Benchmarking DNS64 Implementations: Theory and Practice

Gábor Lencse, and Youki Kadobayashi

***Abstract*—RFC 8219 defined benchmarking methodology for IPv6 transition technologies, including DNS64, which is used together with NAT64 to enable IPv6-only clients to communicate with IPv4-only servers. This paper investigates the performances of the most important DNS64 implementations, BIND, PowerDNS, and Unbound as a function of the number of CPU cores using the compulsory tests of RFC 8219. High differences are pointed out both in their single core DNS64 performances and in their scale-up from 1 to 16 CPU cores: whereas Unbound shows the highest single core performance, PowerDNS scales up the best. A serious issue is reported regarding BIND: its DNS64 performance does not increase from 4 to 16 CPU cores at all. A measurement complementary to RFC 8219 is introduced which proves to be useful in the investigation of the issues identified during testing. For the optional tests of RFC 8219, the requirements for the tester are clarified, which enabled us to carry out the tests at the given rates. Finally, a complementary performance metric, *computing power relative DNS64 performance* is introduced, which may provide further information to network operators to support their selection of the DNS64 implementation, which suits the best for their needs.**

***Index Terms*—Benchmarking, DNS64, IPv6 transition, NAT64**

I. Introduction

RFC 8200 [1] raised IPv6, the new version of the Internet Protocol to “Internet Standard” state. Unfortunately, the deployment of IPv6 was rather slow until the latest few years because of several reasons [2], and the public IPv4 address pool was depleted in 2011, before IPv6 could have replaced IPv4. Thus, on the one hand, network operators can not provide public IPv4 addresses to their new customers, but on the other hand, a high number of servers are still available only over IPv4. We contend that the DNS64+NAT64 protocol suite [3] is an adequate solution for this problem. Ref. [4] concludes that “The only actual address sharing mechanism that really pushes forward the transition to IPv6 is Stateful NAT64 (Class 4). All other (classes of) mechanisms are more tolerant to IPv4.” As this citation illustrates, DNS64 [5] is often overlooked and only NAT64 [6] is mentioned and focused on even by researchers. However, the reliable and trustworthy operation of DNS64 is essential, and “the poor performance of the DNS64 server directly influences the users’ quality of experience (QoE)” [7]. RFC 8219 [8] defines a benchmarking methodology for IPv6 transition technologies, including DNS64. In Ref [9], we have discussed the details of the benchmarking methodology for DNS64 servers and disclosed our consideration behind the methodology described in Section 9 of RFC 8219.

In this paper we examine the performance of three DNS64 implementations. Our aim is twofold.

1. We provide network operators with ready to use benchmarking results to support their selection of a DNS64 implementation, which suits the best for their needs.
2. We advance the theory of benchmarking DNS64 servers by three contributions:
    a. We clarify the requirements for the Tester for the optional tests.
    b. We demonstrate that DNS64 benchmarking procedure in Section 9 of RFC 8219, can be complemented by a slightly different procedure using non zero percent loss ratio as acceptance criterion, which may be useful in practical point of view (see Section V.B.2 for more details).
    c. We propose another complementary metric, namely *computing power relative DNS64 performance* of DNS64 implementations.

The remainder of this paper is organized as follows. Section II surveys the available DNS64 performance analysis results and benchmarking tools. Section III gives an introduction to the methodology for benchmarking DNS64 implementations and also clarifies the requirements for the Tester for the optional tests, what is not covered by RFC 8219. Section IV describes the most important details of our measurements including benchmarking environment, DNS64 implementation selection and setup, the most important types of tests performed and also a few hypotheses are set up. Section V presents and evaluates the results, as well as discusses the unexpected results and investigates their possible causes. Section VI proposes measurement procedure and reporting format for the results of the computing power relative DNS64 performance measurements. Section VII is a short case study investigating a specific issue. Section VIII is a brief discussion of our results and disclosure of our future plans. Section IX concludes our paper.

Submitted: March 16, 2018.

G. Lencse is with the Department of Networked Systems and Services, Budapest University of Technology and Economics, Magyar tudósok körútja 2, H-1117 Budapest, Hungary (e-mail: lencse@hit.bme.hu).

Y. Kadobayashi, is with the Laboratory of Cyber Resilience, Nara Institute of Science and Technology, 8916-5 Takayama, Ikoma, Nara, 630-0192, Japan. (e-mail: youki-k@is.aist-nara.ac.jp).

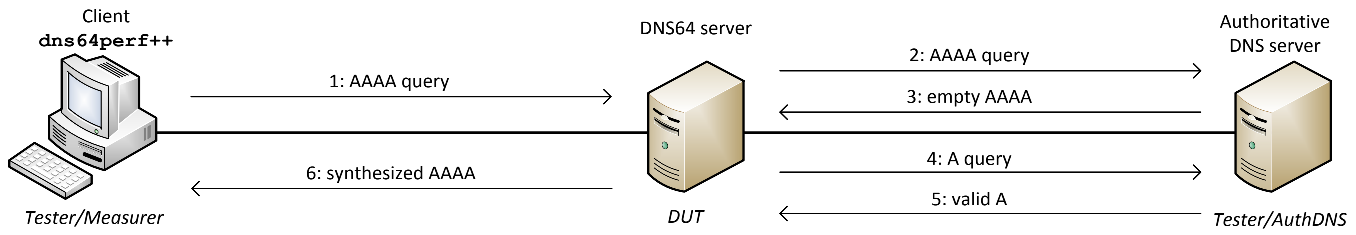

Fig. 1. Test and traffic setup for benchmarking DNS64 servers. [20]

II. Related Work

In our earlier paper on the performance analysis and comparison of different DNS64 implementations [10], we have pointed out that in all other papers than ours, the performance of a DNS64 server and that of a NAT64 gateway were measured together:

"The performance of the TAYGA NAT64 implementation (and implicitly of the TOTD DNS64 implementation) is compared to the performance of NAT44 in [11]. The performance of the Ecdysis NAT64 implementation (that has its own DNS64 implementation) is compared to the performance of the authors' own HTTP ALG in [12]. The performance of the Ecdysis NAT64 implementation (and implicitly the performance of its DNS64 implementation) is compared to the performance of both the NAT-PT and an HTTP ALG in [13]." [10]

We have also shown that the performances of the DNS64 server and the NAT64 gateway should be measured separately [10]. Although we have compared the performance of four DNS64 implementations BIND [14], TOTD [15], Unbound [16] and PowerDNS [17] in [10], the usability of our results is rather limited for two reasons:

1. The measurements were executed using only old and/or low performance devices up to 4 CPU cores as DUT (Device Under Test), in order to be able to overload them with our then available load generating solutions. However, modern server computers have more CPU cores.
2. The testing method was only suitable for performance comparison and stability analysis, but not for benchmarking as defined in RFC 8219. (Please see [9] for more details of the difference between the two.)

As for measurement tools, neither Nominum's **`dnsperf`** [18], nor our earlier **`dns64perf`** [19] complies with RFC 8219. As far as we know, the only RFC 8219 compliant DNS64 tester is Dániel Bakai's **`dns64per++`** [20], which implements the compulsory tests of RFC 8219. We have enabled it for measuring caching performance of DNS64 servers [21] and made a correction in its timing algorithm [22].

We have done some RFC 8219 compliant benchmarking in a few papers but their aim was always something different then the exhaustive benchmarking of DNS64 implementations. We aimed to demonstrate the operation and the expectable quality of the results of our benchmarking methodology [9]. We have demonstrated that mtd64-ng, our new tiny DNS64 proxy outperforms BIND, but the measurements were performed using a SBC (Single Board Computer) with four cores as DUT [23]. We demonstrated the operation of the DNS64 caching performance testing [21], and the improved quality of the results using the corrected timing algorithm [22].

Unfortunately, we cannot report any RFC 8219 compliant DNS64 benchmarking paper other than ours, which can be explained by the fact that RFC 8219 was published in August 2017.

Therefore, we contend that there is a need for RFC 8219 compliant DNS64 benchmarking tests, where the major DNS64 implementations are tested using contemporary servers with higher number of CPU cores.

III. Benchmarking Methodology for DNS64 Servers

A. *Summary of the Methodology*

In this section, we give a short summary of the benchmarking methodology for DNS64 servers mainly on the basis of RFC 8219, but we also adjust its presentation to our current needs: whereas RFC 8219 follows the traditional two devices setup of RFC 2544 [24], where the two devices are the *Tester* and the *DUT* (Device Under Test), now we use two separate devices for the two logical functions of the Tester.[1] (We found the usage of two physical devices necessary during the measurements for performance considerations.)

The test and traffic setup for benchmarking DNS64 servers is shown in Fig. 1. The message flow of six messages complies with the "worst case" scenario of a domain name resolution using DNS64, where the six messages are:

1. Query for the AAAA record of a domain name
2. Query for the AAAA record of the same domain name
3. Empty AAAA record answer
4. Query for the A record of the same domain name
5. Valid A record answer
6. Synthesized AAAA record answer.

To comply with RFC 8219, which requires the usages of all six messages, Tester/Measurer should send requests for *all different domain names* to prevent DUT from having a cache hit, as well as Tester/AuthDNS should be configured so that it reply with empty AAAA record and return exactly one A record for all the domain names used for testing.

