement of around 10 seconds, with each next test the rate of the queries is incremented by 100 qps. At high rates *tcpreplay* does not have enough resolution to output the requested packets per second, this causes the bands in the X-axis. It is believed this does not affect the experiment much although intermediate rates can not be tested.

It is verified that all requests are actually received by the server (tested after the performance drop at 40K qps) but not all answers are being send back. When this happens the performance stays stable at around 25K qps answered, taking the average response size into account (see Section 3) we see this is very close to the network limit of 100 Mbps.

[1] Tcprewrite is part of the tcpreplay suite.
[2] http://tcpreplay.synfin.net/



(a) Number of answered queries per second.

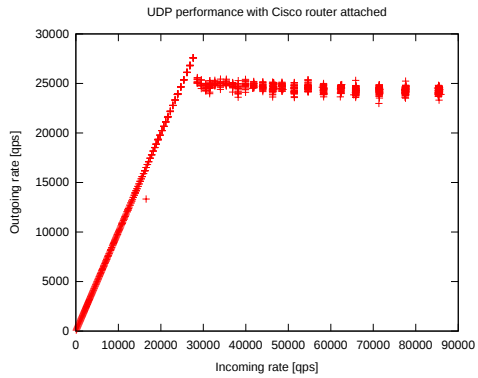


(b) Ratio of answered queries.

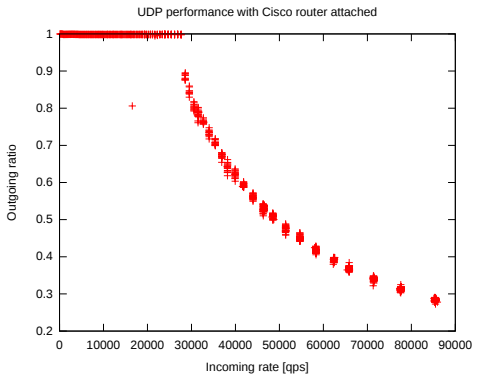
Figure 2: Juniper. Measured from 100 qps to 100,000 qps with steps of 100 qps.

2.3 Cisco results

The same test is repeated with the Cisco router attached, the results are shown in figure 3. Before the network limit is reached ,the figures show no significant degradation (there is a jump from a constant < 0.2 percent to around 11 percent dropout around 28K qps) for both routers. Additionally table 1 shows the sudden leap in the critical point in a numerical fashion.



(a) Number of answered queries per second.



(b) Ratio of answered queries.

Figure 3: Cisco. Measured from 100 qps to 100,000 qps with steps of 100 qps.

The graphs for Juniper and Cisco show no notable difference. Both vendors advertize stateless load-balancing, when performed in hardware it is likely both will be capable doing this at wire speed.

When carefully inspected the graphs for both routers show an outlier at around 15,000 qps where performance drops from 100 to 81 percent. The cause for this anomaly is not looked in to any further.

Queries/s	Ratio answered
...	...
27641.5	0.998271
27657.6	0.998311
28585.3	0.892781
28571.4	0.875466
...	...

Table 1: Successrate around the critical point. Note that this data looks a bit skewed by the banding of tcpreplay*.*

3 Bandwidth and size distribution

The size of DNS responses will increase after signing. The total bandwidth increase depends on the types of queries answered. We will look at the average size, the size distribution, and the types of the responses. Additionally we'll take a look at the EDNS properties of each query.

3.1 Method

The same test setup as with the UDP performance experiment is used, i.e. queries are send from Player and answers received by Listener. To monitor the actual bytes on the wire we use the SNMP capabilities of the switch. *IF-MIB::ifHCInOctets.3* and *IF-MIB::ifHCOutOctets.3* give the total amount of bytes passing interface 3 since restart of the switch. Interface 3 of the switch is connected to the router at the same VLAN as the servers, it is colored red in figure 1

With *tcpreplay* 30K are sent queries at low speed to ensure every one of them can be answered by the server without any problems. Before and after the replay the switch is queried for the port statistics. The experiment is performed for an unsigned root zone and repeated for a signed version of that zone.

