Telecommunication Systems (2021) 77:643–653
https://doi.org/10.1007/s11235-021-00780-3

Performance evaluation of DNS servers to build a benchmarking system of DNS64 implementations

Gábor Lencse[1] · **Attila Pivoda**[2] · **Keiichi Shima**[3]

Accepted: 20 March 2021 / Published online: 5 April 2021

Abstract
DNS64 is an important IPv6 transition technology that facilitates the communication of an IPv6 only client with an IPv4 only server, which becomes a more and more common scenario. Several different DNS64 implementations exist, and their performance is a relevant decision factor for network operators. RFC 8219 has defined a benchmarking methodology for DNS64 servers, which requires the operation of an authoritative DNS server at 220% of the query rate used for DNS64 benchmarking. In this paper, we aim to build an authoritative DNS server that operates at 2.2 million qps (queries per second) rate, thus it facilitates DNS64 benchmarking up to 1,000,000 qps rate. To that end, we compare the performance of BIND, YADIFA, NSD, Knot DNS and FakeDNS (a special purpose software) to find the best suiting one of them. We fully disclose the details of our measurements including the configuration of the DNS implementations, the usage of our improved software tester called **`dns64perf++`**, and the details of the hardware and software measurement environment in the NICT StarBED, Japan. We perform a series of measurements to examine, how the performance of the tested solutions scale up with the number of the active CPU cores from 1 to 32. Besides their performance, we also measure their memory consumption and zone load time. We present and discuss all the results. In addition to successfully building an authoritative DNS server with the required performance, we also make recommendations, which solutions suit to different special needs.

Keywords Authoritative DNS server · Benchmarking · DNS64 · NAT64 · Performance analysis

1 Introduction

Currently, we are in a transition from IPv4 to IPv6 [1]. Unfortunately, these two versions of the Internet Protocol are not compatible with each other, and IETF has standardized several IPv6 transition technologies to facilitate their cooperation in various communication scenarios [2]. For example, DNS64 [3] servers and NAT64 [4] gateways are used to enable the IPv6 only clients to communicate with IPv4 only servers, which is going to be a more and more common scenario due to the deployment of IPv6 on the client side and the fact that some servers will remain IPv4 only for the foreseeable future. Network operators will need to choose the best suiting DNS64 implementations to their purposes from among several ones, and performance is one of the key decision factors. We have already published a paper about the performance comparison of four DNS64 implementations in 2016 [5]. Then RFC 8219 [6] has defined a benchmarking methodology for IPv6 transition technologies including DNS64 in 2017. The benchmarking procedure for DNS64 servers, which we have described in more details in [7], requires the usage of a high performance authoritative DNS server during the benchmarking of DNS64 servers (we give more details in Sect. 2.1). The first author of this paper had the opportunity to measure the performance of three DNS64 implementations in an RFC 8219 compliant way as a guest researcher in Japan [8]. The tested implementations showed a moderate performance, which is very far away from the performance of the Google public DNS server, which is about 810,000 qps (query per second) on average [9]. Therefore, we set our goal to develop a high performance DNS64 server and to be able to benchmark DNS64 implementations up to *1 million queries per second* rate.

✉ Gábor Lencse
lencse@sze.hu

[1] Department of Telecommunications, Széchenyi István University, 1 Egyetem tér, Győr 9026, Hungary

[2] Wisper s.r.o., 10 Jánošíkova, 94078 Nové Zámky, Slovakia

[3] IIJ Innovation Institute Inc, Iidabashi Grand Bloom, 2-10-2 Fujimi, Chiyoda-ku, Tokyo 102-0071, Japan

The aim of our current effort is *to build a high performance authoritative DNS server* that facilitates the benchmarking of DNS64 implementations up to 1,000,000 qps rate. To that end, we evaluate the performance of different authoritative DNS server implementations BIND, YADIFA, NSD, Knot DNS and FakeDNS (a special purpose software) to find the best candidate for our purpose. We also disclose the details of their configuration as well as the necessary settings of our test environment to be able to receive several millions of packets per second.

The remainder of this paper is organized as follows: In Sect. 2, we summarize all background information, including the highlights of DNS64 benchmarking, as well as the operation of our testing tool and its improvements in a nutshell. In Sect. 3, we introduce the tested authoritative DNS server implementations and disclose their settings. In Sect. 4, we explain the details of the measurements. In Sect. 5, we report and discuss our results. In Sect. 6, we unfold our plans for future research. In Sect. 7, we give our conclusions.

2 Background information on benchmarking DNS64 servers

2.1 Benchmarking methodology for DNS64 servers

We have defined a benchmarking methodology for DNS64 servers in Section 9 of RFC 8219 [6] and elaborated its details in [7]. Now we present only a very brief summary of it. The compulsory test of the DNS64 benchmarking procedure follows the test and traffic setup shown in Fig. 1. This is the "worst case" scenario, when all the following six messages are used:

1. The Measurer subsystem of the Tester sends a query for an IPv6 address ("AAAA" record) of the domain name in question to the tested DNS64 server, which is the DUT (Device Under Test).
2. The DUT sends a query for an IPv6 address of the same domain name to the AuthDNS (Authoritative DNS Server) subsystem of the Tester.
3. The AuthDNS subsystem of the Tester replies with an empty "AAAA" record.
4. The DUT sends a query for an IPv4 address ("A" record) of the same domain name to the AuthDNS subsystem of the Tester.
5. The AuthDNS subsystem of the Tester replies with a valid "A" record.
6. The DUT synthesizes an *IPv4-embedded IPv6 address* [10] using either the NAT64 well-known prefix or a network specific prefix plus the received "A" record, and then it returns the synthesized "AAAA" record to the Measurer subsystem of the Tester.

To eliminate caching, *all different domain names* must be used during the at least 60 s long testing. The tester sends the queries at a constant rate and checks the replies if they arrive within the required timeout time and contain a valid "AAAA" record. If yes, then the test is successful, otherwise it is failed. The DNS64 performance is the highest rate at which the test is successful. In practice, a binary search is used to find the highest such rate. And the binary search is to be executed at least 20 times, and the final result is the median of the at least 20 results, whereas their first and 99th percentiles are used to show the stability of the results.