The measurement procedure requires that Tester/Measurer

[1] We do so because we believe that it is easier to follow and understand its operation in this way, and also because we follow his approach in our measurement setup for performance reasons, and thus the two parts of the paper are synoptic.

sends DNS queries to the DUT at a constant rate for at least 60s, and checks the replies. The test is passed if and only if all the replies arrive within 1s timeout time from the sending of the corresponding queries and all the replies contain a AAAA record. Now we quote our arguments for this “strict” criterion of 0% loss, because it has serious consequences, which are demonstrated in Section V.B of our current paper. Our arguments were:

“We note that using a higher rate than the one measured with the above procedure might result in more successfully processed DNS record requests per second (and also non zero unanswered queries). However, we prudentially defined the performance metric with the above procedure for at least three reasons:

1. Our goal is a well-defined performance metric, which can be measured simply and efficiently. Allowing any packet loss would result in a need for scanning/trying a large range of rates to discover the highest rate of successfully processed DNS queries.
2. Even if users may tolerate a low loss rate (please note the DNS uses UDP with no guarantee for delivery), it cannot be arbitrarily high, thus, we could not avoid defining a limit. However, any other limits than zero percent would be hardly defendable.
3. Other benchmarking procedures use the same criteria of zero packet loss (possibly for the above two considerations).” [9]

We still consider these arguments valid from theoretical point of view, but on the basis of our measurement results of three important DNS64 implementations, we contend that, from practical point of view, there are some complementary performance metrics, which may be useful, when certain issues of DNS64 implementations are investigated and also in practice for a given class of users or under special circumstances. (Please refer to Section V.B for more details.)

In practice, binary search is used to find the highest rate at which the DUT can pass the test.

As there can be random events during the tests, which may influence the measurement, the binary search should be executed at least 20 times. Median should be used as summarizing function, plus 1 and 99 percentiles as indices of dispersion, which correspond to the minimum and maximum values if we have no more than 100 results. (Please refer to [9] for our considerations behind the choice of the summarizing function.)

Besides the compulsory “worst case” test, there are also optional tests.

If a domain name has a AAAA record then it is returned in message 3, messages 4 and 5 are omitted, no *IPv4-embedded IPv6 address* [25] synthesis is needed, but the AAAA record from message 3 is returned in message 6. Optional tests may be performed using domain names, 20%, 40%, 60%, 80%, and 100% of which have a AAAA record. We consider that the results of this test will become more and more realistic in the upcoming phases of IPv6 transition.

When there is a cache hit, message 1 is followed by message 6, which contains the AAAA record from the cache of the DNS64 server. Optional tests may be executed, when domain names are 20%, 40%, 60%, 80%, and 100% cached. We consider that the performance of a DNS64 server at given cache hit rates may be a useful factor in the evaluation of its performance, but we do not claim the direct usability of these numbers, as it is not a simple task to predict the actual cache hit rate of a DNS or DNS64 server.

RFC 8219 requires that a *self-test of the Tester* is performed to ensure that not the Tester but the DUT is the bottleneck during the measurements. For performing a self-test, the Tester is looped back, that is its Measurer subsystem is directly connected to its AuthDNS subsystem leaving out the DUT. A Tester (including its Measurer and AuthDNS subsystems) can be used for testing up to rate r with timeout t, if it passes a self-test at rate s with timeout $0.25t$, where s is defined by (1) and δ is at least 0.1.

$$s = 2r(1+\delta) \tag{1}$$

Its rationale is that the resolution of an AAAA record request by the DNS64 server may require the resolution of two requests (one for a AAAA record and one for an A record) by the authoritative DNS server, thus if $0.25t$ is used for each resolution, then $0.5t$ remains for all the other work, and δ ensures a performance reserve.

From practical viewpoint, if the performance of a Tester was measured and it was found that the highest rate it could pass the self-test was s_1, then the Tester can be used up to r_1 rate for DNS64 testing:

$$r_1 = \frac{s_1}{2(1+\delta)} \tag{2}$$

We note that if higher than r_1 measurement results are produced, they still prove that the DUT passed that test, but leave the question open what the maximum performance of the DUT is.

B. Clarification of the Self-Test Rules for Optional Tests

We have formulated the above described self-test rule focusing on the “worst case” test. Of course, it would be too strict to apply it literally for the optional tests. Therefore, now we clarify the self-test requirement for the optional tests.

First, we consider the case, when there are some AAAA records exist. Let α denote their proportion, where $0 < \alpha \leq 1$. If a AAAA record exists for a domain name then the authoritative DNS server needs to answer only one query, otherwise two. Thus, in this case s_{4A} rate defined by (3) is enough instead of s defined by (1).

$$s_{4A} = \left(1\alpha r + 2(1-\alpha)r\right)(1+\delta) = r(2-\alpha)(1+\delta) \tag{3}$$

Now, let us see that, depending on the proportion of the domain names with AAAA record, up to what r_1 rates we can use a Tester that passed a self-test at a given s_1 rate. Substituting s_1 for s_{4A} in (3), we obtain for r_1 as shown in (4).

$$r_1 = \frac{s_1}{(2-\alpha)(1+\delta)} \tag{4}$$

Let us see a numeric example. If we would like to test up to r=50,000qps, and δ=0.1 then the required self-test rate for the “worst-case” tests is 110,000qps. In the case when a AAAA record exist for 100% of the domain names, the Tester that passed the 110,000qps test, can be used up to 100,000qps instead of 50,000qps.

Now, we examine the case when some of the domain names are cached. Let β denote the cache hit rate, where $0 < \beta \le 1$. If a domain name is cached then its resolution by the DNS64 server does not require any effort from the authoritative DNS server. Thus, *considering only the AuthDNS subsystem*, the necessary s_C rate can be calculated according to (5).

$$s_{C,AuthDNS} = 2(1-\beta)r(1+\delta) \tag{5}$$

However, *Measurer subsystem* has to be able to send requests and receive replies at 10% higher rates then it is actually used.

$$s_{C,Measurer} = r(1+\delta) \tag{6}$$

The Tester must comply with both (5) and (6), that is, with (7).

$$s_C = Max(1, \quad 2(1-\beta))r(1+\delta) \tag{7}$$

Equation (7) may be rewritten as (8).

$$s_C = \begin{cases} 2(1-\beta)r(1+\delta) & if \quad \beta < 0.5 \\ r(1+\delta) & if \quad \beta \ge 0.5 \end{cases} \tag{8}$$

Now, let us see that, depending on the cache hit rate, up to what r_1 rates we can use a Tester that passed a self-test at a given s_1 rate.

$$r_1 = \begin{cases} \dfrac{s_1}{2(1-\beta)(1+\delta)} & if \quad \beta < 0.5 \\ \dfrac{s_1}{(1+\delta)} & if \quad \beta \ge 0.5 \end{cases} \tag{9}$$

Finally, the existence of AAAA records and non-zero cache hit rate can be examined together. In this case, the non-cached domain names require one query if they have a AAAA record, or two queries if they do not have a AAAA record. Thus, the required $s_{4A\&C}$ rate can be calculated according to (10).

$$s_{4A\&C} = Max(1, \quad (2-\alpha)(1-\beta))r(1+\delta) \tag{10}$$

And let us also see that, depending on both the AAAA record rate and the cache hit rate, up to what r_1 rates we can use a Tester that passed a self-test at a given s_1 rate.

$$r_1 = \begin{cases} \dfrac{s_1}{(2-\alpha)(1-\beta)(1+\delta)} & if \quad (2-\alpha)(1-\beta) > 1 \\ \dfrac{s_1}{(1+\delta)} & if \quad (2-\alpha)(1-\beta) \le 1 \end{cases} \tag{11}$$

C. *Very Short Summary of Dns64perf++*

The **`dns64perf++`** program is a command-line tool for benchmarking DNS64 servers [20]. It sends queries for AAAA records at a specified rate and receives the replies until the end of the specified sending time interval plus timeout time. It prints out the number of sent queries, received replies, and valid replies, which is to be understood as a reply arrived within timeout time from its corresponding query and also containing a AAAA record. As it is a program running under Linux using TCP/IP socket interface API for packet sending and receiving, its accuracy is limited. For more details, please refer to our (open access) paper [20].

We have extended **`dns64perf++`** with the feature of testing different cache hit rates [21]. Please refer to [9], how the cache hit rates recommended in RFC 8219 can be easily ensured without the knowledge of the cache size and/or cache control algorithm of the DNS64 implementations.

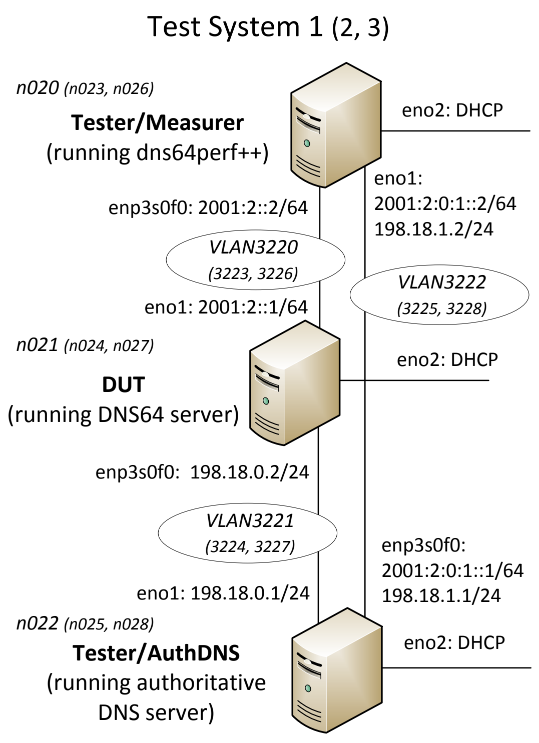

Fig. 2. Measurement setup for all types of measurements.

When using **`dns64perf++`** at higher than 50,000 queries per second rates, we have found an inaccuracy in its self-correcting timing algorithm and corrected it in [22], where we have also disclosed its estimated accuracy. We used **`dns64perf++`** with the corrected timing algorithm in all the measurements for this paper.