3.2 Results

Table 2 contains three tests. The first two used 30,000 queries, one with a unsigned zonefile and one with a signed version of the same file. The last one repeats the signed test but with a tenfold of queries.

For our test set, the responses from the signed zone are on average 29 percent larger than the unsigned zone. With DNSEC, responses for non-existing domains could easily get 8 times as big. Since a large part of the answers are of this kind the increase was expected to be higher.

Because the increase is less than initially expected, we gather some additional information (Note: L-root signing at a later date confirmed these figures). First we used *tcpdump* to obtain the response codes for each answer:

```
tcpdump -nn -r unsigned.pcap -X "udp and port 53"| grep 0x0010 |
```

Test	Direction	Bytes	Bytes/q
unsigned 30K	in	2727724	91
	out	9682243	321
signed 30K	in	2727788	91
	out	12395175	413
signed 300K	in	27285254	91
	out	124924911	416

Table 2: Average size for DNS requests and responses in signed and unsigned zonefile.

```
sed -re "s/.* [0-9a-z]{3}([0-9a-z])  .*/\1/" | sort | uniq -c
```

The results are presented in table 3. Because both samples are not equal in size, the percentage is included as well. Since the responses codes are very similar distributed we can use this data to compare reply sizes. There is a large portion of *NXDomain* responses (43%) and those responses increase more in size than other types thus one *would* expect a large increase in overall response size.

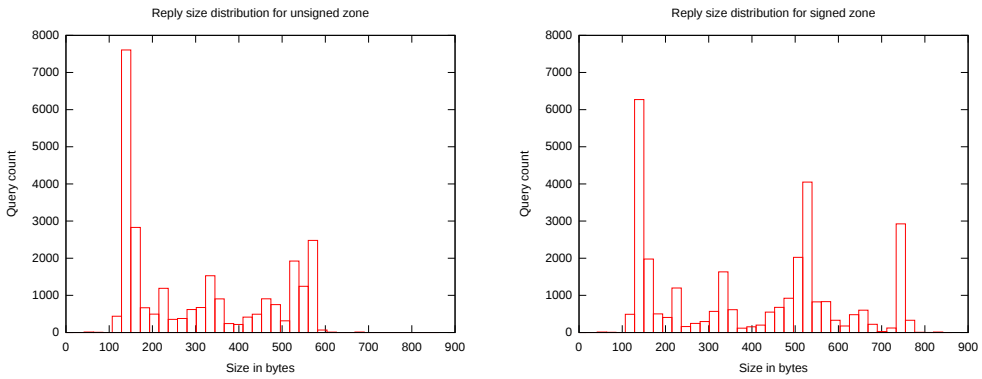
		Number of queries	
Code	Response	Unsigned	Signed
0	No error	15318 (57.2%)	17051 (56.9%)
1	Format error	2	2
3	NX domain	11450 (42.8%)	12904 (43.1%)
4	Not Implemented	10	10
Total		26780	29967

Table 3: NSD responses by code for both an unsigned and a signed zone.

Figure 4 shows the reply size distribution for both a signed and an unsigned zone. This presents us a clue why the average reply size for the signed zone is lower than expected: The first peak in figure 4(a) represents probably mostly *NXDomain* responses. Due to *DNSSEC*'s *denial of existence* these responses should become relatively large in the signed version but in the signed version (figure 4(b)) there is still a large peak around 150 Bytes. *Wolfgang Nagele* of RIPE NCC sugested that the majority of queries leading to *NXDomain* answers do request for *DNSSEC* data (*DO* bit set). We will test this hypothesis.

The *DO* bit is not copied from the query. With the assumption that all answers with both an *OPT* and *RRSIG* record had the bit set in the matching query, the amount of *NXDomain* responses with DNSSEC support can be estimated. The rest of the queries do not support DNSSEC, see table 4.