To certify the Tester (including its both subsystems: Measurer and AuthDNS) for testing up to r rate with t timeout, a self-test must be performed at $2.2*r$ rate and with $0.25*t$ timeout. In short, the rationale for 220% of the rate is that the AuthDNS subsystem has to serve two queries for each query to the DUT, and 10% is the performance reserve. As for the timeout, thus together the half of the time can be used up by the AuthDNS subsystem for serving the two queries, and the other half remains for the DUT. (Please refer to [6] and [7] for further details.)

As for our measurements, the 220% of the query rate means that the requirement for the Authoritative DNS server subsystem is 2,200,000 qps rate to be able to test the DNS64 servers up to 1,000,000 qps rate. As for timeout, we have shown it in [7] that it should be 1 s for the DNS64 servers and thus the timeout should be 250 ms for the Authoritative DNS server used to support DNS64 benchmarking.

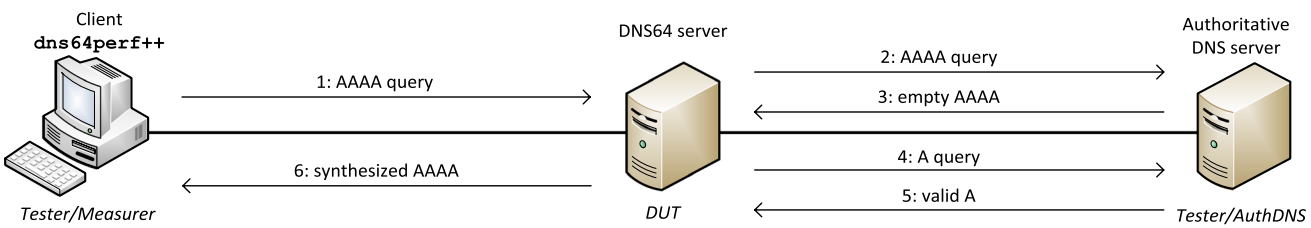

Fig. 1 Test and traffic setup for benchmarking DNS64 servers [11]

2.2 The operation of dns64perf++ in a nutshell

The operation of the original version of **dns64perf++** is described in detail in our open access paper [11], therefore, we only give a very short summary of it.

To be able to generate a high number of requests for all different domain names that can be described systematically, **dns64perf++** uses the following namespace: {0.0.255}-{0.0.255}-{0.0.255}-{0.0.255}.dns64perf.test. During DNS64 tests, these names are resolved to IPv4 address. (For example, 000-001-002-003.dns64perf.test is resolved to 0.1.2.3.) A subset of the namespace to be used can be expressed by the IPv4 address range, which the domain names are mapped to. E.g. 10.0.0.0/8 means a name space of 2^{24} size with the following names: 010-{0.0.255}-{0.0.255}-{0.0.255}.dns64perf.test.

However, for the self-test of the tester, the domain names have to be resolved to IPv4-embedded IPv6 addresses by the DNS server, because **dns64perf++** can send only requests for "AAAA" records. (The domain name in the above example can be mapped to 2001:db8::0.1.2.3.)

The task of the **dns64perf++**program is to perform one elementary test, and a bash shell script is used to perform the binary search and its 20 repetitions.

Originally, **dns64perf++** used only two threads (one thread for sending the queries and another thread for receiving the replies), and it was capable of testing up to 200,000 qps rate [11]. When we tested its accuracy, we have discovered a bug in its timing algorithm, which made it unreliable over about 50,000qps rate, and we have corrected it and rechecked its accuracy [12]. We have also enabled **dns64perf++** for benchmarking the caching performance of DNS64 servers [13], which is an optional test of RFC 8219.

Dániel Bakai, the author of **dns64perf++**, has made different developments on **dns64perf++** so that it can be used for benchmarking up to several million queries per second. The "multiport" feature [14] (latest commit d6fa119 on Oct 8, 2018) includes all the following ones:

1. Usage of 2**n* threads (*n* threads for sending queries and another *n* threads for receiving and processing the replies)
2. Usage of IPv4 as transport protocol. (For DNS64 testing, it has to communicate over IPv6, but we experienced higher self-test performance over IPv4, therefore we used IPv4 in our experiments.)
3. Usage of multiple source port numbers. We need it for RSS (Receive Side Scaling), please refer to Sect. 4.2.

Besides his development, we also added a further feature that pins the threads to given CPU cores using the function `pthread_setaffinity_np()`, to avoid their wandering among CPU cores. Our modified source code is available from [15].

2.3 Size of the name space

The size of the name space was determined to be "/5", which corresponds to $2^{27} = 134{,}217{,}728$ different domain names, and it is enough up to 2,236,962 qps rate, if the test last for the required 60 s.

3 DNS implementations and their settings

In this section, we enumerate the selected authoritative DNS server implementations, give the reasons for their selection and disclose their settings. We have considered only free software [16] implementations for the same reasons as given in Sect. 3 of [5].

3.1 BIND

BIND of ISC [17] is the most popular DNS server implementation, therefore it was a must to be included in our benchmarking.

We have tested its two different versions. The 9.10.3-P4-Debian version, which was included in our Debian distribution, and also its version 9.12.4, which was downloaded in source and compiled with the `--with-tuning=large` option to test their performance difference.

Its configuration was very simple, the following lines were appended to the `/etc/bind/named.conf.local` file:

```
zone "dns64perf.test" {
        type master;
        file "/etc/db.dns64perf.test-AAAA";
};
```

3.2 YADIFA

YADIFA of EURid is told to be one of the lowest memory footprint authoritative DNS server [18]. We have successfully used it to support DNS64 benchmarking and we experienced that it outperformed BIND [8]. For our current project, we used its 2.2.3–6237 version, which was included in our Debian distribution. Its configuration was very simple, we added the following lines to `/etc/yadifa/yadifad.conf`:

```
<zone>
        type              master
        domain            dns64perf.test
        file              db.dns64perf.test-AAAA
        allow-transfer    none
        allow-update      none
        allow-update-forwarding none
</zone>
```

In addition to that, we had to copy the zone file to `/var/lib/yadifa`.

