Computer Communications 156 (2020) 54–67

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Performance analysis of SIIT implementations: Testing and improving the methodology

Gábor Lencse [a,*], Keiichi Shima [b]

[a] Department of Networked Systems and Services, Budapest University of Technology and Economics, Magyar tudósok körútja 2, Budapest, H-1117, Hungary
[b] IIJ Innovation Institute Inc., Iidabashi Grand Bloom, 2-10-2 Fujimi, Chiyoda-ku, Tokyo 102-0071, Japan

ARTICLE INFO

Keywords:
Benchmarking
IPv6 deployment
IPv6 transition solutions
SIIT
Stateless NAT64
Performance analysis

ABSTRACT

In this paper, the viability of the throughput and frame loss rate benchmarking procedures of RFC 8219 is tested by executing them to examine the performance of three free software SIIT (also called stateless NAT64) implementations: Jool, TAYGA, and map646. An important methodological problem of the two tested benchmarking procedures is pointed out: they use improper timeout setting. A solution of individually checking the timeout for each frame is proposed to get more reasonable results, and its feasibility is demonstrated. The unreliability of the results caused by the lack of requirement for repeated tests is also pointed out, and the need for relevant number of tests is demonstrated. The possibility of an optional non-zero frame loss acceptance criterion for throughput measurement is also discussed. The benchmarking measurements are performed using two different computer hardware, and all relevant results are disclosed and compared. The performance of the kernel based Jool was found to scale up well with the number of active CPU cores and Jool also significantly outperformed the two other SIIT implementations, which work in the user space.

1. Introduction

Stateless IP/ICMP translation (SIIT) [1] (also referred to as *stateless NAT64*) plays an important role in the current phase of transitioning from IPv4 to IPv6 as it is used in several contexts. For example, it is a part of the well-known *stateful NAT64* [2], which is used together with DNS64 [3] to enable IPv6-only clients communicating with IPv4-only servers. It works in the CLAT devices of 464XLAT [4], too. SIIT can also be applied to provide IPv4 access to IPv6-only data centers or services. One of its earliest such application is documented in [5].

Several free software [6] SIIT implementations exist and some of them support stateful NAT64, too. When network operators select the best fitting one for their purposes, they are interested in the performance of the different implementations. For carrying out performance measurements, one needs a well-defined methodology, measurement tools (hardware or software), a suitable testbed, benchmarking expertise, and a lot of time. As for methodology, RFC 8219 [7] defined a benchmarking methodology for IPv6 transition technologies. It classified the high number of IPv6 transition technologies [8] into a small number of categories, and it defined the benchmarking methods per categories. Both SIIT and stateful NAT64 fell into the category of *single translation solutions*, thus basically, they have the same benchmarking tests, plus some additional tests were defined for stateful IPv6 transition technologies (for the details, please refer to Section 8 of RFC 8219).

The aim of this paper is twofold:

- to test the viability of the benchmarking methodology defined in RFC 8219 concerning SIIT implementations, and also to amend it, where it proves to be necessary,
- to measure the performance of several SIIT implementations and thus provide network operators with ready to use benchmarking results.

The most important contributions of our paper are:

- The first demonstration of the feasibility of the RFC 8219 tests for the single translation solutions.
- Pointing out potential methodological problems in RFC 2544 [9] and RFC 8219, please refer to Section 3.8 for the details.
- Performance comparison of three SIIT implementations.

The remainder of this paper is organized as follows. In Section 2, we give a short introduction to the benchmarking method for single translation solutions defined in RFC 8219 and also give a brief survey of the related work including the available testing tools. In Section 3, first, we design the test and traffic setup, next, we give a very brief summary of the DPDK-based NAT64 Tester, then we describe our measurements, after that, we disclose and discuss our results, and finally, we point out some problems with the existing benchmarking methodology. In Section 4, we describe a different measurement method, which can

* Corresponding author.
E-mail addresses: lencse@hit.bme.hu (G. Lencse), keiichi@iijlab.net (K. Shima).

https://doi.org/10.1016/j.comcom.2020.03.034
Received 11 September 2019; Received in revised form 26 February 2020; Accepted 20 March 2020
Available online 27 March 2020