IV. MEASUREMENTS

A. *Measurement Environment*

The measurements were carried out in the Hokuriku StarBED Technology Center of NICT, Japan. The N nodes were used, which are Dell PowerEdge C6620 servers. We needed three computers for DNS64 measurements but reserved more computers to be able to speed up experimentation. As we have pointed out performance differences during the self-test phase, we performed the same kinds of tests of each examined DNS64 implementation using the very same computers and used the different three tuples of computers for different kinds of tests. Fig. 2 shows the topology of our measurement setup. It is to be understood in the way that *Test System 1* contained nodes n020, n021, and n022, which were interconnected by VLANs 3220, 3221, and 3222. *Test System 2* and *Test System 3* were built up in the same way containing the elements mentioned in parentheses in the first and second positions, respectively. The roles of the computers were the same as we have introduced in Fig. 1, namely: Tester/Measurer, DUT, and Tester/AuthDNS. The two subsystems of the Tester were also interconnected to facilitate self-test. As each connection used its own VLAN, we could use the same IP addresses in all three Test Systems. As for the IP address family, IPv6 had to be used between Tester/Measurer and DUT, as DNS64 offers a service to IPv6-only clients. Any IP version may be used between DUT and Tester/AuthDNS. Following the practice of [9] and [10], we used IPv4 for the measurements, but performed the self-test of the Tester using both IP versions, and the higher performance with IPv4 justified our practice. The IP addresses of the interfaces were set up according to Fig. 2. As all the links interconnected one 10Gbps interface and one 1Gbps interface, thus the speed of the connections was always 1Gbps, which was enough for our tests.

For the repeatability of our measurements, we disclose the hardware and software of the measurement environment in the appendix.

B. *DNS64 Implementation Selection*

As for DNS64 implementations to be tested, we have considered only free software [26] (also called open source [27]) for the same reasons given in [10].

We have tested the following DNS64 implementations:

- BIND 9.10.3-P4-Debian [14]
- PowerDNS Recursor 4.0.4 [17]
- Unbound 1.6.0 [16]
- mtd64-ng 1.1.0 [23] (for the calibration of the test system only)

The selection of the first three ones was not a question, as they are commonly used DNS servers, which support DNS64. We have also considered TOTD 1.5.3 [28], which we included in both [9] and [10], however, it is single-threaded, no more developed, and it always crashed after a few 60s long tests during our preliminary measurements, therefore we omitted it.

For testing the accuracy of the test system only, we have included mtd64-ng [23], which is an experimental DNS64 implementation and not yet ready for deployment. It produced very good quality results in our earlier tests in the sense that the difference between the minimum and maximum value of the results of the required 20 tests was very small compared to the median [9], therefore, now we also used it as a kind of result quality reference for the other implementations.

Although we were looking for, we did not find any other DNS64 implementations worth testing. We have also checked the Ecdysis [29] NAT64 implementation, which has its own DNS64 implementation, but it is either a patch for other DNS servers (BIND and Unbound), or its standalone version is a **`perl`** script [30], the performance of which was not worth measuring.

We did not intend to optimize the tested DNS64 implementations, because it is a nearly endless game, and several settings can be debated. Rather, we have used them as they are distributed in Debian (the first three) or downloadable from GitHub (mtd64-ng). The only setting beyond the configuration of DNS64 was, that we set the number of threads for the implementations that needed it.

As the configuration settings may influence the performance of the DNS64 servers, we disclose them for the repeatability of the measurements in the appendix.

C. *Main Types of Tests*

The four main types of tests follow the requirements of RFC 8219. Besides them, we performed some complimentary tests, which we found useful during the evaluation of the results: they will be described there.

1) *Self-test of the Tester*

As described in Section III.A, the aim of these tests was to determine, up to what rates the Tester can be used. As YADIFA [31] outperformed BIND [14], we do not include the details of the latter.

For self-tests, YADIFA 2.2.3 was used to serve AAAA records for all the domain names. We tested whether the version of the IP protocol used for carrying the DNS messages has an influence to the performance. (Although **`dns64perf++`** inputs the IPv6 address of the server to be tested, it did not prevent us from testing over IPv4: we used the *IPv4 mapped IPv6 address* of the authoritative DNS server, that is, ::ffff:198.18.1.1, which caused the Linux kernel to send out IPv4 packets to the 198.18.1.1 destination address.)

We have performed the self-test measurements for all three Test Systems, and found non-negligible differences among their DNS resolution performances.

We have completed this "official" self-test with a practical test, where mtd64-ng was benchmarked in all there test systems. Our preliminary tests showed that mtd64-ng achieved its best performance when 8 CPU cores were used, and its performance degraded on 16 CPU cores. (We did not do a detailed analysis of the performance results of mtd64-ng, because it is not ready for deployment yet, and it has several problems, to mention only one, which prevents mtd64-ng form scaling up better: it uses only a single listener, which becomes a bottleneck.)

Later, we added another self-test performed with NSD [32] as authoritative DNS server to serve as further result quality reference.

2) *Compulsory DNS64 Tests with Scale-up Test*

As the ongoing development in the hardware sector favors an increasing number of processing units over an increasing speed of a single unit [33], we consider it important to measure how the performances of the examined DNS64 implementations scale up with the number of CPU cores. Therefore, we have performed the "worst case" DNS64 test of the examined DNS64 implementations using the DUT with 1, 2, 4, 8, and 16 CPU cores online. (Technically, we performed the on/off switching of the *i*-th CPU core by writing 1/0 values into the `/sys/devices/system/cpu/cpu$i/online` file.)

3) *AAAA Record Rate Examinations*

We examined the effect of the AAAA record rate to the performance of PowerDNS, Unbound and BIND.

4) *Cache Hit Rate Examinations*

We examined the effect of the cache hit rate to the performance of PowerDNS, Unbound and BIND.

D. *Hypotheses*

On the basis of our previous experience with DNS64 testing, we set up a few hypotheses, which express our expectations.

Hypothesis 1. Mtd64-ng will show good quality, non-scattered results [9].

TABLE I
AUTHORITATIVE DNS PERFORMANCE OF YADIFA OVER IPV4

Test System	1	2	3
Median (queries per second)	180140	163641	180200
Minimum (queries per second)	163839	162303	176127
Maximum (queries per second)	182915	164641	185345

TABLE II
AUTHORITATIVE DNS PERFORMANCE OF YADIFA OVER IPV6

Test System	1	2	3
Median (queries per second)	162129	160003	159546
Minimum (queries per second)	147391	155133	157055
Maximum (queries per second)	166017	164357	163857

TABLE III
DNS64 PERFORMANCE OF MTD64-NG USING 8 CPU CORES

Test System	1	2	3
Median (queries per second)	63303	63093	63003
Minimum (queries per second)	62271	62343	61951
Maximum (queries per second)	64121	64001	64531
Dispersion (%)	2.92	2.63	4.10

TABLE IV
AUTHORITATIVE DNS PERFORMANCE OF NSD, SINGLE CPU CORE, IPV4

Test System	1	2	3
Median (queries per second)	168623	163904	167137
Minimum (queries per second)	167935	163447	165887
Maximum (queries per second)	168961	164361	167489
Dispersion (%)	0.61	0.56	0.96

Hypothesis 2. The different implementations will scale up differently [10].

Hypothesis 3. PowerDNS will scale up better than BIND [10].

Hypothesis 4. Unbound will show high performance when using a single CPU core [10].

V. RESULTS AND EVALUATION

A. *Self-test*

The Self-test results of YADIFA for IPv4 and IPv6 transport protocols are shown in Table I and Table II, respectively. Our most important observation is that the results are rather scattered in the majority of the test cases (Test System 2 using IPv4 is an exception). At this point, we cannot tell the reason, why: as it can be caused by several elements of the system, including YADIFA, **`dns64perf++`**, Linux, or even the hardware (the computers or the network). We do not go into deeper details, but return to the issue of the quality of the results at mtd64-ng, and answer the question at NSD.

As we use the system for measurements, we contend that not the median, but the minimum values are to be considered, because the unsatisfactory performance of the Tester should not make a negative impact for our measurements. Thus using the authoritative DNS server with IPv4, its at least 162,000qps performance enables us to perform DNS64 tests up to 73,000qps rate in all three test systems.

We note that it would have been enough to execute **`dns64perf++`** with a constant rate of 162,000qps (20 times) by all three test systems to satisfy the self-test requirement of RFC 8219, however we wanted to explore and disclose the performance of the three test systems.

As we mentioned before, mtd64-ng was used to check and demonstrate the accuracy of the result produced by the three test systems. The results are shown in Table III. We have added a row, which reflects the quality of the results: it is characterized by the dispersion of the results which is the proportion of the range of the results and the median, expressed in percentage, as defined by (12).

$$dispersion = \frac{\max - \min}{median} \cdot 100\% \qquad (12)$$

Our most important observation is that the dispersion was about 3-4%, which we interpret that all three test systems are able to produce meaningful results and thus may be used (and *Hypothesis 1* is confirmed).

We also note that the difference between the minimum (63,003qps) and maximum (63,303qps) of the medians of the results produced by the three test systems is 300qps, that is, less than 0.5% of any of them.

Before presenting some further self-test results, we would like to mention that, in parallel with the DNS64 measurements, we have also benchmarked several authoritative DNS server implementations using another set of computers. (We could not wait for the results of those tests before starting the DNS64 tests, as our time was limited.) We have found that NSD showed high and very stable results when executed using only a single core. Therefore, we have performed an additional self-test on all three tests systems using NSD with only a single online CPU core as authoritative DNS server. The results produced by the three test systems are shown in Table IV. Our most important finding is that the dispersion is always under 1%, thus the scattered nature of the results in Table I and Table II are to be attributed to the YADIFA tests using all 16 CPU cores. Another observation is that the difference between the highest median (168,623qps) and the lowest median (163,904qps) is 5,019, which is about 3%, thus it justifies our cautiousness that the benchmarking tests for comparison of the performances of different DNS64 implementations were always executed by the same test system.