3.3 NSD

The NSD of NLnet Labs was optimized for serving high number of requests per second [19]. We used its 4.1.14 version, which was included in our Debian distribution.

NSD supports the SO_REUSEPORT socket option [20], which improves performance of the network stack on a multi-core computer if `server-count` is set to higher than 1 [21].

Its zone file was configured by adding the following lines to the `/etc/nsd/nsd.conf` file:

```
zone:
    name: dns64perf.test
    zonefile: /etc/db.dns64perf.test-AAAA
```

Unlike BIND and YADIFA, NSD can utilize only a single CPU core, unless the required number of processes is explicitly specified by the `server-count` option. Therefore, we always set this value to the number of active CPU cores.

3.4 Knot DNS

Knot DNS of CZ.NIC is a modern, high performance DNS server [22]. Knot DNS also supports the SO_REUSEPORT socket option, and it does not need to be enabled in the configuration file, as it is enabled by default [23].

Its configuration was very simple, the following lines were added to `/etc/knot/knot.conf`:

```
zone:
  - domain: dns64perf.test
    file: "/etc/db.dns64perf.test-AAAA"
```

3.5 FakeDNS

FakeDNS is a special purpose program developed by Dániel Bakai using the code base of the **`mtd64-ng`** DNS64 server to eliminate the need for an authoritative DNS server, when performing DNS64 benchmarking [24]. FakeDNS does not use a zone file, and it can serve only the name space used by **`dns64perf++`**: it simply takes the information from the first label (e.g. 000-001-002-003) to calculate the appropriate IPv4 address (e.g. 0.1.2.3). As it does not use a zone file, it starts very fast and it uses only a very low amount of memory. Similarly to **`mdt64-ng`**, FakeDNS is also multithreaded. As it did not provide the required performance during our preliminary measurements, Dánial Bakai has developed an experimental feature, called as "moreproc". It starts a separate process for every single CPU core, and a modified version of **`iptables`** is used to distribute the requests among the processes.

FakeDNS was configured in `/etc/fakedns.conf` as follows:

```
have-AAAA 1 # 1=yes
num-servers 32 # number of processes
start-port 1053 # ports used: 1053..1084
start-cpu 0 # CPU cores used: 0, 1, ..., 31
debug no # no debug information is logged
```

As **`dns64perf++`** sends its requests to port 53, we used a special kernel module and **`iptables`** patch prepared by Dánial Bakai, which could rewrite the destination port numbers in the requests and the source port number in the replies. The requests were distributed equally among the **`fakedns`** processes using the **`nth`** mode of the **`statistics`** module of **`iptables`**.

4 Measurements

4.1 Hardware and software environment

The measurements were carried out using the resources of the NICT StarBED, Japan. The measurement setup is shown in Fig. 2. Dell PowerEdge R430 servers were used both as the Tester and as the DUT (Device Under Test). They had two 2.1 GHz Intel Xeon E5-2683 v4 CPUs having 16 cores each, 384 GB 2400 MHz DDR4 RAM and Intel 10G dual port X540 network adapters. They were interconnected by a 10 Gbps VLAN.

Debian Linux 9.6 with kernel version 4.9.0-8-amd64 was installed on both computers.

Based on our benchmarking experience [8] and [25], we switched off Hyper-Threading on both computers to achieve consistent results. Turbo Boost was left enabled on the Tester, but it was switched off on the DUT to avoid the influence of the power budget on the *scale up tests* (see below). Thus the CPU clock frequency of the Tester could theoretically vary from 1.2 to 3 GHz, but power budget limited it to 2.6 GHz, when all cores were used by **`dns64perf++`**. The CPU clock frequency of the DUT

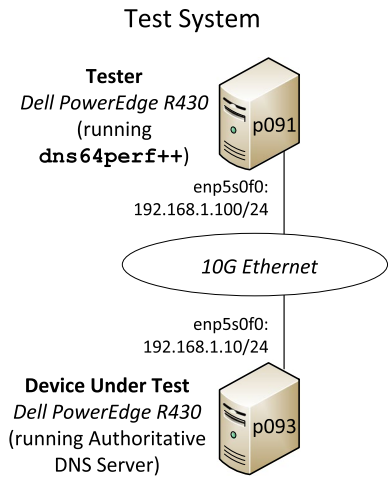

Fig. 2 Measurement setup for benchmarking authoritative DNS servers

could vary from 1.2 to 2.1 GHz. The CPU clock frequency scaling governor was set to "performance" for all active CPU cores on both computers.

We wanted to know, how the performance of each DNS implementation scales up with the number of CPU cores. To set the required number of active CPU cores, we used the `maxcpus=N` kernel parameter to activate *N* number of CPU cores at boot time. We tested with 1, 2, 4, 8, 16 and 32 active CPU cores.

4.2 Receive-side scaling

In the past network interfaces used only a single queue for forwarding the packets from the hardware to the operating system kernel. This solution limited the processing of the received packets to a single CPU core, which became a bottleneck. RSS (Receive-Side Scaling) [26], (also called *multi-queue receiving*) can efficiently distribute the incoming packets among the CPU cores, thus increasing the performance of the networking stack. In practice, the NICs (Network Interface Cards) use a hash value to distribute the incoming packets into the queues, that is, to assign them to the CPU cores. By default, only the source and destination IP addresses are used to compute the hash value. In real life, there are a high number of different source IP addresses can be found in the requests arriving to a DNS server. However, in our case, they were all the same. The Intel X540-AT2 adapters of our servers facilitated the usage of the *four tuple* (source IP address, destination IP address, source port number, destination port number). Therefore, we enabled this feature both on the Tester and on the DUT by the following command:

```
ethtool -N enp5s0f0 rx-flow-hash udp4 sdfn
```

This setting was a prerequisite to be able to test up to 2.2 million query per second rate [27].