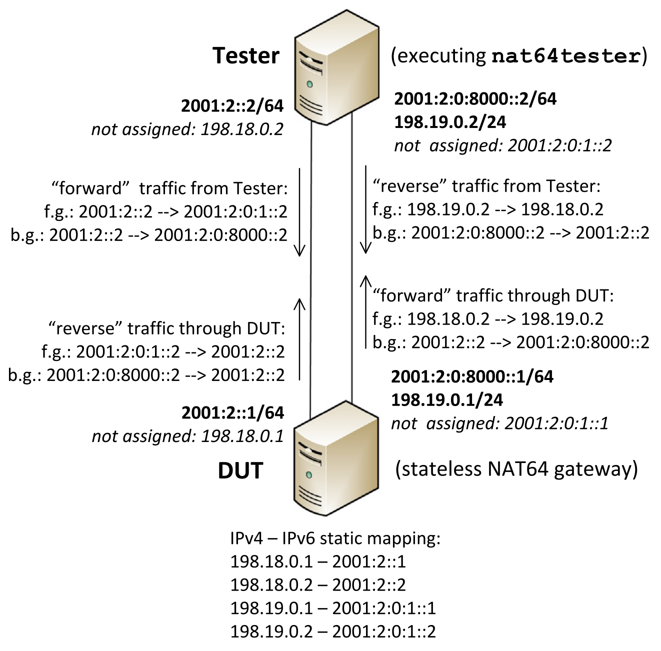
Tester
(executing nat64tester)
2001:2::2/64
not assigned: 198.18.0.2
2001:2:0:8000::2/64
198.19.0.2/24
not assigned: 2001:2:0:1::2
"forward" traffic from Tester:
f.g.: 2001:2::2 --> 2001:2:0:1::2
b.g.: 2001:2::2 --> 2001:2:0:8000::2
"reverse" traffic from Tester:
f.g.: 198.19.0.2 --> 198.18.0.2
b.g.: 2001:2:0:8000::2 --> 2001:2::2
"forward" traffic through DUT:
f.g.: 198.18.0.2 --> 198.19.0.2
b.g.: 2001:2::2 --> 2001:2:0:8000::2
"reverse" traffic through DUT:
f.g.: 2001:2:0:1::2 --> 2001:2::2
b.g.: 2001:2:0:8000::2 --> 2001:2::2
2001:2:0:8000::1/64
198.19.0.1/24
not assigned: 2001:2:0:1::1
2001:2::1/64
not assigned: 198.18.0.1
DUT
(stateless NAT64 gateway)
IPv4 – IPv6 static mapping:
198.18.0.1 – 2001:2::1
198.18.0.2 – 2001:2::2
198.19.0.1 – 2001:2:0:1::1
198.19.0.2 – 2001:2:0:1::2

Fig. 1. Test and traffic setup for benchmarking stateless NAT64 gateways.

complement our previous measurements and demonstrates the feasibility of our proposed solution for the methodology problem, then we present and discuss our results. In Section 5, we make three recommendations to update the two benchmarking procedures of RFC 8219 (RFC 2544) and disclose our plans for future research. Section 6 links ours raw results and measurement scripts for the sake of reproducibility. Section 7 concludes our paper.

2. Benchmarking SIIT gateways

2.1. Summary of the theoretical background

We give a short summary of the benchmarking methodology for SIIT (also called stateless NAT64) gateways on the basis of RFC 8219. We note that stateful NAT64 tests are basically the same with the exception that communication may be initiated only from the IPv6 side and there are some further tests for examining the stateful behavior.

The test setup is very simple: it consists of the Tester and the DUT (Device Under Test). The two devices are interconnected by two links as shown in Fig. 1. Testing with bidirectional traffic is required and testing with unidirectional traffic is optional.

The measurement procedure for the *throughput* test was taken verbatim from RFC 2544. It also means that the strict absolutely zero frame loss criterion of the throughput measurement was kept: the throughput is the highest rate, at which the number of frames transmitted by the Tester is equal with the number of frames received by the Tester, during an at least 60 s long test. The literal wording of the measurement procedure is as follows: "Send a specific number of frames at a specific rate through the DUT and then count the frames that are transmitted by the DUT. If the count of offered frames is equal to the count of received frames, the rate of the offered stream is raised and the test is rerun. If fewer frames are received than were transmitted, the rate of the offered stream is reduced and the test is rerun". [9] In practice, the highest such rate can be efficiently determined by a binary search, where the initial upper limit of the interval is the maximum frame rate of the media and the lower limit is zero. (For the general step of the binary search, the average of the upper and lower limits is calculated. A test is performed at that frame rate. If the test is successful, then the upper half interval is used. If the test fails, then the lower half interval is used.)

The recommended frames sizes are: 64, 128, 256, 512, 768, 1024, 1280, and 1518 bytes. And it is also mentioned that 64 should be replaced by 84 in the IPv6 to IPv4 direction due to minimum frame size issue. The RFC does not mention, but it is deliberate that 1518 should also be replaced by 1498 in the IPv4 to IPv6 direction due to maximum frame size issue. However, all other frame sizes are also changed by the translator and it is not mentioned, whether the other values (128, 256, 512, 768, 1024, 1280) are meant to be IPv4 or IPv6 frame sizes. As RFC 8219 also has tests, where translated and native IPv6 traffic is to be mixed, we suggest that the listed frame sizes should be used for IPv6. Thus, we interpret that the frame sizes for IPv6 are: 84, 128, 256, 512, 768, 1024, 1280, 1518, and frame sizes for IPv4 are 64, 108, 236, 492, 748, 1004, 1260, 1498.

As for the before mentioned mixed traffic, the SIIT gateway should act as a router for the native IPv6 traffic, and 100%, 90%, 50% and 10% are recommended for the proportion of the translated traffic, where the rest should be native IPv6 traffic.

The throughput measurement procedure only counts the number of the sent and received frames, but it does not identify them individually, thus it cannot check if their order is kept. Although our applications are sensitive to delay,[1] the throughput measurement procedure does not use any *per frame timeout*.[2] According to our interpretation, a very loose timeout is defined in Section 23 of RFC 2544 as follows: after running a particular test trial, one should "wait for two seconds for any residual frames to be received". We follow this approach in Section 3, and then challenge it in Section 3.8.

Section 12 of RFC 2544 also mentions that the tests should be performed first with a single flow (using a single source address and a single destination address) and then they should be repeated with 256 flows, where the destination addresses are randomly chosen from 256 different networks.