Indeed the amount of queries with the *DO* bit set is much smaller for answers with an error than for answers without any error. This is likely to cause the smaller than expected size increase. Also, if 60 percent of the answers do not



(a) Unsigned zonefile (b) Signed zonefile

Figure 4: Reply size distribution.

Type	No DNSSEC
NXDomain	70%
No Error	52%
Overall	60%

Table 4: Percentage of queries with no OPT record or DO=0.

increase in size due a lack of *DNSSEC* support, the rest must have grown with a factor 1.73.

The PMTU's advertized in the requests are counted and presented in table 5.

Count	Size in bytes	Percentage
2167133	4096	84%
344402	2048	13%
51564	512	2%
28121	1280	1%
186	2600	
73	768	
18	1024	
4	1420	

Table 5: Advertised PMTU's for sample of 2.6M queries.

4 TCP loadbalancing

All incoming DNS requests at a k-root instance are load balanced over two independent machines. Since the combination of DNS and UDP is completely

stateless this is a trivial thing to do. DNS over TCP is not stateless, the router must make sure the complete connection is directed to precisely one of the servers. Failing to do so would introduce failing or slow connections. Both routers should handle this correctly.

4.1 Method

Both servers log all incoming TCP traffic on port 53. At one of the players a script iterates over the addresses in the range 80.81.192.0/23 (508 unicast addresses). For each of those addresses a virtual interface is created, a *dig* is ran 100 times with the *+tcp* option while monitoring for failed connections, and finally the interface is brought down again.

From both server captures all unique IP source addresses are extracted, these two sets are then compared for overlapping sources. If the router is functioning properly we expect both sets to be completely disjunct.

4.2 Result

Three addresses in the range where in use by other machines, the total amount of unique tests was 505. While testing with the Juniper router 1 more address was in use by on of the players. For both brand routers all connections succeeded and no TCP fragments ended at an other server than the rest of the connection. While testing with the Juniper router Server1 got offered slighly less connection than Server2: 250/254. With the Cisco router this was divided 253/252.

5 TCP performance

Larger DNS replies may introduce problems for some in getting an answer via UDP. An increase in TCP carried queries is to be expected due to TCP fallback mechanisms. Since DNS server software is generally geared towards handling UDP and TCP is significantly harder for the OS, it would be a good to have an idea what a K-root instance is capable of in terms of TCP. The purpose of this measurement is to find how many connections a K-root instance is able to handle reliably.

5.1 Method

To test the amount of concurrent connections a server can handle a way to set up these connections is needed. For this purpose we have created a *C* program using *libevent*[3] version 1.4.13. All network operations are in non-blocking mode. As soon as an operation on a socket would block a new event is scheduled for that socket. The event will trigger as soon as more data is available or a timeout is reached. When the timeout occurs before the connection is fully handled (i.e. answer is received and connection is closed) the connection is considered a failure.

[3] An event driven library to handle many simultaneous file descriptors. http://www.monkey.org/~provos/libevent/

To ensure the Players maintain the connection rate, each event is scheduled at an absolute time calculated from the start of the simulation. A check is done to make sure every connection has enough time between initialization and timeout. The scheduler is granted a 5 percent margin. In case a single connection does not get enough time from the scheduler, the whole test is discarded from the experiment. This ensures we would notice CPU usage becoming a bottleneck on the players.

Although we are not interested in the answer itself, each connection must be handled completely from start till end. Since both the servers and players are on equal hardware, the player and the server will be roughly equally busy. To be on the save side at least two player must be used when testing with a single server.

All tests ran for 30 seconds and where performed twice and averaged. A measurement is done for increasing rates with a resolution of 100 queries per second.

5.1.1 Player modifications

To increase the TCP performance on the players we have enabled `/proc/sys/net/ipv4/tcp_tw_recycle`. Closed connections leave sockets in *TIME-WAIT* state for a while. At the rates we are testing this would imply running out of ephemeral ports in a few seconds. Setting *tcp_tw_recycle* allows the TCP stack to reuse ports in this state.