4.3 Execution of the measurements

A single execution of the `dns64perf++` program can be used to decide, if the system can serve all the requests at the specified rate during the 60 s long time interval required by RFC 8219. The highest such rate can be determined by a binary search. As for DNS64 measurements, RFC 8219 requires to execute the binary search at least 20 times, and the final result is the median of the 20 results, whereas the first percentile and the 99th percentile are used to express the indices of dispersion, which are the minimum and maximum, if the number of repetitions of the binary search is less than 100.

The binary search and its 20 repetitions were implemented by a bash shell script. The upper limit of the binary search was set to 2,236,962 qps, and the 8.0.0.0/5 name space was used for the measurements.

Independently from the number of active CPU cores of the DUT, always all 32 core were used on the Tester, and both the number of sending and receiving threads were always 16. The number of *source port numbers per sending thread* were set to 2048, to facilitate an even distribution of the arriving packets among the CPU cores.

4.4 Measuring further important quantities

DNS resolution performance is definitely the most important decision factor, but some other quantities may also be important, when selecting the most suitable implementation to support DNS64 benchmarking. Not to interfere with the benchmarking tests, we have measured separately the memory consumption of the different authoritative DNS server implementations as well as the load time of a "/5" size zone file.

5 Results

First, we disclose and evaluate the performance of the different implementations one by one focusing on the scale up of their performance, and then we make a comparison.

In addition to that, we also expose the memory consumption and start up time of the different DNS implementations.

5.1 BIND

The domain name resolution performance of BIND 9.10.3-P4-Debian is shown in Fig. 3. (The columns show the median of the 20 results, and the error bars show the first and 99th percentiles.) As for the scale up of the performance of BIND, there is a strange phenomemon at four cores. Unlike with any other number of CPU cores, at four cores, the performance is not doubled compared to two cores, but it is even lower than that. We have found its root cause by checking the number of started UDP listeners in the `/var/log/syslog` file. At one and two cores, the number of UDP listeners was equal with the number of active CPU cores, however, from 4 cores, the number of UDP listeners was only the half of the number of active CPU cores. Thus the number of listeners beacame a bottleneck at four active CPU cores.

The domain name resolution performance of BIND compiled with the `--with-tuning=large` option is shown in Fig. 4. On the one hand, the single core performance increased drastically from 10,564 to 57,670 qps compared with the previous case, however, on the other hand, there are problems with the scale up. The phenomenon that the performance did not increase from one to two active CPU cores can be explained by the fact that BIND used a single UDP listener in both cases. From four active CPU cores, BIND used one less UDP listeners than the number of active CPU cores. Unfortunately, the performance showed decrease from 16 to 32 cores. The high difference between the first percentile and the 99th percentile of the results from 4 to 32 cores is another issue. (We need to consider the first percentile for DNS64 benchmarking.)

We did not do any further performance tuning and we did not go into deeper analysis of the behavior of BIND, as our purpose was to find an implementation which suited to our needs and BIND was deliberately far from it.

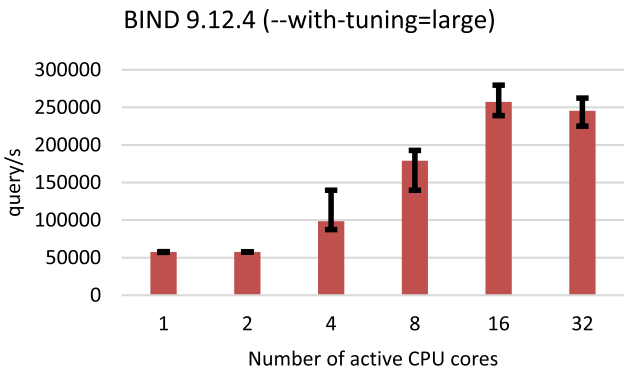

Fig. 4 The domain name resolution performance of BIND 9.12.4 compiled with the `--with-tuning=large` option as a function of the number of active CPU cores

5.2 YADIFA

The domain name resolution performance of YADIFA 2.2.3–6237 is shown in Fig. 5. On the one hand, YADIFA showed a good single core performance (154,800 qps), but on the other hand its performance scaled up very poorly with the number of active CPU cores, which is a fundamental problem in the case of our current multi-core CPUs. (It also showed significantly scattered results at two active CPU cores.) Thus YADIFA is definitely not a candidate for our purposes.

5.3 NSD

We have performed two measurement series with NSD. In the first case, RSS was set to include the source and destination port numbers in the hash value, as described in Sect. 4.2. In addition to that, we have also performed another measurement series excluding the port numbers from the hash to demonstrate the difference.

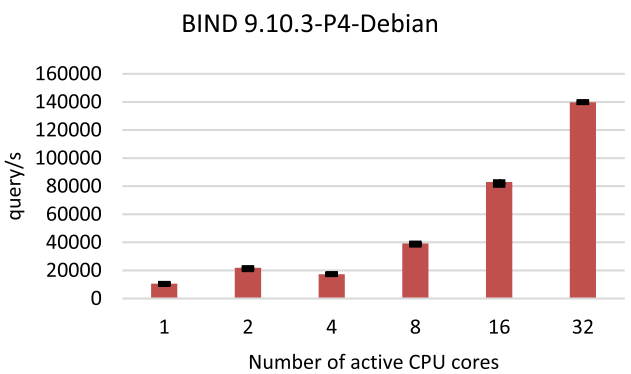

Fig. 3 The domain name resolution performance of BIND 9.10.3-P4-Debian as a function of the number of active CPU cores

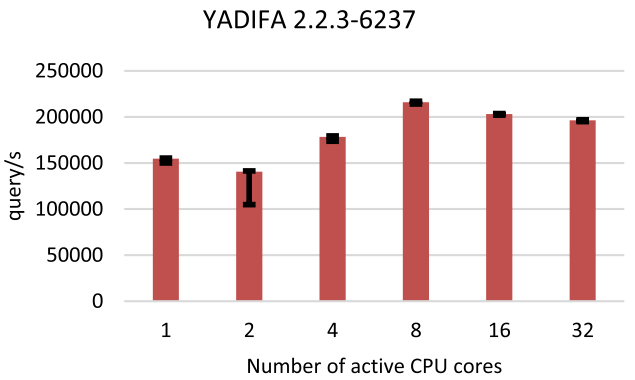

Fig. 5 The domain name resolution performance of YADIFA 2.2.3–6237 as a function of the number of active CPU cores