There is one more thing, in which *we have found a gap in the methodology of RFC 8219*. It is the *required number of repetitions* of the throughput and frame loss rate tests. RFC 8219 mentions at four different places that the tests must be repeated at least 20 times. These places are the benchmarking procedures for:

- latency (Section 7.2)
- packet delay variation (Section 7.3.1)
- inter packet delay variation (Section 7.3.2)
- DNS64 performance (Section 9.2).

As for DNS64 benchmarking measurements, we have explained the need for at least 20 repetitions in [10] as follows. "There may be random events, which influence the results. Consequently, the tests should be repeated multiple times and the final result should be calculated by using a particular summarizing function". The test was performed at least 20 times and we used the median value to summarize the results. To account for the variation of the results across the 20 repetitions, the 1st and 99th percentiles were used. It is also explained, that median was preferred over average because median is less sensitive to outliers than average.

In our case, the benchmarked SIIT implementations are software components executed by computers, thus we contend that the same conditions apply. Therefore, in this paper, we follow the same approach and we believe that this is the true spirit of RFC 8219, even if its literal wording does not say anything about the repetitions of the throughput and frame loss rate measurements. RFC 2544 writes in its Section 4 that: "Furthermore, selection of the tests to be run and

[1] It is trivial for real-time voice or video transmission, but it is also true for all applications using TCP, as TCP handles late segments as lost ones, thus significant latency of considerable proportion of the segments degrades the throughput experienced by the user (also called goodput).

[2] Under per frame timeout we mean that the frames sent by the Tester should arrive back to the Tester within a predefined time interval after their sending.

evaluation of the test data must be done with an understanding of generally accepted testing practices regarding repeatability, variance and statistical significance of small numbers of trials". As for repeatability, RFC 2330 [11] says that "A methodology for a metric should have the property that it is repeatable: if the methodology is used multiple times under identical conditions, the same measurements should result in the same measurements". It was true, when most of the switching devices were simple hardware based devices, where we can define some upper limit for packet processing. However, we use software based switching recently, and it sometimes has more variance in performance. Therefore, we need to find some appropriate method to understand the representative performance of such devices.

2.2. Related work and testing tools

We note that we are not aware of anyone else benchmarking SIIT implementations in an RFC 8219 compliant way. One of its causes is the *lack of compliant testers*. We know about a single publication only, which reported the design and implementations of such tester [12]. However, it has never been publicly released due to its insufficient performance. Rather, it was re-implemented using DPDK (Intel Data Plane Development Kit [13]) by its author, Péter Bálint, a PhD student at the Széchenyi István University, Győr, Hungary under the supervision of the first author of this paper. We planned to use this program for our measurements. However, our tests showed that this program failed to work correctly, and we have found fundamental problems in it during a systematic code review. We have rewritten its most important parts, namely the receiving and sending functions and their synchronization. Thus, the program became usable, and we could use it for the current paper. But we decided to re-implement it from scratch in C++ using a proper object oriented design. During the first review of this paper, we have successfully implemented `siitperf`, the world's first standard free software SIIT benchmarking tool and made it publicly available from GitHub [14] under the GPLv3 license. We have reported its design, implementation and initial performance estimation in [15].

Right before our current effort, we have examined the possibility of benchmarking stateless NAT64 implementations using a legacy RFC 2544/RFC 5180 [16] compliant Tester. We have reported our results in [17].

As for the performance analysis of not stateless but *stateful* NAT64 implementations, we are aware of several papers. In some early papers, the performance of a given NAT64 implementation was measured together with a given DNS64 implementation, please see [18,19] and [20]. We have pointed out that: "on the one hand this is natural, as both services are necessary for the operation, on the other hand this is a kind of 'tie-in sale' that may hide the real performance of a given DNS64 or NAT64 implementation by itself. Even though both services are necessary for the complete operation, in a large network, they are usually provided by separate, independent devices; DNS64 is provided by a name server and NAT64 is performed by a router. Thus, the best implementation for the two services can be – and therefore should be – selected independently" [21]. We have developed a method suitable for the performance analysis of NAT64 implementations independently from DNS64 [22], and we compared the performance of TAYGA and OpenBSD PF using ICMP in [23] and using ICMP, TCP and UDP in [24]. Another paper [25] dealt with the performance of different IPv6 transition solutions including the TAYGA and the Jool NAT64 implementations measuring one way delay and throughput (using `iperf` for the latter). All these papers were published before RFC 8219, and their measurement methods did not comply with RFC 8219.

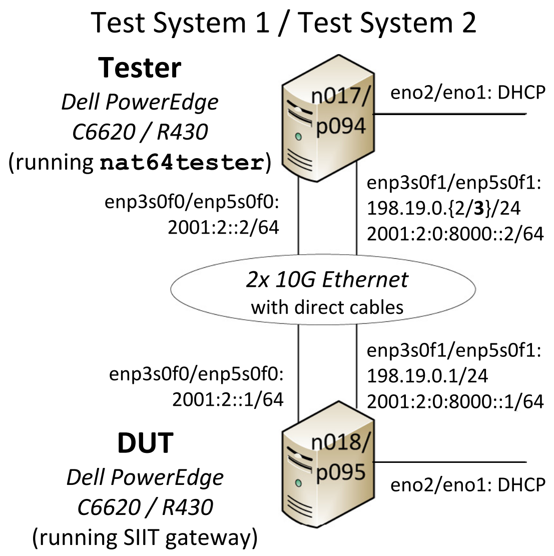
Test System 1 / Test System 2
Tester
Dell PowerEdge C6620 / R430
(running nat64tester)
n017/ p094
eno2/eno1: DHCP
enp3s0f0/enp5s0f0: 2001:2::2/64
enp3s0f1/enp5s0f1: 198.19.0.{2/3}/24 2001:2:0:8000::2/64
2x 10G Ethernet with direct cables
enp3s0f0/enp5s0f0: 2001:2::1/64
enp3s0f1/enp5s0f1: 198.19.0.1/24 2001:2:0:8000::1/64
DUT
Dell PowerEdge C6620 / R430
(running SIIT gateway)
n018/ p095
eno2/eno1: DHCP