In Unix a socket is represented by a file descriptor. Linux limits the amount of file descriptors to 1024 per process. In order to maintain more simultaneous connections we stretch this limit in our program using *rlim*. E.g., for 2000 connections per second with a 2 second timeout we need to set the limit to at least 4000 file descriptors.

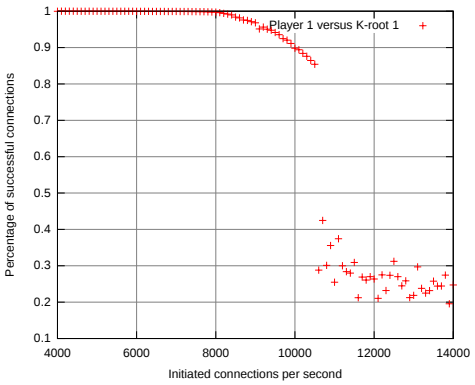
5.2 Results

5.2.1 One player, one server

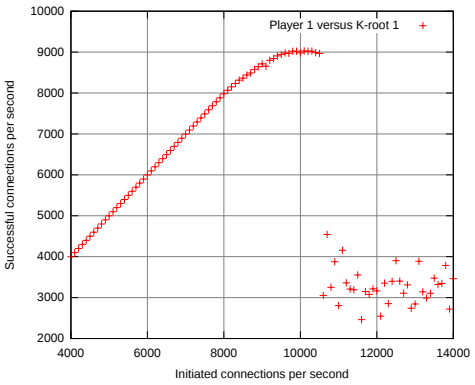
We first start the test using only Player1 sending queries to Server1 (figure 5). The server is able the answer all queries up to 8000qps after which the performance will gradually decrease (1 permill start to fail at 7400qps and 1 percent fails at 8400qps). At about 10600qps the handled connection drops dramatic, and the plot becomes scattered. At this time there is no final answer what exactly causes this effect. The following tests allow us to to find what element of our system is causing the bottleneck.

5.2.2 Two players, one server

The second experiment uses the same replay rate but two players are used instead of one. The effect is that each player has to do only half the work, but the same bandwidth and server load are generated. Compared to figure 5, we have three observations considering figure 6.



(a) Fraction of succeeded connections.



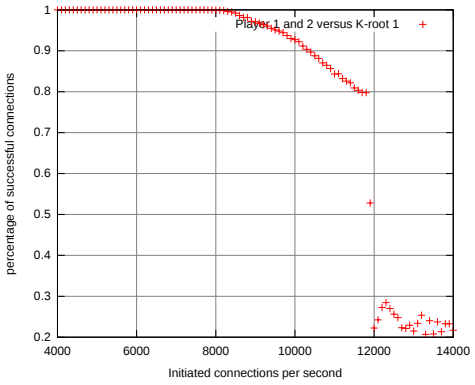
(b) Rate of succeeded connections.

Figure 5: TCP performance using 1 player and 1 server.

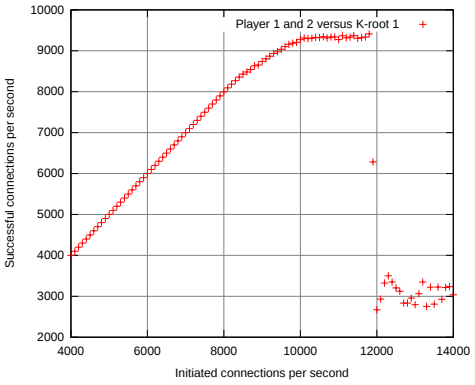
1. The performance degradation starts at around the same point. This implies that either the network or the server are the cause for this characteristic, not the players.

2. The performance decrease is in the new situation somewhat slower. With two players the maximum rate is about 9400qps whereas with one player this is around 9000qps. The behavior is expected since for TCP a player has to do about the same work as the server.

3. The sudden performance drop moved from 10600 to 11900qps. Presumably the bottleneck is shifted from the players to the server here as well. The problem could also be sought in the network performance.



(a) Fraction of succeeded connections.