B. *Scale-up Tests*

1) *General Overview and Problem Identification*

The DNS64 performance of PowerDNS, Unbound, and BIND using different number of CPU cores (and the same number of threads) are shown in Tables V, VI, and VII respectively. Our most important observations are:

- PowerDNS showed relatively low performance at a single CPU core (3,077qps), and it scaled up well up to 16 cores (26,620qps).
- Unbound performed excellently at a single CPU core (8,708qps), thus hypothesis 4 is confirmed, and it scaled up well up to 8 cores (31,502qps).
- The performance of Unbound degraded by more than 45% at 16 cores (17,131qps) compared to its performance 8 cores (31,502qps), which is a severe problem.
- The high dispersion of the results of Unbound from 2 to 16 CPU cores is another serious issue.
- BIND showed low performance at a single CPU

TABLE V
DNS64 PERFORMANCE AS A FUNCTION OF THE NUMBER OF CPU CORES AND THREADS, POWERDNS

Num. CPU cores	1	2	4	8	16
Median (qps)	3077	5498	10030	17290	26620
Minimum (qps)	3061	5439	9855	16603	24447
Maximum (qps)	3105	5633	10153	18563	27665
Dispersion (%)	1.43	3.53	2.97	11.34	12.09

TABLE VI
DNS64 PERFORMANCE AS A FUNCTION OF THE NUMBER OF CPU CORES AND THREADS, UNBOUND (SCATTERED RESULTS)

Num. CPU cores	1	2	4	8	16
Median (qps)	8708	15623	19993	31502	17131
Minimum (qps)	8511	11121	16223	24575	11955
Maximum (qps)	8865	16897	25665	32753	22017
Dispersion (%)	4.07	36.97	47.23	25.96	58.74

TABLE VII
DNS64 PERFORMANCE AS A FUNCTION OF THE NUMBER OF CPU CORES AND THREADS, BIND

Num. CPU cores	1	2	4	8	16
Median (qps)	2425	4788	6659	6659	6661
Minimum (qps)	2303	4731	6659	6659	6661
Maximum (qps)	2441	4897	6659	6659	6661
Dispersion (%)	5.69	3.47	0	0	0

core (2,425qps) and nearly doubled its performance at 2 CPU cores (4,788qps) but it showed a constant performance of 6,659qps from 4 to 8 cores and a very close fixed value of 6,661qps at 16 cores, which suggests a fundamental problem.

We note that hypotheses 2 and 3 are also confirmed, but the constant performance of BIND was surprising for us, which we had to investigate, as well as the two other issues.

2) Identifying the Causes

Looking for the reasons of these three issues, we have noticed in the measurement log files that all three problems implied that the rate of the valid answers was very often higher that 99% but somewhat less than 100%. Therefore, we have re-executed all the experiments with an acceptance criterion of 99% valid answers, which does not comply with RFC 8219, but can help revealing the reason of the three issues. The results are shown in Tables VIII, IX, and X. As for Unbound, its results are now not at all scattered and the dispersion is always under 10%. From 4 to 16 cores, its performance values are now significantly higher, but its performance still shows degradation form 8 to 16 cores. As for BIND, its performance now scales up well from 1 to 8 cores (from 2,448qps to 17,508qps) but shows somewhat degradation at 16 cores (15,748qps).

Another important observation is, that the single core (and therefore single thread) performances of all three DNS64 implementations showed only a negligible increase due to the change of the acceptance criterion. Therefore, we contend that the root cause of all three issues is the imperfect cooperation among the threads: whereas the vast majority of the requests are replied in time, a small fraction of them are not. This phenomenon can be observed with all three DNS64 implementations at a certain extent, but their measure is rather different. As for PowerDNS, there is practically no difference

TABLE VIII
DNS64 PERFORMANCE AS A FUNCTION OF THE NUMBER OF CPU CORES AND THREADS, POWERDNS
ACCEPTANCE CRITERION: 99%, NOT RFC 8219 COMPLIANT!

Num. CPU cores	1	2	4	8	16
Median (qps)	3092	5527	10146	19541	30954
Minimum (qps)	3071	5481	9949	17887	30393
Maximum (qps)	3121	5633	10321	20485	31793
Dispersion (%)	1.62	2.75	3.67	13.30	4.52

TABLE IX
DNS64 PERFORMANCE AS A FUNCTION OF THE NUMBER OF CPU CORES AND THREADS, UNBOUND
ACCEPTANCE CRITERION: 99%, NOT RFC 8219 COMPLIANT!

Num. CPU cores	1	2	4	8	16
Median (qps)	8710	16624	33852	66552	43790
Minimum (qps)	8447	16379	32767	65407	40959
Maximum (qps)	8793	16949	34305	67585	45089
Dispersion (%)	3.97	3.43	4.54	3.27	9.43

TABLE X
DNS64 PERFORMANCE AS A FUNCTION OF THE NUMBER OF CPU CORES AND THREADS, BIND,
ACCEPTANCE CRITERION: 99%, NOT RFC 8219 COMPLIANT!

Num. CPU cores	1	2	4	8	16
Median (qps)	2448	4852	9467	17508	15748
Minimum (qps)	2367	4727	9403	17135	15577
Maximum (qps)	2477	4993	9601	17731	16385
Dispersion (%)	4.49	5.48	2.09	3.40	5.13

TABLE XI
DNS64 PERFORMANCE USING 4 CORES FROM EACH CPU, POWERDNS

Num. CPU cores	4+4
Median (qps)	16771
Minimum (qps)	16095
Maximum (qps)	18529
Dispersion (%)	14.51

TABLE XII
DNS64 PERFORMANCE USING 4 CORES FROM EACH CPU, UNBOUND
ACCEPTANCE CRITERION: 99%, NOT RFC 8219 COMPLIANT!

Num. CPU cores	4+4
Median (qps)	59832
Minimum (qps)	58879
Maximum (qps)	60681
Dispersion (%)	3.01

between the results of the two measurement series from 1 to 4 threads, and the difference is relatively moderate even at 16 threads (26,620qps vs. 30,954qps). As for Unbound, the problem of high dispersion starts from 2 threads, although here the performance difference is not high (15,623qps vs. 16,624qps), but high performance differences can be observed from 4 to 16 cores. As for BIND, the problems is so serious from 4 cores that its performance is limited to a constant value in the RFC 8219 compliant tests.

As for the benchmarking methodology of DNS64 servers, we would like to emphasize that we do not mean to introduce any other percentage than 0% as the acceptance criterion for benchmarking tests, defined by RFC 8219. We have used 99% as a random value, which suited for our purposes to find the explanation of various issues.

3) Performance Model

To summarize our findings in a high level model of the multicore performance of DNS64 implementations, we can say that it depends on two factors: the single core performance and

TABLE XIII
DNS64 PERFORMANCE AS A FUNCTION OF AAAA RECORD RATE, POWERDNS, 4 CPU CORES, 4 THREADS

AAAA record rate	0%	20%	40%	60%	80%	100%
Median (qps)	10051	10893	11992	13249	14842	18107
Min. (qps)	9939	10751	11775	13051	14591	17775
Max. (qps)	10177	11265	12105	13449	15057	18561
Disp. (%)	2.37	4.72	2.75	3.00	3.14	4.34

TABLE XIV
DNS64 PERFORMANCE AS A FUNCTION OF AAAA RECORD RATE, UNBOUND, 1 CPU CORE, 1 THREAD

AAAA record rate	0%	20%	40%	60%	80%	100%
Median (qps)	8725	9495	10506	11837	13567	16459
Min. (qps)	8447	9151	10231	11263	13179	15735
Max. (qps)	8801	9729	10753	11993	13825	16897
Disp. (%)	4.06	6.09	4.97	6.17	4.76	7.06

TABLE XV
DNS64 PERFORMANCE AS A FUNCTION OF AAAA RECORD RATE, BIND, 2 CPU CORES, 2 THREADS

AAAA record rate	0%	20%	40%	60%	80%	100%
Median (qps)	4796	5203	5709	6277	6659	6659
Min. (qps)	4701	5119	5119	6143	6659	6659
Max. (qps)	4881	5281	5761	6345	6661	6721
Disp. (%)	3.75	3.11	11.25	3.22	0.03	0.93

the proper thread cooperation. Whereas for the optimization of the first one, program profiling is a matured method, which is also supported by modern tools such as CPU flame graphs [34], the optimization of the latter is a much harder task, although profiling might also help (e.g. checking the time while the threads are waiting on locks/mutexes), but even if one finds a bottleneck, its mitigation may be a more difficult problem then the optimization of a sequential program code.

4) Not All CPU Cores are Equal

Until now, we have mentioned only the number of the online CPU cores. However, cores 0-7 and cores 8-15 reside in two different CPUs. Using cores from two different CPUs for the execution of the threads of the same task may result in:

- *performance degradation* due to communication bottleneck between threads being executed by cores in different CPUs, or due to NUMA (Non-Uniform Memory Access) issues [36],
- *performance gain* due to better cache coverage.

To investigate the case, we have executed the 8-core tests, which used earlier cores 0-7, also using cores 0-3 and cores 8-11.

The results of PowerDNS are shown in Table XI. Comparing them with the 8-core results in Table V, it is visible that the median value decreased only in a small extent from 17,290qps to 16,771qps, where the difference is only about 3%. The increase of the dispersion is also relatively small (from 11.34% to 14.51%).

As for Unbound, we used the 99% acceptance criterion to get non-scattered results (for comparability). Therefore the results of Unbound in Table XII are to be compared with 8-core results in Table IX. Here the median decreased from 66,552qps to 59,832qps, which is about 10%. (We dedicate a separate case study to the investigation, whether this decrease is caused by NUMA issues, see Section VII.) Taking 59,832qps as a base, the 43,790qps performance at 16 cores is still a significant decrease (26,8%).