Fig. 2. Test System 1 is built up by N nodes, with a fixed 2 GHz CPU clock frequency DUT/Test System 2 is built up by P nodes, with a variable (1.2–3 GHz) CPU clock frequency DUT.

Table 1
The building elements of the test systems for basic tests.

Test system	Tester		Device Under Test (DUT)		
	Node	Speed	Node	Speed	Active cores
TS1	n017	2–2.8 GHz	n018	2 GHz	0–7
TS2	p094	1.2–3 GHz	p095	1.2-3 GHz	0–31

3. Benchmarking measurements with a stateless NAT64 tester

3.1. Test and traffic setup

Following the requirements of RFC 8219, we have designed the test and traffic setup for benchmarking stateless SIIT gateways using EAM (Explicit Address Mapping) [26]. Both the traffic to be translated and the native IPv6 traffic are shown in Fig. 1. We call the first one and the second one as *foreground traffic* (abbreviated as f.g.) and *background traffic* (abbreviated as b.g.), respectively. Concerning the foreground traffic, only IPv6 addresses are assigned to the left side network interfaces of both the Tester and the DUT (from 2001:2::/64), and similarly, their network interfaces on their right side have only IPv4 addresses (from 198.19.0/24). The addresses typeset in italic font are not assigned to the interfaces, they are written there to help the reader to follow the operation of the system. They are used to refer to the given interfaces in the other address space (IPv4 or IPv6). DUT translates the addresses according to its *static mapping table* shown below the DUT. We call the traffic from the IPv6 interface of the Tester flowing through DUT and arriving to the IPv4 interface of the Tester as "forward" direction traffic, and we call the other direction as "reverse" direction, because RFC 8219 requires the use of native IPv6 traffic, too, thus in that case the terms of "IPv6 side" and "IPv4 side" would have been questionable. Concerning the native IPv6 traffic (using the 2001:2::2 and 2001:2:0:8000::2 IPv6 addresses), the DUT acts as a router.

3.2. NAT64 tester in a nutshell

We give a very brief summary of the functional design of the DPDK-based NAT64 tester (called `nat64tester`) used for our measurements.

Following the high level design of Dániel Bakai's excellent DNS64 tester called `dns64perf++` [27], `nat64tester` performs only one test, and the binary search, as well as the further repetitions are executed by a `bash` shell script. The functionality of `nat64tester` is rather limited, as it can only be used for the two most important tests, namely *throughput* and *frame loss rate*. A further limitation is that it can only perform a single flow test. (We overcome this limitation in Section 4 by using `dns64perf++`, which is able to use up to 64,000 different source ports.)

When the test is finished, `nat64tester` reports the number of sent and received frames in each direction, and the shell script evaluates the results.

- If a throughput measurement is done, the shell script checks if the number of received frames is equal with the number of frames had to be sent by the tester at the required rate during the required testing time and makes the decision for the binary search.
- If a frame loss measurement is done, the shell script determines the frame loss using the number of frames received and the calculated number of frames had to be sent.

The script does not use the reported value of the number of frames sent, but it is logged to help error debugging.

Unfortunately, `nat64tester` is not able to reply to ARP or ND requests, therefore, we used direct cable connections between the Tester and the DUT, and static ARP/ND table entries were set manually.

3.3. Measurement environment

Measurements were carried out using the resources of the NICT StarBED,[3] Japan. Two different types of servers (*N nodes* and *P nodes*) were used.

- The N nodes are Dell PowerEdge C6220 servers with two 2 GHz Intel Xeon E5-2650 CPUs having 8 cores each, 128 GB 1333 MHz DDR3 RAM and Intel 10G dual port X520 network adapters.
- The P nodes are Dell PowerEdge R430 servers with 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.

We have used two very similar tests systems with somewhat different goals. In *Test System 1* (see Fig. 2 and Table 1), we switched off hyper-threading in both computers and set the clock frequency of the DUT to 2 GHz (fixed), because we knew from our previous benchmarking experience [28] that they could cause *scattered measurement results*. (We mean under scattered measurement results that the results of the 20 measurements are significantly different.) Our aim with Test System 1 was to eliminate all circumstances that could cause scattered measurement results. However, Turbo Mode was enabled on the Tester to give some extra performance. (In such case, the power budget is a limit for the clock frequency of the cores. We have checked that the clock frequency could reach 2.8 GHz, when no more than 4 cores were used, and `nat64tester` uses 4 cores for bidirectional tests and 2 cores for unidirectional tests.)

In the Tester (n017), we have reserved cores 4–7 to execute `nat64tester`, using the `isolcpus=4,5,6,7` kernel parameter. (It means that no other user tasks could be scheduled on these cores.)