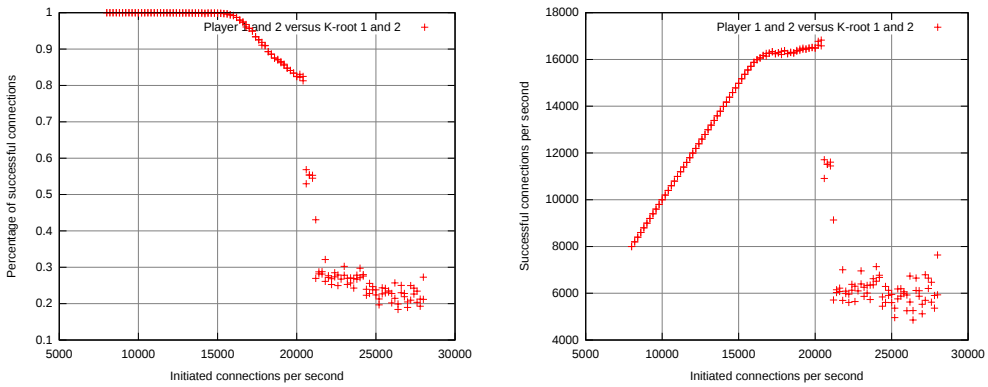
(b) Rate of succeeded connections.

Figure 6: TCP performance using 2 players and 1 server.

5.2.3 Two players, two servers

To test if the network (including the intermediate switch and router) is limiting the performance, the experiment is repeated with two players and two servers simulating in pairs. This are basically two independent experiments performed simultaneously. The results (figure 7) is very much like the first TCP experiment (figure 5) with all values doubled.

This can be used to show the network itself does not obstruct the experiments. Also the players' behavior does not seem to be affected by a busy network.



(a) Fraction of succeeded connections.

(b) Rate of succeeded connections.

Figure 7: TCP performance using 2 players and 2 servers.

The second TCP experiment (section 5.2.2) is the most accurate. The network is shown to be able to handle at least twice the traffic needed for that experiment and the performance drop is not caused by the players.

6 Conclusion

The UDP performance test was capped by the maximum network speed. Before this ceiling was reached the servers did not show any sign of degradation. Both routers performed equally well.

The average reply size for our query sample grows about 29 percent for a signed root zone. For queries leading to *NXDomain* responses our sample has a small portion of *DNSSEC OK* flags. Many of those queries are the result of badly configured machines, which in many cases lack *DNSSEC* support.

Both router vendors advertize stateless load balancing. We see no signs either of them effecting the performance of a K-root instance. Both routers seem to use the IP source address as the only balancing metric. During the load balance test (section 4) as well as during the other tests we have never observed unexpected traffic towards one of the servers.

A single K-root server (i.e. half an instance) can handle at least 8,000 queries per second continuously without any problem. Between that and 12K qps the

success rate lowers to 80 percent. At higher rates the performance drops suddenly towards 2500 to 3500 qps. Different tests have shown the network and the players can handle at least twice as many queries per second. The exact reason of this performance drop is not looked in to any further at this point.

It is good to note this are quite static lab experiments. As many (bogus) queries as possible from the original trace are used to simulate clients. However all the simulated clients where just one hop away. This causes all successful connections to have an extremely short lifespan. Normally these connections would occupy resources on the server for a longer time. In a production environment the performance is likely to be much lower.

7 Future research

After these experiments there is no definite answer for the sudden performance drop in the TCP tests. We suspect the cause is in the network stack of the server. For DNS server operators such as RIPE NCC this isn't much of a problem since the expected TCP load is a few orders of magnitude lower. For NLnet Labs however it is useful to know whether the problem is caused by NSD's own code.

The UDP performance tests are limited by the networking hardware only capable of 100 Mbps. Using a Gigabit network a better view on scalability can be obtained. Also, the graphs show a strange outlier on roughly the same rate for both routers. It could be a coincidence but also an artifact of unexpected behavior.