As for BIND, we did not see any point in testing it using 4+4 cores since its performance reached its fixed maximum value at 4 cores (and we did not want to increase the number of non-RFC 8219 compliant measurements beyond necessity).

5) Implementation Selection Guide

As for practical support for network operators, Tables V, VI, and VII can help in the selection of the appropriate DNS64 implementations depending on performance requirements.

As Tables VIII, IX, and X contain non RFC 8219 compliant results, they are not recommended for general use. However, they can be useful under special circumstances. E.g. in the case of a metropolitan free WiFi operator, where the packet loss rate is over 1% in the access network, they may be used. Or when a DoS attack of 50,000 qps rate against the DNS system is anticipated, Unbound at 8 cores might be an acceptable solution according to Table IX. For other special cases, some other acceptance criteria, e.g. 0.1% or 0.01% loss, may be suitable. Although we believe that these kind of tests may be useful under special circumstances, but we insist on that the 0% loss criterion should be used for benchmarking DNS64 servers in general.

C. Results of the AAAA Record Rate Examinations

Although it was natural for us to use all 16 cores of the DUT for these tests, but as we have received scattered results both with PowerDNS and Unbound, thus we decided not to include them. Whereas the 99% acceptance criterion made the results non-scattered, we did not want to include more non-RFC 8219 compliant results. Therefore, we decided to reduce the number of CPU cores to a value, which results in meaningful results. We used different number of CPU cores for each implementations, thus giving up performance comparison of the different DNS64 implementations in these tests: they were intended for the individual examinations of the given DNS64 server implementations.

As the tests with various AAAA record rates were executed by Test System 2, we have also performed 0% AAAA record rate tests to serve as a base for comparison, because they could be different from the results of the scale-up tests, which were executed by Test System 1.

1) PowerDNS

For PowerDNS, we used 4 CPU cores at the DUT. The DNS64 performance results of PowerDNS as a function of AAAA record rate is shown in Table XIII. The median value of its 0% AAAA record performance is 10,051qps, which is very close to 10,030qps, the 4-core performance of PowerDNS in the scale up tests.

The dispersion of the results remained under 5% for all AAAA record rates. And the performance of PowerDNS showed an increase, which complies with our simple theoretical model and experimental results in [9].

2) Unbound

For Unbound, we used only a single CPU core. The DNS64

TABLE XVI
DNS64 PERFORMANCE AS A FUNCTION OF CACHE HIT RATE, POWERDNS, 4 CPU CORES, 4 THREADS

cache hit rate	0%	20%	40%	60%	80%	100%
Median (qps)	10055	12147	14731	19179	29242	100816
Min. (qps)	9975	11759	14463	18927	28671	80831
Max. (qps)	10197	12423	15873	19361	29715	102993
Disp. (%)	2.21	5.47	9.57	2.26	3.57	21.98

TABLE XVII
DNS64 PERFORMANCE AS A FUNCTION OF CACHE HIT RATE, UNBOUND, 1 CPU CORE, 1 THREAD

cache hit rate	0%	20%	40%	60%	80%	100%
Median (qps)	8680	10261	12975	17760	29677	102395
Min. (qps)	8415	10077	12543	17335	28953	65535
Max. (qps)	8961	10513	13145	18241	30273	115745
Disp. (%)	6.29	4.25	4.64	5.10	4.45	49.04

TABLE XVIII
DNS64 PERFORMANCE AS A FUNCTION OF CACHE HIT RATE, BIND, 2 CPU CORES, 2 THREADS

cache hit rate	0%	20%	40%	60%	80%	100%
Median (qps)	4791	5817	7383	8996	15898	25819
Min. (qps)	4695	5743	7255	8571	12287	24575
Max. (qps)	4873	5953	7465	9473	16193	26641
Disp. (%)	3.72	3.61	2.84	10.03	24.57	8.00

performance results of Unbound as a function of AAAA record rate is shown in Table XIV.

The dispersion of the results varies around 4-7%, which is still acceptable, and the performance of Unbound increases as expected.

3) BIND

In parallel with the scale-up tests, we have tested BIND using 16 CPU cores. Regardless of the AAAA record percent, the results were always 6,661qps. This result reveals that the fixed performance issue is likely not caused by a bottleneck in the communication of BIND with the authoritative DNS server or a bottleneck in the IPv4-embedded IPv6 address synthesis code of BIND as the results did not improve with the more frequent occurrences of the AAAA records. To present some more meaningful results, we selected two cores, which still enables us to reveal what happens when the strange performance limit of BIND is reached. The DNS64 performance results of BIND as a function of AAAA record rate is shown in Table XV. The strange performance limit is reached at 80% AAAA record rate. There happened a single occurrence of the 6721qps result at 100% AAAA record rate, which we attribute to a very rare random event, a kind of coincidence, which was favorable to the performance of BIND. Please recall that this has never happened among the 6*20=120 measurements performed at 16 cores for different AAAA record rates.

D. Results of the Cache Hit Rate Examinations

We used the same number of CPU cores for the cache hit rate examinations as with the AAAA record rate examinations.

As the tests with various cache hit record rates were executed by Test System 3, we have also performed 0% cache hit rate tests to serve as a base for comparison.

1) PowerDNS

The DNS64 performance results of PowerDNS as a function of cache hit rate is shown in Table XVI. The median value of its 0% cache hit rate performance is 10,055qps, which is very close to 10,030qps, the 4-core performance of PowerDNS in the scale up tests. The dispersion of the results remained under 10% for all cache hit rates from 0% to 80%, but it was high at 100% cache hit rate (21.98%). This high dispersion was caused by the smallest result of 80,831qps, which occurred only ones, and the next smallest result was 85,453qps, which was also far from the median.

The performance of PowerDNS showed an increase, which complies with our simple theoretical model and experimental results in [9]. The large difference between the 80% and 100% cache hit rate results might suggest that other rates (e.g. 90%, 95%, or 99%), would have been worth testing, but although they might be interesting from theoretic point of view, we do not think that higher than 80% cache hit rate would be relevant in practice.

We note that the 100,816qps DNS64 resolution rate achieved at 100% cache hit rate is higher than 73,000qps, thus it should have been invalidated without our clarification to the requirements for the tester in Section III.B.

2) Unbound

The DNS64 performance results of PowerDNS as a function of cache hit rate is shown in Table XVII. The dispersion of the results is very high at 100% cache hit rate (49.04%) but it remained under 7% for all other tested cache hit rates. The very high dispersion is caused by the 65,535 values, which occurred 9 times among the 20 results. As the [0, 131,072] initial range was used for the binary search, this value was produced in a way that the first test of the binary search at 65,536qps rate failed, and then all other tests were successful. We contend that in case of a single failure it could be debated whether it was caused by a random event inside or outside Unbound, but the 9 failures have to be attributed to Unbound and not a random event in the test system outside Unbound, as they did not occur during the testing of PowerDNS although the measurement results were in the same order of magnitude.

3) BIND

The DNS64 performance results of BIND as a function of cache hit rate is shown in Table XVIII. In this case the DNS64 performance of BIND is not limited by a constant value, and it complies with our simple theoretical model and experimental results in [9].

E. Where is the DNS64 Performance Bottleneck of BIND?

First, let us collect our observations. The DNS64 performance of BIND increases with the number of CPU cores up to 4 cores, where it reaches a fixed upper bound. Although this upper bound seems to be sharp, much higher rates can be achieved if some loss is allowed. Thus it is probably not a classic performance bottleneck but perhaps some inter-thread cooperation issue. The 100% AAAA record rate, which halves the number of messages between the DNS64 server and the authoritative DNS server and eliminates the need for IPv4-embedded IPv6 addresses synthesis, does not help at all. But the strange limit can be crossed, when there is a cache hit.

Before presenting our hypothesis about a possible cause of the DNS64 performance bottleneck of BIND, we need to give a short introduction to the *cache poisoning* [35] threat and its countermeasures.

1) Very Basic Introduction to Cache Poisoning and its Countermeasures

The basic idea of cache poisoning is that the attacker sends a request for a given domain name to the attacked recursive DNS server, and if the given domain name is not yet cached, the recursive DNS server sends out a DNS query for the same domain name to an authoritative DNS server, whereas the attacker sends a forged answer to this query to the recursive DNS server, spoofing the authoritative DNS server, before the reply of the authoritative DNS server arrives. The recursive DNS server accepts an answer only in the case if it comes from the right IP address and port number to the right IP address and port number, and the Transaction ID as well as the query matches. The attacker performs blind spoofing, that is the attacker has to guess the source port number and transaction ID of the query sent by the recursive DNS server to the authoritative DNS server. To be successful, the attacker has to send a high number of forged replies with different port number and Transaction ID combinations in a short time window of about 100ms. The protection against cache poisoning must include Transaction ID and source port randomization using cryptographically secure random numbers as well as refraining from sending equivalent queries (with identical QNAME, QTYPE, and QCLASS fields) concurrently, which could lead to a birthday paradox based attack, thus efficiently reducing the number of messages necessary for a successful attack [37].

2) Our Hypothesis for a Possible Cause of the Low DNS64 Performance of BIND

In our understanding, recursive DNS servers have to account the currently outstanding queries to avoid sending out multiple equivalent queries concurrently. Regardless of its implementation, we call it *outstanding query database*. We contend that the improper organization of the locking (for write access) of this database by multiple threads might be a possible cause of the low DNS64 performance of BIND. When there is a cache hit, there is no need to access the database thus no problem occurs. Regardless of the probability of the existence of the AAAA records, the queries have to be inserted to and deleted from the database. And due to improper organization, the access to the database is denied at a high enough rate in a small proportion of the cases, which results in the failure of the RFC 8219 tests, but enables much higher rate for a more permissive test, e.g. the one which tolerates 1% loss.

We cannot prove this hypothesis, we only contend that it can be an explanation to the symptoms we experienced. We have reported the issue to the developers of BIND.