In the DUT (n018), we have limited the online CPU cores to cores 0–7, using the `maxcpus=8` kernel parameter to avoid possible NUMA issues. (It was done so, because all the I/O devices, as well as cores 0–7 belonged to NUMA node 0. Scheduling sometimes the SIIT implementation on one of the cores 8–15, which belonged to NUMA node 1, could have resulted in a decreased performance and thus scattered measurement results, which we wanted to avoid.)

Our aim with *Test System 2* (see Fig. 2) was to test the same implementations on a more modern CPU, the clock frequency of which may not be set to a fixed value. In addition to that, CPU cores 0, 2, 4, …, and 30 belonged to NUMA node 0, and cores 1, 3, 5, …, and 31 belonged to NUMA node 1. All NICs and disks belonged to NUMA node 0.

We have disabled hyper-threading in both computers. Plus, in the Tester (p094), we have reserved cores 2, 4, 6, and 8 to execute `nat64tester`, using the `isolcpus=2,4,6,8` kernel parameter. The CPU clock frequency of both computers could vary from 1.2 GHz to 3 GHz, which is the maximum turbo frequency of the CPU. We have changed the “powersave” CPU frequency scaling governor (`cpufrequtils`) to “performance” in both computers.

Besides the different node and interface names, the reader may notice a small but significant difference between Test System 1 and Test System 2 in Fig. 2. The last octet of the IPv4 address of the tester was set to 3 in the latter. We explain its reason, when disclosing the results of the throughput test in Section 3.5.1.

The Debian Linux operating system was updated to 9.9. (the latest version at the time of testing) and the kernel version was 4.9.0-4-amd64 and 4.9.0-8-amd64 on the N nodes and on the P nodes, respectively. The DPDK version was 16.11.8-1+deb9u1.

3.4. SIIT Implementations to be benchmarked

We deal only with free software [6] SIIT implementations for the same reasons we presented in [29]:

- “The licenses of certain vendors (e.g. [30] and [31]) do not allow the publication of benchmarking results.
- Free software can be used by anyone for any purposes, thus our results can be helpful for anyone.
- Free software is free of charge for us, too”.

We have made a survey of existing free software stateless NAT64 implementations in [17]. Now, we decided to benchmark the same implementations using both different DUTs and a different tester. This situation gives us both a basis for comparison and an opportunity to dig deeper into the behavior of the tested implementations. The implementations and software versions for our current benchmarking test are:

- TAYGA 0.9.2 (released on June 10, 2011) [32], Debian package version: 0.9.2-6+b1
- Jool 4.0.1 (released on April 26, 2019) [33]
- map646 (GitHub latest commit cd93431 on Mar 31, 2016) [34]

We note that in our previous paper [17], we tested Jool 3.5.7. Now, we have also checked its performance during our preliminary test on the N node (its DKMS build failed on the P node), but having seen no major differences, we have omitted the old version. The versions of the two other SIIT implementations were the same as now. As for the rationale for choosing these three SIIT implementations, first of all, we could not find any other free software SIIT implementations under Linux (only stateful NAT64). We note that Jool is still actively developed, TAYGA is no more developed but it is a part of the Debian Linux distribution, and it seems to be still in use, because we have found several posts from the last three years about how to configure TAYGA. Map646 was created by the second author of this paper, and we were interested in its performance, because it is still in use as the NAT46 gateway for the WIDE project [5].

To make our tests repeatable, we give the most important information in the Appendix, how we set the different SIIT implementations.

3.5. Throughput test results

First, we disclose and analyze the throughput test results of each implementation individually, and then we compare them and discuss our most important findings.

[3] http://starbed.nict.go.jp/en/aboutus/index.html.

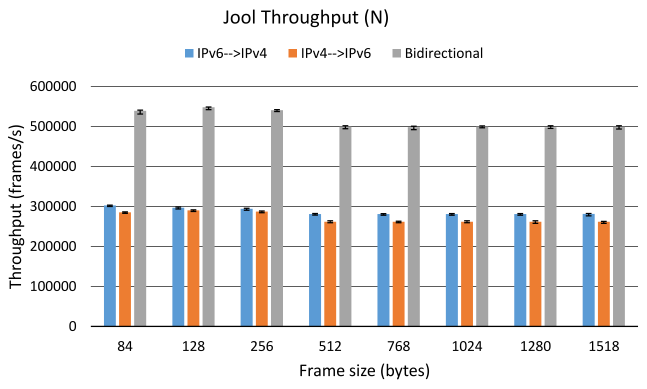

Fig. 3. Throughput results of Jool, TS1.

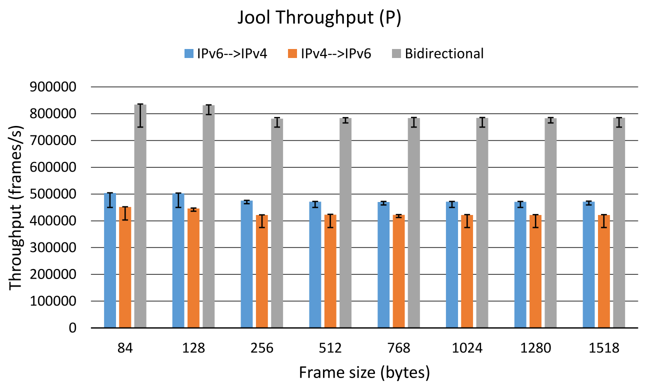

Fig. 4. Throughput results of Jool, TS2.