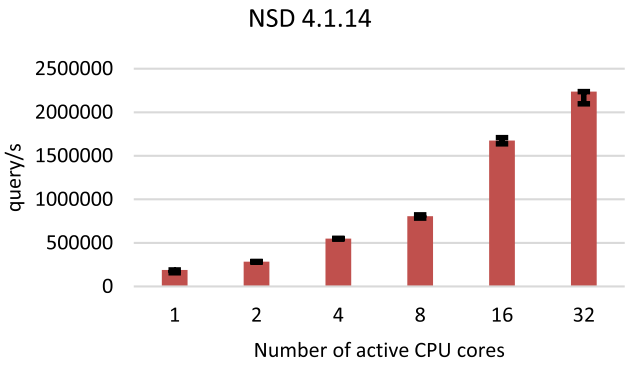

Fig. 6 The domain name resolution performance of NSD 4.1.14 with properly set RSS as a function of the number of active CPU cores. (Its median performance at 32 cores was limited by the size of the zone file.)

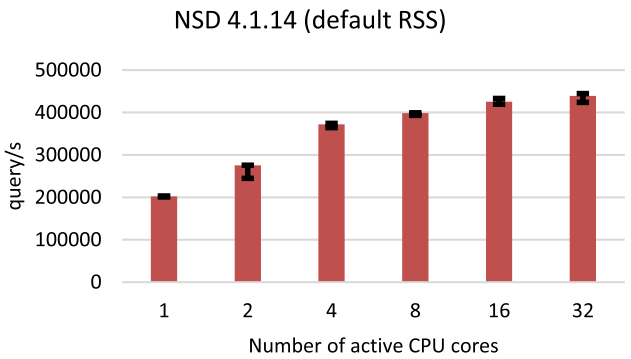

Fig. 7 The domain name resolution performance of NSD 4.1.14 with default RSS as a function of the number of active CPU cores

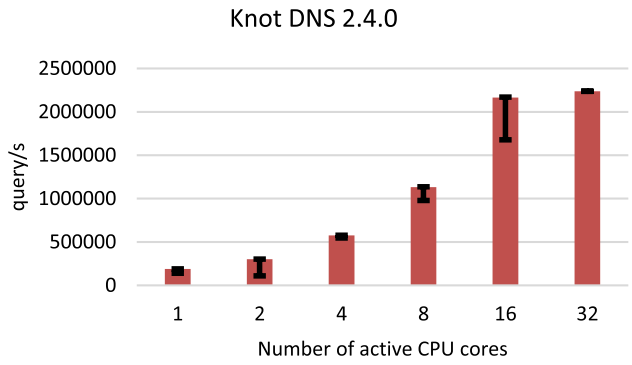

Fig. 8 The domain name resolution performance of Knot DNS 2.4.0 as a function of the number of active CPU cores. (Its median performance at 32 cores was limited by the size of the zone file.)

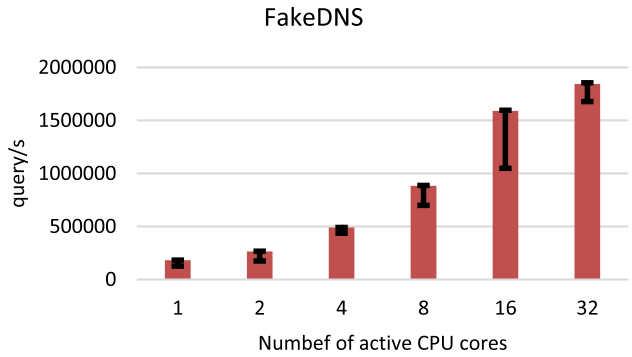

Fig. 9 The performance of FakeDNS as a function of the number of active CPU cores

The domain name resolution performance of NSD 4.1.14 with properly set RSS is shown in Fig. 6. NSD has shown an excellent performance, having both a high single core performance (188,872 qps) and very also a good scale up. We mention that its median performance has reached the upper limit of the binary search at 32 active CPU cores, but its DNS64 benchmarking relevant performance (that is the 1st percentile) was not limited by the size of the zone file.

The domain name resolution performance of NSD 4.1.14 with default RSS is shown in Fig. 7. As expected, its performance was limited by the packet receiving performance of a single CPU core. We note that the CPU core, which processed all the packet arrivals (interrupts) was not excluded from the operation of NSD. (We did not want to do any tuning, our only purpose with this experiment was to demonstrate the need for the properly set RSS.)

5.4 Knot DNS

The domain name resolution performance of Knot DNS 2.4.0 is shown in Fig. 8. It has shown an excellent performance with any number of CPU cores, and its median performance at 32 cores was significantly limited by the size of the zone file, but the first percentile was "only" 2,236,824 qps. Even though its results were significantly scattered at 16 active cores, the first percentile of its results was definitely higher than our targeted 2.2 Mqps at 32 cores.

5.5 FakeDNS

The performance of FakeDNS is shown in Fig. 9. It both showed a good single core performance and it scaled up quite well, though it produced significantly scattered results (especially at 16 active CPU cores).

5.6 Performance comparison

We have compared the DNS64 benchmarking relevant performance of the different solutions, that is, the first percentiles of the results of their 20 tests in Fig. 10.