VI. Computing Power Efficiency of DNS64 Servers

A. Definition

The energy efficiency of the Internet infrastructure is an important concern for several years [38]. We intend to contribute to this field by defining and measuring the computing power efficiency of DNS64 servers. We are aware that there is no linear relationship between the computing power and the energy consumption, but we intend to define a metric, which is easy to measure and is definitely relevant to the operators of DNS64 servers. We are also aware that these results are dependent from the computer used for benchmarking, and therefore we propose it only as a complementary metric, but we contend that it can be a useful additional factor to the network operators for decision making. We define the *computing power relative DNS64 performance* as the number of processed queries per second divided by the CPU utilization of the computer. For clarity, we count the full utilization of a single CPU as 1. (Linux reports the CPU utilization in percentage, e.g. the maximum CPU utilization of a 16-core system is reported as 1600%, and we count it as 16.)

B. Development of the Measurement Method

During benchmarking DNS64 servers, DUT is handled as a "black box". The CPU utilization measurements could influence the performance of the DNS64 implementations, thus it has to be done separately. Designing these tests, we have identified two challenges:

1. In the beginning of a test, there may be an initial transient of unknown length, during which the CPU utilization of the DNS64 implementations differ from its steady state value.
2. The ratio of the query rate and the CPU utilization is not necessarily constant, but may depend on the load conditions.

Therefore, we had to check these phenomena.

1) Examining the Length of the Initial Transient

We have tested how the CPU utilization changes over time during the benchmarking of the three tested implementations. Our goal was to determine the length of the initial transient, which have to be left out from the evaluation, when determining the computing power consumption of the different DNS64 implementations.

The measurements were performed at different rates, using 180s long tests. The CPU utilization of the DNS64 implementations was measured with the top command using it in batch mode and printing out the CPU utilization values in 1s resolution. The relevant part of the measurement script was:

```
top –b –n 180 –d 1 –p $(pidof $DNS64server)
```

We have examined several DNS64 implementation, CPU core number and query rate combinations, and selected some meaningful ones for Fig. 3. Whereas the initial transient is very short (2-3s) for Unbound, it is about 10-15s for BIND and about 40-50s for PowerDNS. We have chosen 60s as a common upper bound for the length of the initial transient for all three implementations. And we also decided to use 180s long tests, thus we have 120 results after the deletion of the first 60 ones. (We recommend this relatively high number of results because of the fluctuations observed in the CPU utilization of PowerDNS. Otherwise, a smaller value, e.g. 30 could do.)

The significant oscillation of the CPU utilization of PowerDNS at 25,000qps rate is another interesting observation.

2) Testing the Query Rate and CPU Utilization Ratio

To examine how the computing power relative performances

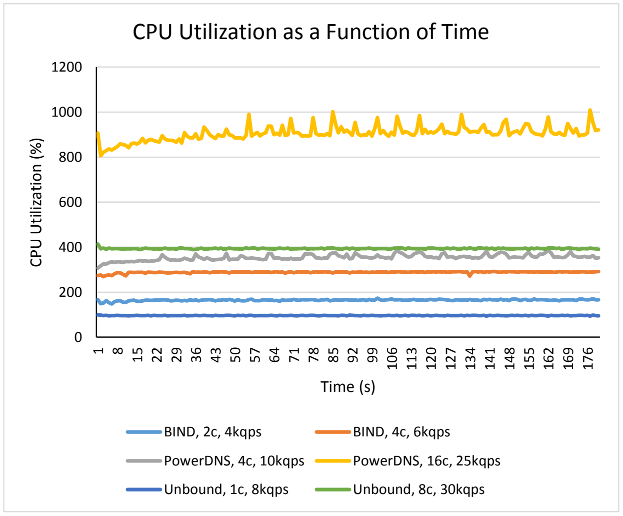

Fig. 3. Study of the initial transient of the CPU utilization.

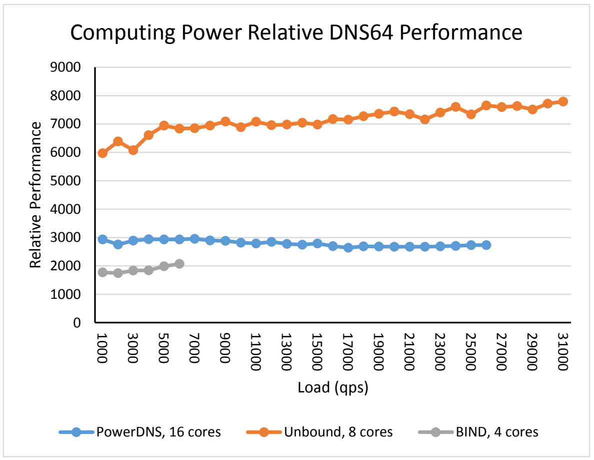

Fig. 4. Computing power relative performances of PowerDNS, Unbound and BIND under different load conditions.

of the three tested DNS64 implementations depend on the load conditions, we have measured their CPU utilization at several rates using as large intervals as possible, and calculated their computing power relative performances. Fig. 4 shows the results. Whereas the computing power relative performance of PowerDNS is close to constant, that of Unbound and BIND show somewhat increasing tendency with the load. Because of the slightly increasing tendency, we propose two methods. The fine grain one determines the computing power relative performance of the DNS64 servers at 3 working points, namely, when the query rate is the 25%, 50%, and 75% of the maximum DNS64 performance, whereas the coarse one does it only at a query rate of 50% of the maximum DNS64 performance.

C. *Measurement and Results*

Using our DNS64 benchmarking results, namely the median values for PoweDNS, Unbound, and BIND from Table V, Table VI, and Table VII, respectively, we have performed the measurements for their fine grain characterization. The computing power relative DNS64 performance results for PowerDNS, Unbound, and BIND are displayed in Table XIX, Table XX, and Table XXI, respectively. In the last lines of the tables, we have also included the CPU utilization values measured at full load, as they may give further insight into the behavior of the given DNS64 implementations. (We took no care if the tests failed or passed: they could fail due to the extra load generated by the CPU utilization measurement.)

As for the computing power relative DNS64 performance results of PowerDNS, there are some fluctuations, but there is no significant degradation at 16 CPU cores compared to 8 or any other number of CPU cores. In contrast, the computing power relative DNS64 performance results of both Unbound shows significant decrease, when using 16 CPU cores. Considering the 50% load results and comparing the computing power relative DNS64 performance results at 8 and 16 cores, they decrease from 7,141qps to 5,671qps. We seek the reasons in a separate case study in Section VII.

Comparing the computing power relative DNS64 performance results at different load levels, we cannot observe significant tendencies: they sometimes grow (e.g. Unbound, 4 cores), sometimes decrease (PowerDNS 16 cores) with the increase of the load. As the results at 25% or 75% load usually do not significantly differ from the results at 50% load, we consider that in the case of the three tested DNS64 implementations it is enough to disclose only the computing power relative DNS64 performance results measured at 50% of the maximum DNS64 performance. However, the situation may be different in the case of other DNS64 implementations, therefore we recommend to perform the measurement in all three working points. And as for the presentation of the results, it is always imperative to disclose the query rate and CPU core numbers, as well as CPU type used for testing to avoid the

TABLE XIX
COMPUTING POWER RELATIVE DNS64 PERFORMANCE (CPRDP) UNDER DIFFERENT LOAD CONDITIONS, POWERDNS

Num. CPU cores	1	2	4	8	16
Full rate (qps)	3077	5498	10030	17290	26620
CPRDP @25%	2960	2855	2938	2973	2932
CPRDP @50%	2947	2830	2806	2831	2808
CPRDP @75%	2932	2751	2780	2717	2655
CPU util. @100%	99.3%	196.6%	360.7%	617.1%	974.6%

TABLE XX
COMPUTING POWER RELATIVE DNS64 PERFORMANCE (CPRDP) UNDER DIFFERENT LOAD CONDITIONS, UNBOUND

Num. CPU cores	1	2	4	8	16
Full rate (qps)	8708	15623	19993	31502	17131
CPRDP @25%	7735	7524	7598	6855	5369
CPRDP @50%	7735	7273	7793	7141	5667
CPRDP @75%	7749	7786	7805	7517	5847
CPU util. @100%	98.8%	188.2%	247.5%	407.8%	291.3%

TABLE XXI
COMPUTING POWER RELATIVE DNS64 PERFORMANCE (CPRDP) UNDER DIFFERENT LOAD CONDITIONS, BIND

Num. CPU cores	1	2	4	8	16
Full rate (qps)	2425	4788	6659	6659	6661
CPRDP @25%	1861	1735	1785	1786	1538
CPRDP @50%	1980	1929	1826	1645	1550
CPRDP @75%	2062	2452	2005	1696	1333
CPU util. @100%	99.3%	196.6%	312.8%	387.6%	565.9%

TABLE XXII
NUMASTAT RESULTS OF POWERDNS USING 8 CORES AND 8 THREADS, TEST RATE: 1500QPS

		node0 hit	node1 hit	all hit	node0 miss	node1 miss	all miss
using cores 0-7	median	282114	0	282114	0	16	16
	minimum	271221	0	271221	0	2	2
	maximum	300544	0	300544	0	29	29
using cores 0-3 and 8-11	median	173514	115177	288363	2	7	9
	minimum	162120	101598	274902	0	0	0
	maximum	187409	128805	310166	9	22	24

TABLE XXIII
NUMASTAT RESULTS OF UNBOUND USING 8 CORES AND 8 THREADS, TEST RATE: 1500QPS

		node0 hit	node1 hit	all hit	node0 miss	node1 miss	all miss
using cores 0-7	median	111166	0	111166	0	16	16
	minimum	90326	0	90326	0	1	1
	maximum	115258	0	115258	0	30	30
using cores 0-3 and 8-11	median	51608	23349	74488	6	3	11
	minimum	42668	13819	62445	0	0	2
	maximum	60379	29118	86297	23	15	27

possibility of gaming.