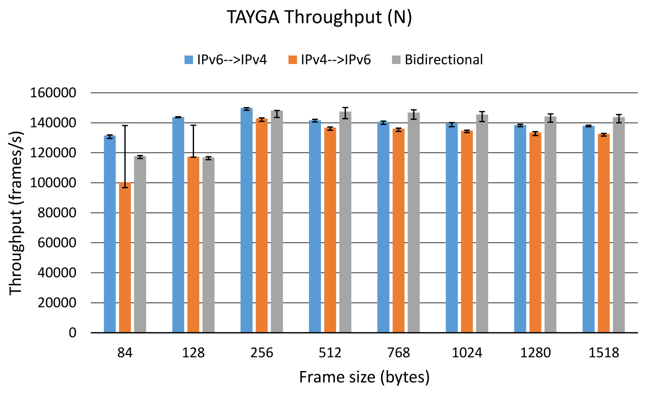

Fig. 5. Throughput results of TAYGA, TS1.

As we wrote in Section 2, we have executed all the measurements 20 times and calculated the median as well as the 1st and 99th percentiles. (Of course, the latter two are also the minimum and maximum, as the number of measurements is less than 100.)

All the results are presented in the same format: bar charts are used for displaying the median values, and error bars show the 1st and 99th percentile values.

3.5.1. Jool

The throughput results of Jool produced by TS1 (Test System 1) are shown in Fig. 3. The error bars are hardly visible, because the 1st percentile and 99th percentile values are very close to the median. Thus we were definitely successful in the elimination of all possible factors that could cause scattered results. The throughput values are nearly constant, they show only a very slight decreasing tendency with the increase of the frame size. This observation is in a complete agreement with our previous results [17], and it can be explained by the fact that the bottleneck is the processing power of the CPU and not the transmission capacity of the 10Gbps Ethernet link. (The amount of work needed for header processing does not depend on the frame size and the transmission through the PCI Express bus is also very fast.)

Let us examine the exact figures for a given frame size, for example 128 bytes, which actually means 128 bytes long Ethernet frames carrying IPv6 datagrams and 108 bytes long Ethernet frames carrying IPv4 datagrams. The median throughput values of the forward (form IPv6 to IPv4), reverse (from IPv4 to IPv6) and bidirectional traffic are 296,972 fps (frames per second) 290,234 fps, and 547,848 fps, respectively. The observation that the bidirectional throughput is 5.62% less than the double of the minimum of the unidirectional throughput (580,468 fps) can be explained by the fact that although Ethernet is full duplex and the packets in the two directions are handled by two separate CPU cores, some other resources (e.g. the memory and the PCI express bus) are shared.

The throughput results of Jool produced by TS2 (Test System 2) are shown in Fig. 4. The error bars are usually well visible, indicating that the 1st percentiles are significantly lower than the 99th percentiles. We attribute this scattered nature of the results to the varying clock frequency of the CPU, because during the preliminary tests, we have observed that always the same CPU cores were loaded,[4] thus differences in the NUMA situation can be excluded as potential causes of the differences.

We note that originally we used TS2 with the same IP addresses as TS1. In that case, the two flows of the bidirectional tests were served by the same CPU core, and thus the throughput results were lower. However, in a real system, usually a lot of different IP addresses are used, thus this clash is not a typical behavior, therefore, we have eliminated it by changing the IP addresses.

Otherwise, the results are very similar to that of the measurements with the N nodes, but of course, the values are higher, due to the higher CPU clock frequency.

3.5.2. TAYGA

The throughput results of TAYGA produced by TS1 are shown in Fig. 5. One of the most salient thing is that the bidirectional throughput is rather low compared to the single directional ones. It is so because TAYGA can utilize only a single CPU core.

There are visible problems at 84 bytes and 128 bytes frame sizes: the throughput is visibly lower than it is at 256 bytes, and the error bars of the IPv4 to IPv6 traffic are very high, indicating significantly scattered measurement results. (For example, the 1st percentile is 96,777 fps and the 99th percentile is 138,183 fps at 84 bytes frame size.) Behind this phenomenon, we surmise some stability problems of TAYGA, and we show it in Section 3.6.2, when discussing its frame loss rate results. However, from the viewpoint of the theory of benchmarking it is much more important that *this situation demonstrates the need for multiple tests*.

Otherwise, the results from 256 bytes to 1518 bytes frame sizes are nearly constant, showing very small degradation with the increase of the frame size. (Considering that TAYGA works is user-space, we could easily accept even higher degradation than that.)

The throughput results of TAYGA produced by TS2 are shown in Fig. 6. Similarly to Jool, the P node results are more scattered than the N node results (the error bars are well visible) as expected. However, the problem observed in the N nodes results at 84 bytes and 128 bytes frame sizes is hardly noticeable. (Only the bidirectional throughput shows some deviation: the median value at 84 bytes is visibly less, than is should be and the error bar at 128 bytes is higher than at any other places). We consider this phenomenon to be a good example,

[4] In fact, we could observe only the load caused by software interrupts, as Jool works in the Linux kernel.