In general, Knot DNS and NSD have shown the best performance and the third one was FakeDNS. Knot DNS has outperformed NSD at 32 cores, and it was the only

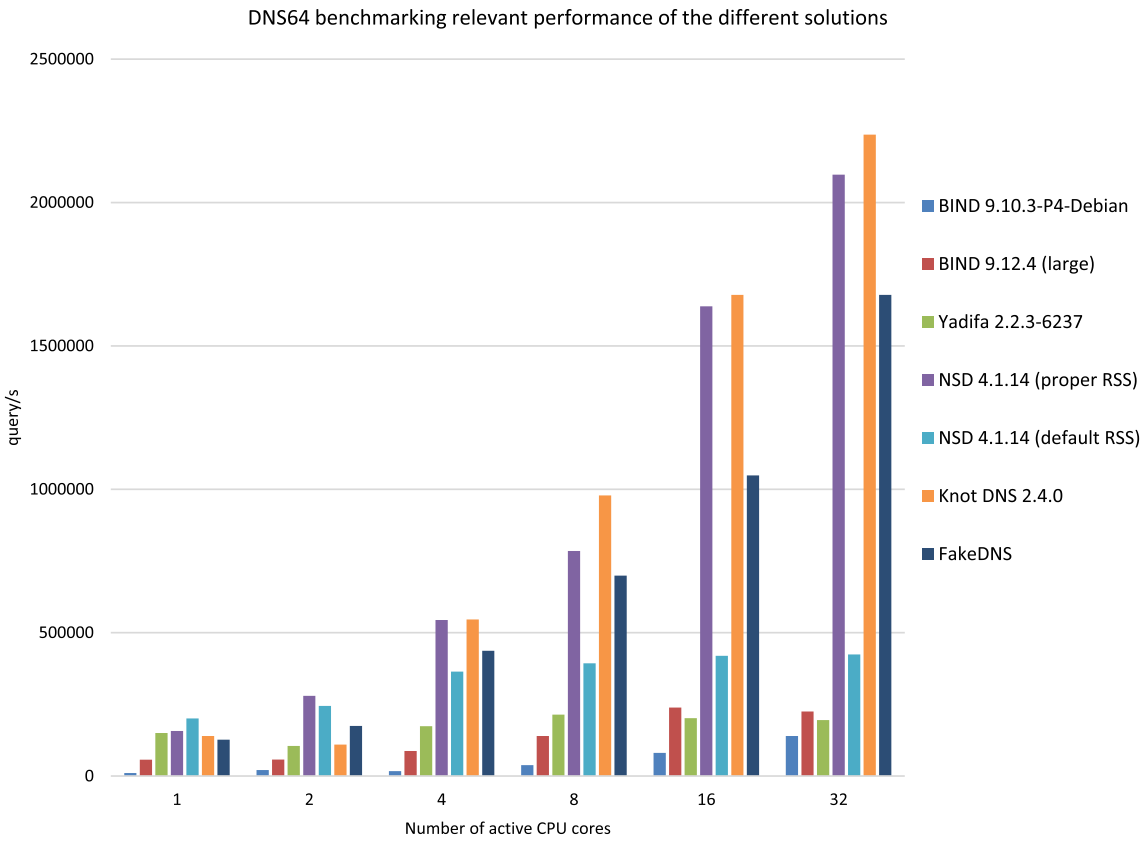

Fig. 10 Comparison of the DNS64 benchmarking relevant performance of the different solutions

implementation that achieved 2.2 Mqps, thus it is our number one recommendation. But NSD was somewhat better at lower number of cores, and NSD with default RSS is definitely the best choice, when only a single CPU core can be used.

Both BIND and YADIFA have shown low performance. BIND suffered from very low single core performance (its bar is hardly visible at a single CPU core in Fig. 10), and the performance of YADIFA scaled up poorly.

For our final recommendation, we need to consider also some other decision factors, which we address in the next subsection.

5.7 Memory consumption and zone load time

The memory consumption of the tested authoritative DNS servers with a "/5" size zone is shown in Fig. 11. YADIFA has shown the lowest memory consumption (23 GB), which complies with the claim of its developers that YADIFA is a low memory footprint server. However it cannot utilize the entries of such a large zone file in a 60 s long test. On the other hand, NSD required the highest amount of memory, 38.1 MB, which is still affordable, if the server has at least 64 MB of RAM.

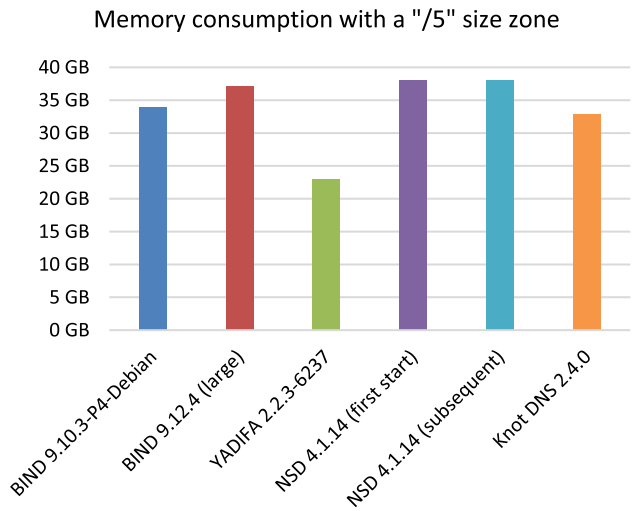

Fig. 11 Comparison of the memory consumption of the different authoritative DNS server implementations. (Lower is better.)

The load time of the tested authoritative DNS servers with a "/5" size zone is shown in Fig. 12. It is important to mention that NSD (with its default settings) prepares a binary zone file at the time of its first start. Using a "/5" size

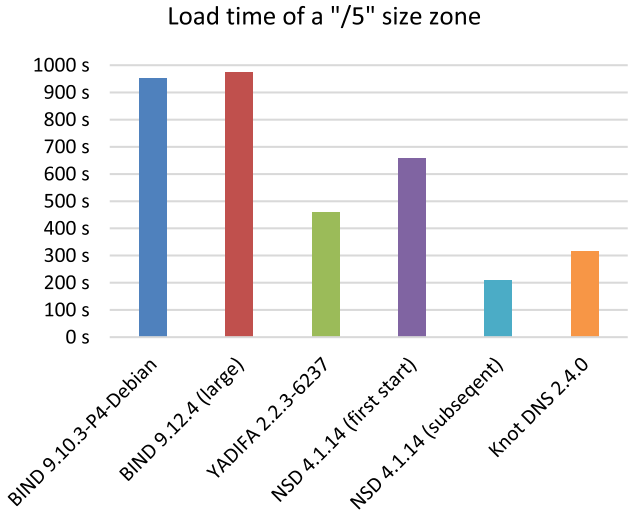

Fig. 12 Comparison of the load time of a "/5" size zone of the different solutions. (Lower is better.)

zone, this binary zone file occupied 84.1 GB disk space, and we stored it on a RAM-drive. The initial load time of NSD was 657 s and the subsequent ones were only 210 s. (If the binary zone file is stored on a hard drive, the subsequent zone loads last much longer due to the lower disk reading speed.) The second fastest one was Knot DNS (315 s).