VII. CASE STUDY: NUMA INSIGHTS

Modern multiprocessor systems use NUMA (Non-Uniform Memory Access) instead of SMP (Symmetric Multiprocessing) for scalability considerations.

We have examined, whether the performance degradation of Unbound at 8 cores, when the cores reside in two physical CPUs, might be caused by NUMA issues.

We have measured the NUMA hits and misses for both PowerDNS and Unbound using 8 cores first as cores 0-7, which all reside in the first CPU (node0 in NUMA terminology) and then as cores 0-3 (in node0) plus cores 8-11 (in node1).

For the measurements, we used the **`numastat`** Linux command without arguments, as in this way it displays the *number of memory requests*, whereas it gives MBs, if any arguments are specified. Thus we could not filter to the memory requests of the DNS64 implementations (because it would have needed an argument), but all the memory requests in the Linux system were included. Therefore, we performed the measurements 20 times and calculated the median values to filter out the effect of possible random events (outside the DNS64 implementations), and also included the minimum and maximum values to reflect the dispersion of the results. (As **`numastat`** displays the cumulative values, we have queried the counters before and after each tests and calculated the differences.)

The NUMA results of PowerDNS are shown in Table XXII. The miss values are negligible in both cases as expected. An interesting observation is that the memory requests are not distributed evenly between the two CPUs even though 8 threads were used. This uneven distribution of the work did not turn out from the CPU load measurements, as then the cumulative load of all CPU cores was measured.

The NUMA results of Unbound are shown in Table XXIII. As the number of NUMA miss count are negligible in both cases, we can conclude that the performance degradation of Unbound is not caused by NUMA issues. Our result is in a good agreement with the fact that our Linux kernel version (4.9.0-4-amd64) is higher than any of the ones mentioned in [36] as having scheduler issues and thus resulting in significant performance loss on NUMA architectures.

Another interesting observation is, that Unbound had significantly lower number of memory requests then PowerDNS.

VIII. DISCUSSION AND FUTURE WORK

On the one hand, both the highest single core performance (8,708qps) and the highest overall DNS64 performance (31,502qps) results were produced by Unbound (at 8 cores). It was also Unbound that showed the best computing power relative performance. Therefore, Unbound seems to be the best choice for a DNS64 implementation at the moment.

On the other hand, although PowerDNS showed only a moderate single core performance (3,077qps) and its highest performance is only (26,620qps), but is scaled up well and it outperformed Unbound at 16 cores, which showed a serious performance degradation at 16 cores (17,131qps). We surmise that an average DNS64 server administrator will not even think of switching off 8 cores of a 16-core server to increase its performance, thus we believe that the 17,131qps value is to be primarily considered as the DNS64 performance of Unbound at 16 cores. And the development of the CPUs has not stopped. For example, the Intel Xeon Phi 7200 processor [39] has 64 cores. Thus, we contend that scalability of DNS64 implementations has utmost importance.

We plan to examine the performance of PowerDNS and Unbound using a server with higher number of CPU cores as DUT. The expected new version of mtd64-ng is also a candidate. BIND definitely needs a bugfix to be included. Suggestions for testing further DNS64 implementations are also welcome.

IX. CONCLUSION

For the optional DNS64 tests of RFC 8219, we have clarified the requirements for the Tester, which are thus less demanding then the literal application of the requirements formulated for the compulsory tests.

By carefully examining the DNS64 performances of BIND, PowerDNS and Unbound using 1, 2, 4, 8, and 16 CPU cores, we came to the conclusion that on the one hand, currently Unbound provides the highest single core DNS64 performance (8,708qps), the highest overall DNS64 performance (31,502qps, at eight CPU cores) and the highest computing power relative DNS64 performance (about 5,300qps/core - 7,700qps/core depending on the load and the number of CPU cores used). But on the other hand, PowerDNS is the only one that scaled up well, which has utmost importance due to the current trends in CPU development. BIND showed the lowest single core DNS64 performance (2,425qps) an also the lowest overall DNS64 performance: it achieved about 6,660qps rate at four CPU cores and its performance did not increase using 8 or 16 cores, which is a fundamental problem.

By defining and measuring the computing power relative performance of DNS64 servers, we provided energy efficiency aware DNS64 server administrators with another important factor for DNS64 implementation selection.

ACKNOWLEDGMENTS

This work was supported by the International Exchange Program of the National Institute of Information and Communications Technology (NICT), Japan.

The measurements were carried out by using the resources of NICT StarBED, Japan.

The authors would like to thank Shuuhei Takimoto, Masatoshi Enomoto, Yutaka Nomura, and Marius Georgescu for their help and advice in NICT StarBED usage related issues.

X. APPENDIX

A. Parameters of the Measurement Environment

For the repeatability of our measurements, we specify the most important parameters of the computers. Each Dell PowerEdge C6620 computer contained two Intel Xeon E5-2650 2GHz CPUs, having 8 cores each, 16x8GB 1333MHz DDR3 RAM, two Intel I350 Gigabit Ethernet NICs, two Intel 10G 2P X520 Adapters, but only one of them contained a 10G interface module, 500GB HDD, and also SSDs, but they were not used.

On the basis of our previous experience, we disabled hyper-threading and set the CPU frequency of the computers to fixed 2GHz, in order to prevent scattered measurement results [9]. The applied BIOS settings were:

1. `Advanced / CPU Configuration / Hyper-Threading Technology`: **`Disabled`**
2. `Server / ACPI SPMI Table`: **`Disabled`**
3. `Advanced / Power Management / Power Management`: **`Maximum performance`**
4. `Advanced / CPU Configuration / Turbo Mode`: **`Disabled`**

However, Turbo Mode was enabled on nodes n020, n023, and n026 to ensure high enough performance for the Measurer subsystem.

Debian GNU/Linux 9.2 operating system with kernel version 4.9.0-4-amd64 was installed to all the computers.

As for authoritative DNS server, YADIFA 2.2.3-6237 was used, because this is the version included in the Debian 9.2 distribution.

B. Configuration Settings

1) BIND DNS64 Settings

The DNS64 function of BIND was enabled and set up in the `/etc/bind/named.conf.options` file as follows:

```
options {
  directory "/var/cache/bind";
  forwarders { 198.18.0.1; };
  dns64 2001:db8:abba::/96 { };
  dnssec-validation no;
  auth-nxdomain no;    # conform to RFC1035
  listen-on-v6 { any; };
};
```

We did not change other configuration options of BIND, e.g. the number of working threads, because BIND automatically starts the number of working threads equal to the number of CPU cores available. It also starts multiple listeners since version 9.9.0 [40].

2) PowerDNS Settings

We have made the following relevant modifications to the `/etc/powerdns/recursor.conf` file:

```
allow-from=::/0, 0.0.0.0/0
forward-zones=dns64perf.test=198.18.0.1
local-address=2001:2::1, ::1, 127.0.0.1
lua-dns-script=/etc/powerdns/dns64.lua
threads=(varied from 1 to 16)
```

The number of threads was set to be equal with the number of active CPU cores.

For enabling DNS64, we placed the following lines into the `/etc/powerdns/dns64.lua` file:

```
prefix = "2001:db8:edda::"
function nodata ( dq )
  if dq.qtype ~= pdns.AAAA then
    return false
  end -- only AAAA records

  dq.followupFunction = "getFakeAAAARecords"
  dq.followupPrefix = prefix
  dq.followupName = dq.qname
  return true
end
```

3) Unbound Settings

To the `/etc/unbound/unbound.conf` file, we have added the following lines:

```
access-control: ::/0 allow
module-config: "dns64 iterator"
dns64-prefix: 2001:db8:bd::/96
forward-zone:
  name: dns64perf.test.
  forward-addr: 198.18.0.1
server:
  interface: 2001:2::1
  interface: ::1
  interface: 127.0.0.1
  num-threads: (varied from 1 to 16)
```

The number of threads was set to be equal with the number of active CPU cores.

4) Mtd64-ng Settings

The `/etc/mtd64-ng.conf` file had the following relevant settings:

```
nameserver 198.18.0.1
dns64-prefix 2001:db8:d64::/96
debugging no
timeout-time  0.5
resend-attempts   0
response-maxlength  512
num-threads 30 # default setting
```

5) Authoritative DNS Server Configuration

As for authoritative DNS server, we used YADIFA 2.2.3-6237, because it outperformed BIND 9.10.3-P4-Debian, which we also tested in the role of the authoritative DNS server.

The relevant settings of the YADIFA authoritative DNS server in the `/etc/yadifa/yadifad.conf` file were:

```
        listen          0.0.0.0, ::
<zone>
        type            master
        domain          dns64perf.test
        file            db.dns64perf.test
</zone>
```

The content of the **`db.dns64perf.test zone`** file was generated by a script. The zone file contained 2^{24} number of different entries for the 010-{0..255}-{0..255}-{0..255}.dns64perf.test namespace, see [20] for the details. For self-tests, these domain names had AAAA records, because **`dns64perf++`** can send queries for AAAA records only. For general DNS64 tests and caching tests, they were mapped to A records only. For tests examining the effect of the existence of AAAA records, they occurred in 20%, 40%, 60%, 80%, or 100%. For the details of the solution, please refer to [9].

We have also used NSD 4.1.14 as authoritative DNS server in a special self-test.