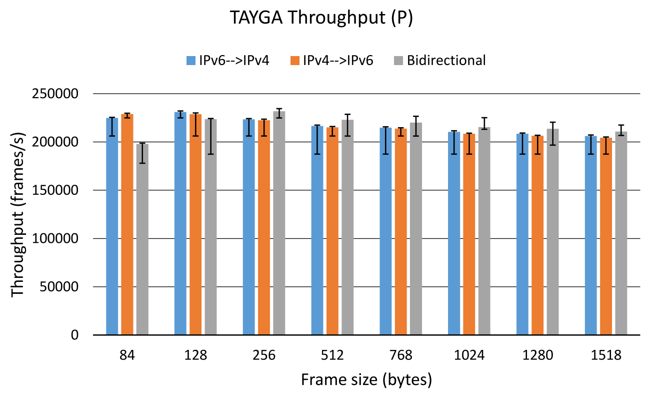

Fig. 6. Throughput results of TAYGA, TS2.

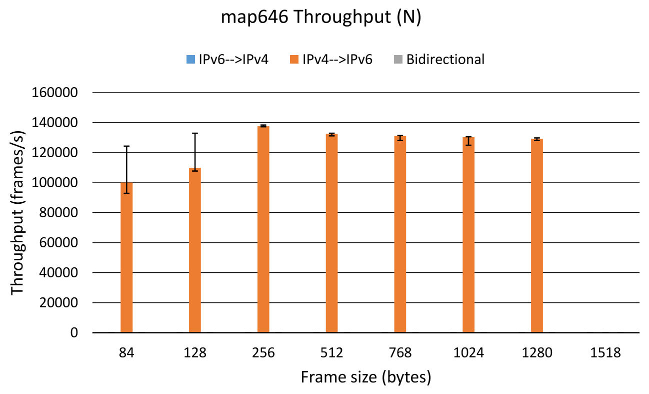

Fig. 7. Throughput results of map646, TS1.

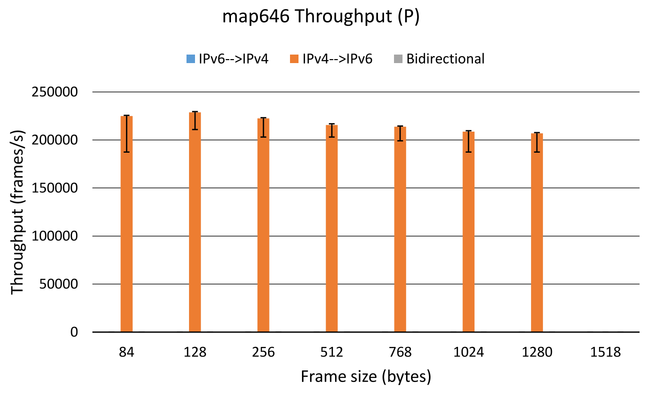

Fig. 8. Throughput results of map646, TS2.

why it is worth repeating benchmarking tests using different types of hardware: not only the measured performance may be different, but different anomalies may be pointed out, which may remain hidden on other systems.

3.5.3. Map646

The throughput results of map646 produced by TS1 are shown in Fig. 7. Similarly to TAYGA, there are visible problems at 84 bytes and 128 bytes frame sizes: the throughput is visibly lower than at 256 bytes, and the error bars are very high, indicating significantly scattered measurement results.

The throughput result at 1518 bytes is missing because all frames were lost due to (wrongful) fragmentation.

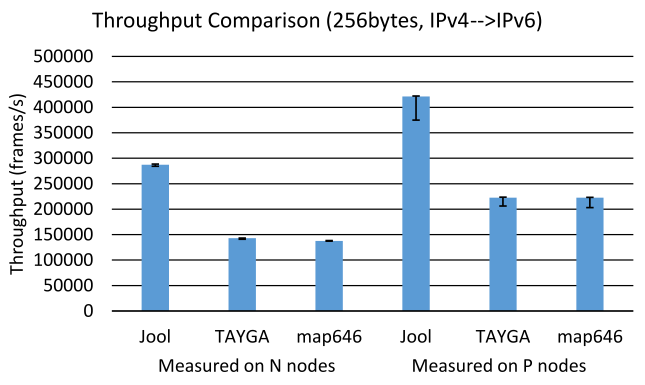

Fig. 9. Throughput comparison with 256 bytes frames, IPv4 to IPv6 direction.

Otherwise, the results from 256 bytes to 1280 bytes frame sizes are nearly constant, showing only a small degradation with the increase of the frame size.

The throughput results of map646 produced by TS2 are shown in Fig. 8. Unlike with TS1, here the median throughput at 84 bytes and 128 bytes frame sizes is very similar to the throughput at other frame sizes, only the somewhat higher error bar at 84 bytes reminds us the problem observed on TS1.

3.5.4. Comparison and discussion

To be able to compare all three implementations, we considered the IPv4 to IPv6 throughput and within that, we have chosen the throughput values measured with 256 bytes long frames to exclude the effect of the strange behavior of TAYGA and map646 at the two shortest frame sizes.

The results are shown in Fig. 9. Jool has significantly outperformed both TAYGA and map646, which was exactly what we expected, because Jool works in the kernel space and the other two implementations work in the user space.

We note that the choice of the throughput results with 256 bytes frame size was in favor of TAYGA and map646 over Jool.