5.8 Recommendation for DNS64 benchmarking

All in all, we recommend Knot DNS as authoritative DNS server to support DNS64 benchmarking up to 1,000,000 qps rate.

When somewhat lower rates are satisfactory, NSD and FakeDNS can also be good choices. FakeDNS is the best choice, if there is not enough memory in the server, or if very fast start is needed. (Please see further explanation in Sect. 5.7.)

When only a single CPU core is available for the authoritative DNS server, NSD with default RSS can provide the highest performance.

5.9 Discussion of FakeDNS

First of all, we note that FakeDNS is only a byproduct of **`mtd64-ng`**, an experimental DNS64 server [24].

Our result that the performance of FakeDNS is lower than that of NSD or Knot DNS can be explained by the fact that FakeDNS has to convert the 4 times 3 decimal digits of the first label of the DNS query to an unsigned 32-bit integer (IPv4 address), whereas the real authoritative DNS servers can read the IPv4 address directly from the memory. NSD and Knot DNS are good examples that the latter solution

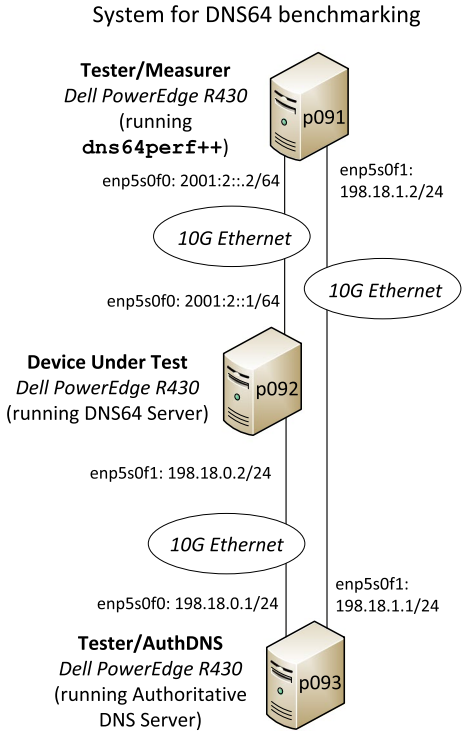

Fig. 13 Measurement setup for benchmarking DNS64 servers

may be faster. BIND and YADIFA are good examples for the opposite case.

We believe that there is room for FakeDNS in DNS64 benchmarking. If the servers used for benchmarking are shipped with only 32 GB of memory, the zone file cannot be loaded into the memory without swapping, but swapping is unacceptable due to its performance penalty. Even if there is enough memory, the fast start of the test system may be attractive, and thus FakeDNS can be a good choice, when its performance is enough.

5.10 Construction of a high performance DNS64 benchmarking system

Following the method of using three devices for DNS64 benchmarking (that is, the Measurer and the AuthDNS subsystems of the Tester are implemented by two separate devices), originally invented and disclosed in Fig. 6. of [7] and also used in Fig. 2 of [8], we have constructed a measurement setup for benchmarking DNS64 servers as shown in Fig. 13. The selected authoritative DNS server implementation is executed by node p093, and it is to be configured as described in Sect. 3 of our current paper. It is important to set RSS so that it considers also the source and destination port numbers for both IPv4 and/or IPv6 an all interfaces used for measurements. We note that the "link" on the right side of the figure serves only the purpose of checking the

performance of the authoritative DNS server, and it is not used during DNS64 benchmarking. (RFC 8219 requires to perform the self-test of the Tester, before it is used for benchmarking DNS64 servers.)

6 Plans for future research

Our plans for future research include the further development of **`mtd64-ng`** our tiny DNS64 proxy [24] and its benchmarking, which became feasible by our current results.

7 Conclusion

We have selected the following free software authoritative DNS server implementations as potential candidates for supporting DNS64 benchmarking: BIND, YADIFA, NSD, Knot DNS, plus a special purpose software called FakeDNS.

We have built a suitable test system for benchmarking the selected solutions, and we have compared their performance, memory consumption, and zone file load time.

We have found that Knot DNS can be used to support DNS64 benchmarking up to 1,000,000 qps rate.

When lower rates are satisfactory, NSD and FakeDNS are also good choices. FakeDNS is the best choice, if there is not enough memory in the server, or if very fast start is needed.

When only a single CPU core is available for the authoritative DNS server, NSD with default RSS can provide the highest performance.

Acknowledgements The experiments were carried out by remotely using the resources of NICT StarBED, 2-12 Asahidai, Nomi-City, Ishikawa 923-1211, Japan. The authors would like to thank Shuuhei Takimoto for the possibility to use StarBED, as well as to Satoru Gonno for his help and advice in StarBED usage related issues.

Authors' contribution GL: Conceptualization, Methodology, Software, Resources, Writing—original draft, Writing—review and editing, Supervision. AP: Investigation, Methodology, Visualization, Writing—review and editing. KS: Writing—review and editing.

Funding Open access funding provided by Széchenyi István University (SZE). Our research received no funding, but we would like to acknowledge that we could freely use the resources of NICT StarBED, Japan, please refer to our acknowledgement at the end of the paper.

Code availability D. Bakai, "Dns64perf++: A C++14 DNS64 Tester", source code. https://github.com/bakaid/dns64perfpp. G. Lencse, Modified source files of dns64perf++, Available: http://www.hit.bme.hu/~lencse/dns64perfpp.