References

[1] S. Deering and R. Hinden, "Internet Protocol, version 6 (IPv6) specification", IETF RFC 8200 (STD: 86), Jul. 2017. DOI: 10.17487/RFC8200

[2] M. Nikkhah and R. Guérin, "Migrating the internet to IPv6: An exploration of the when and why", *IEEE/ACM Transactions on Networking*, vol. 24, no 4, pp. 2291–2304, Aug. 2016, DOI: 10.1109/TNET.2015.2453338

[3] M. Bagnulo, A. Garcia-Martinez and I. Van Beijnum, "The NAT64/DNS64 tool suite for IPv6 transition", *IEEE Commun. Magazine*, vol. 50, no. 7, pp. 177–183, Jul. 2012. DOI: 10.1109/MCOM.2012.6231295

[4] N. Skoberne, O. Maennel, I. Phillips, R. Bush, J. Zorz, M. Ciglaric, "IPv4 Address sharing mechanism classification and tradeoff analysis", *IEEE/ACM Trans. Netw.*, vol. 22, no. 2, pp. 391–404, Apr. 2014, DOI: 10.1109/TNET.2013.2256147

[5] M. Bagnulo, A Sullivan, P. Matthews and I. Beijnum, "DNS64: DNS extensions for network address translation from IPv6 clients to IPv4 servers", IETF RFC 6147, 2011. DOI: 10.17487/RFC6147

[6] M. Bagnulo, P. Matthews and I. Beijnum, "Stateful NAT64: Network address and protocol translation from IPv6 clients to IPv4 servers", IETF RFC 6146, 2011. DOI: 10.17487/RFC6146

[7] G. Lencse, Y. Kadobayashi, "Methodology for the identification of potential security issues of different IPv6 transition technologies: Threat analysis of DNS64 and stateful NAT64", *Computers & Security*, vol. 77, no. 1, pp. 397-411, September 1, 2018, DOI: 10.1016/j.cose.2018.04.012

[8] M. Georgescu, L. Pislaru and G. Lencse, "Benchmarking methodology for IPv6 transition technologies", IETF RFC 8219, Aug. 2017. DOI: 10.17487/RFC8219

[9] G. Lencse, M. Georgescu, and Y. Kadobayashi, "Benchmarking methodology for DNS64 servers", *Computer Communications*, vol. 109, no. 1, pp. 162–175, 2017, DOI: 10.1016/j.comcom.2017.06.004

[10] G. Lencse and S. Répás, "Performance analysis and comparison of four DNS64 implementations under different free operating systems", *Telecommunication Systems*, vol. 63, no. 4, pp. 557–577, Nov. 2016, DOI: 10.1007/s11235-016-0142-x

[11] K. J. O. Llanto, and W. E. S. Yu, Performance of NAT64 versus NAT44 in the context of IPv6 migration, in *Proc. International MultiConference of Engineers 2012 (IMECS 2012)*, Hong Kong, March 14-16, 2012, vol. I., pp. 638–645.

[12] C. P. Monte, M. I. Robles, G. Mercado, C. Taffernaberry, M. Orbiscay, S. Tobar, R. Moralejo, S. Pérez, "Implementation and evaluation of protocols translating methods for IPv4 to IPv6 transition", *Journal of Computer Science & Technology*, vol. 12, no. 2, Aug. 2012, pp. 64–70.

[13] S. Yu, and B. E. Carpenter, "Measuring IPv4 – IPv6 translation techniques", Dept. of Computer Science, Univ. Auckland, Auckland, New Zeeland, *Technical Report 2012-001*, Jan. 2012, [Online]. Available: https://www.cs.auckland.ac.nz/~brian/IPv4-IPv6coexistenceTechnique-TR.pdf

[14] Internet Systems Consortium, "BIND: Versatile, classic, complete name server software", [Online]. Available: https://www.isc.org/downloads/bind

[15] The 6NET Consortium, "An IPv6 deployment guide", Ed. Martin Dunmore, Sept. 2005. [Online]. Available: http://www.6net.org/book/deployment-guide.pdf

[16] NLnet Labs, Unbound, [Online]. Available: http://unbound.net

[17] Powerdns.com BV, "PowerDNS", [Online]. Available: http://www.powerdns.com

[18] Nominum: "Dnsperf: DNS performance tool manual", [Online]. Available: https://github.com/Sinodun/dnsperf-tcp/blob/master/doc/dnsperf.pdf

[19] G. Lencse, "Test program for the performance analysis of DNS64 servers", *International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems*, vol. 4. no. 3. (2015.) pp 60-65. DOI: 10.11601/ijates.v4i3.121

[20] G. Lencse, D. Bakai, "Design and implementation of a test program for benchmarking DNS64 servers", *IEICE Transactions on Communications*, vol. E100-B, no. 6. pp. 948-954, Jun. 2017. DOI: 10.1587/transcom.2016EBN0007

[21] G. Lencse, "Enabling dns64perf++ for benchmarking the caching performance of DNS64 servers", unpublished, review version is available from: http://www.hit.bme.hu/~lencse/publications/

[22] G. Lencse and A. Pivoda, "Checking and increasing the accuracy of the dns64perf++ measurement tool for benchmarking DNS64 servers", *International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems*, vol. 7. no. 1. pp. 10-16. DOI: 10.11601/ijates.v7i1.255

[23] G. Lencse and D. Bakai, "Design, implementation and performance estimation of mtd64-ng, a new tiny DNS64 proxy", *Journal of Computing and Information Technology* vol. 25, no, 2, pp. 91-102, June 2017, DOI: 10.20532/cit.2017.1003419

[24] S. Bradner, J. McQuaid, "Benchmarking methodology for network interconnect devices", IETF RFC 2544, 1999. DOI: 10.17487/RFC2544

[25] C. Bao, C. Huitema, M. Bagnulo, M Boucadair and X. Li, "IPv6 addressing of IPv4/IPv6 translators", IETF RFC 6052, Oct. 2010. DOI: 10.17487/RFC6052

[26] Free Software Foundation, "The free software definition", [Online]. Available: http://www.gnu.org/philosophy/free-sw.en.html

[27] Open Source Initiative, "The open source definition", [Online]. Available: http://opensource.org/docs/osd

[28] G. Lencse and S. Répás, "Improving the performance and security of the TOTD DNS64 implementation", *Journal of Computer Science and Technology*, vol. 14, no 1, pp. 9-15, Apr., 2014.

[29] S. Perreault, J.-P. Dionne, and M. Blanchet, "Ecdysis: Open-source DNS64 and NAT64", *AsiaBSDCon*, 2010, [Online]. Available: http://viagenie.ca/publications/2010-03-13-asiabsdcon-nat64.pdf

[30] Ecdysis project, "Ecdysis: open-source implementation of a NAT64 gateway", download page, [Online]. Available: https://ecdysis.viagenie.ca/download.html

[31] EURid, "YADIFA", [Online]. Available: https://www.yadifa.eu

[32] NLnetLabs, "NSD: Name Server Daemon", [Online]. Available: https://www.nlnetlabs.nl/projects/nsd/
[33] G. Kunz, "Parallel discrete event simulation", In *Modeling and Tools for Network Simulation*, K. Wehrle, M. Günes and J. Gross (Eds.). Springer-Verlag, Berlin, 2010. ISBN 978-3-642-12330-6
[34] B. Gregg, "The flame graph", *Communications of the ACM*, vol. 59, no. 6, Jun. 2016, pp. 48-57, DOI: 10.1145/2909476
[35] S. Son and V. Shmatikov, "The hitchhiker's guide to DNS cache poisoning", in Proc. *Security and Privacy in Communication Networks - 6th Iternational ICST Conference (SecureComm 2010)*, Singapore, Sept. 7-9, 2010, pp. 466–483, DOI: 10.1007/978-3-642-16161-2_27
[36] J-P. Lozi, B. Lepers, J. Funston, F. Gaud, V. Quéma, A. Fedorova, "The Linux scheduler: A decade of wasted cores", *Proc. Eleventh European Conference on Computer Systems (EuroSys 2016)*, London, UK, April 18-21, 2016. DOI: 10.1145/2901318.2901326
[37] A. Hubert, R. van Mook, "Measures for making DNS more resilient against forged answers", IETF RFC 5452, Jan. 2009. DOI: 10.17487/rfc5452
[38] R. Bolla, R. Bruschi, F. Davoli, F. Cucchietti, "Energy efficiency in the future Internet: A survey of existing approaches and trends in energy-aware fixed network infrastructures", *IEEE Communication Surveys & Tutorials*, vol. 13, no. 2, pp. 223-244, Jul. 2010, DOI: 10.1109/SURV.2011.071410.00073
[39] Intel Xeon Phi Processor 7210, product specification, [Online]. Available: https://ark.intel.com/products/94033/Intel-Xeon-Phi-Processor-7210-16GB-1_30-GHz-64-core
[40] Michael Graff, "Performance: Multi-threaded I/O", ISC Knowledge Base, Reference Number: AA-00629, Created: 2012-02-24 18:17, Last Updated: 2017-08-01 20:08, [Online]. Available: https://kb.isc.org/article/AA-00629/0/Performance%3A-Multi-threaded-I-O.html

Gábor Lencse received his MSc and PhD in computer science from the Budapest University of Technology and Economics, Budapest, Hungary in 1994 and 2001, respectively.

He has been working full time for the Department of Telecommunications, Széchenyi István University, Győr, Hungary since 1997. Now, he is an Associate Professor. He has been working part time for the Department of Networked Systems and Services, Budapest University of Technology and Economics, Budapest, Hungary as a Senior Research Fellow since 2005. He was a Guest Researcher at the Laboratory for Cyber Resilience, Nara Institute of Science and Technology, Japan from June 15 to December 15, 2017, where his research area was the security analysis of IPv6 transition technologies.

Dr. Lencse is a member of IEICE (Institute of Electronics, Information and Communication Engineers, Japan).

Youki Kadobayashi received his Ph.D. degree in computer science from Osaka University, Japan, in 1997.

He is currently a Professor in the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. Since 2013, he has also been working as the Rapporteur of ITU-T Q.4/17 for cybersecurity standardization. His research interests include cybersecurity, web security, and distributed systems.

Prof. Kadobayashi is a member of IEEE Communications society.