Declarations

Conflicts of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Nikkhah, M., & Guérin, R. (2016). Migrating the internet to IPv6: An exploration of the when and why. *IEEE/ACM Transactions on Networking, 24*(4), 2291–2304. https://doi.org/10.1109/TNET.2015.2453338.
2. Lencse, G., & Kadobayashi, Y. (2019). Comprehensive survey of IPv6 transition technologies: A subjective classification for security analysis. *IEICE Transactions on Communications, E102-B*(10), 2021–2035. https://doi.org/10.1587/transcom.2018EBR0002.
3. Bagnulo, M., Sullivan, A., Matthews, P., & Beijnum, I. (2011). DNS64: DNS extensions for network address translation from IPv6 clients to IPv4 servers. RFC 6147.
4. Bagnulo, M., Matthews, P., & Beijnum, I. (2011. Stateful NAT64: Network address and protocol translation from IPv6 clients to IPv4 servers. IETF RFC 6146.
5. Lencse, G., & Répás, S. (2016). Performance analysis and comparison of four DNS64 implementations under different free operating systems. *Telecommunication Systems, 63*(4), 557–577. https://doi.org/10.1007/s11235-016-0142-x.
6. Georgescu, M., Pislaru, L., & Lencse, G. (2017). Benchmarking methodology for IPv6 transition technologies. IETF RFC 8219.
7. Lencse, G., Georgescu, M., & Kadobayashi, Y. (2017). Benchmarking methodology for DNS64 servers. *Computer Communications, 109*(1), 162–175. https://doi.org/10.1016/j.comcom.2017.06.004.
8. Lencse, G., & Kadobayashi, Y. (2018). Benchmarking DNS64 implementations: Theory and practice. *Computer Communications, 127*(1), 61–74. https://doi.org/10.1016/j.comcom.2018.05.005.
9. Chen, J. K. Google public DNS: 70 billion requests a day and counting. Google Official Blog. https://googleblog.blogspot.hu/2012/02/google-public-dns-70-billion-requests.html.
10. Bao, C., Huitema, C., Bagnulo, M., Boucadair, M., & Li, X. (2010). IPv6 addressing of IPv4/IPv6 translators. IETF RFC 6052.
11. Lencse, G., & Bakai, D. (2017). Design and implementation of a test program for benchmarking DNS64 servers. *IEICE Transactions on Communications, E100-B*(6), 948–954. https://doi.org/10.1587/transcom.2016EBN0007.
12. Lencse, G., & Pivoda, A. (2018). Checking and increasing the accuracy of the dns64perf++ measurement tool for benchmarking DNS64 servers. *International Journal of Advances*

in Telecommunications, Electrotechnics, Signals and Systems, 7(1), 10–16. https://doi.org/10.11601/ijates.v7i1.255.
13. Lencse, G. (2018). Enabling dns64perf++ for benchmarking the caching performance of DNS64 servers. *Journal of Computing and Information Technology, 26*(1), 19–28. https://doi.org/10.20532/cit.2018.1004078.
14. Bakai, D. Dns64perf++: A C++14 DNS64 Tester", source code. https://github.com/bakaid/dns64perfpp.
15. Lencse, G. Modified source files of dns64perf++, Available: http://www.hit.bme.hu/~lencse/dns64perfpp/.
16. Free Software Foundation, "The free software definition", [Online]. Available: http://www.gnu.org/philosophy/free-sw.en.html.
17. Internet Systems Consortium, "BIND: Versatile, classic, complete name server software", [Online]. Available: https://www.isc.org/downloads/bind.
18. EURid, "YADIFA", [Online]. Available: https://www.yadifa.eu/about..
19. NLnet Labs, „NSD", [Online]. Available: https://www.nlnetlabs.nl/projects/nsd/about/.
20. Michael Kerrisk, "The SO_REUSEPORT socket option", March 13, 2013. LWN.net, [Online]. Available: https://lwn.net/Articles/542629/.
21. NLnet Labs, "NLnet Labs Documentation – NSD – nsd.conf.5", [Online]. Available: https://www.nlnetlabs.nl/documentation/nsd/nsd.conf/.
22. CZ.NIC, "Knot DNS", [Online]. Available: https://www.knot-dns.cz/.
23. Včelák, J. "Knot DNS 2.1.0 (final release)", January 14, 2016, [Online]. Available: https://lists.nic.cz/pipermail/knot-dns-users/2016-January/000771.html.
24. Lencse, G., & Bakai, D. (2017). Design, implementation and performance estimation of mtd64-ng a new tiny DNS64 proxy. *Journal of Computing and Information Technology, 25*(2), 91–102. https://doi.org/10.20532/cit.2017.1003419.
25. Lencse, G., & Shima, K. (2020). Performance analysis of SIIT implementations: Testing and Improving the Methodology. *Computer Communications, 156*(1), 54–67. https://doi.org/10.1016/j.comcom.2020.03.034.
26. Herbert, T., de Bruijn, W. Scaling in the Linux Networking Stack. [Online]. Available: https://www.kernel.org/doc/Documentation/networking/scaling.txt.
27. Majkowski, M. How to receive a million packets per second. June 16, 2015. blog.cloudflare.com, [Online]. Available: https://blog.cloudflare.com/how-to-receive-a-million-packets/.

Gábor Lencse received MSc and PhD in computer science from the Buda-pest University of Technology and Economics, Budapest, Hungary in 1994 and 2001, respectively. He has been working for the Department of Telecommunications, Széchenyi István University, Győr, Hungary since 1997. Now, he is a Professor. He has been working part time for the Department of Net-worked Systems and Services, Budapest University of Tech-nology and Economics as a Senior Research Fellow since 2005. His research area is the performance analysis of IPv6 transition technologies.

Attila Pivoda received BSc in electrical engineering from the Széhenyi István University, Győr, Hungary in 2019. He has experience in managing wireless ISP network with MikroTik, Ubiquiti and Cisco devices since 2010, and he is currently working for an ISP company as network engineer. In part time he manages Linux based web hosting servers and he does programming in HTML, PHP, MySQL.

Keiichi Shima is a deputy director at the Research Laboratory of IIJ Innova-tion Institute, Inc. His research field is the Internet, including designing and implementing communication protocols, com-puter networking technologies, computer network security, AI-based anomaly detection, and so forth. He also works as a board member of the WIDE project operating a nation wide research network in Japan.