3.6. Frame loss rate tests

According to RFC 2544, frame loss rate tests should be performed for all frame sizes, which means very high number of measurements to perform and results to evaluate. Knowing that throughput has shown only a very slight decrease with the increase of the frame size, we considered performing all possible measurements as pointless in our particular case.[5] Therefore, we have chosen two frame sizes, 128 bytes and 1280 bytes for testing. (Frame size 128 falls into the range, where TAYGA and map646 showed strange behavior, whereas 1280 is the largest frame size that can be tested with all three implementations, and it is exactly 10 times 128.) To have comparable results, we performed the frame loss tests in the IPv4 to IPv6 direction for all three implementations.

As for the measurement procedure described in RFC 2544, frame loss test should be performed for different frame rates starting from the maximum frame rate of the media, decreased in not more than 10% steps until two consecutive measurements show zero frame loss. However, in our case the maximum frame rate of the 10Gbps Ethernet with 128 bytes frame size is 8,445,945 fps, which is more than an order of magnitude higher than any of the tested implementations could achieve, thus such results would be meaningless. We have chosen more realistic ranges, which ensured us meaningful results for all three tested implementations. They are: from 50,000 fps to 500,000 fps using 50,000 fps steps.

[5] Of course, it is meaningful in other cases, e.g. when the bottleneck is the transmission capacity and not the CPU power.

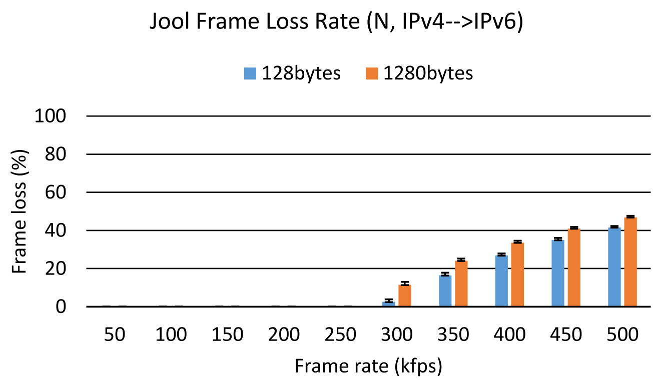

Fig. 10. Frame loss rate results of Jool, TS1.

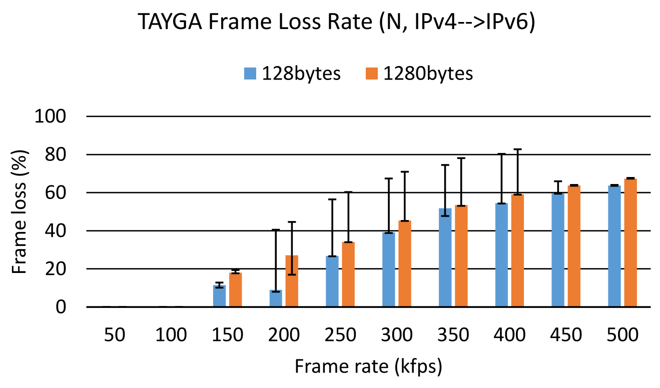

Fig. 11. Frame loss rate results of TAYGA, TS1.

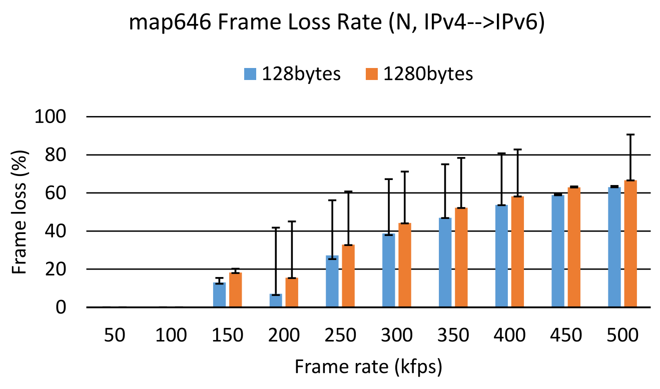

Fig. 12. Frame loss rate results of map646, TS1.

3.6.1. Jool

The frame loss rate results of Jool measured on TS1 are shown in Fig. 10. In agreement with Fig. 3, no frame loss occurred from 50k fps to 250 kfps. From 300 kfps rate, an increasing frame loss rate can be observed. For any given frame rate, the frame loss rate with 1280 bytes frames were higher than with 128byte frames, which also complies with the results shown in Fig. 3: the throughput expressed as frames per second had a slightly decreasing tendency with the growth of the frame size. And, what is not shown directly on the graph, but we could see from the raw data: from 300 kfps, the number of frames transmitted during the 60 s long tests were approximately constant, about 17.5 million and 15.9 million with 128 bytes and 1280 bytes frames, respectively. (The low measure of the deviations can be seen from the tiny sizes of the error bars.) Thus, we can lay down that Jool has shown a very stable behavior even under serious overload conditions.

3.6.2. TAYGA

The frame loss rate results of TAYGA measured on TS1 are shown in Fig. 11. Whereas the median values roughly comply with what could be expected on the basis of the previous results (frame loss appears from 150,000 fps, its tendency is increasing, and the frame loss is higher for the longer frames), the most conspicuous thing is the appearance of the high error bars from 200 kfps to 400 kfps caused by some outliers. This result is in a complete agreement with the high fluctuation of the throughput results of TAYGA in the IPv4 to IPv6 direction at 128 bytes shown in Fig. 5. However, the throughput results on the same figure are very stable at 1280 bytes. Yet the frame loss rate (and thus also the throughput) shows high fluctuations under serious overload conditions with 1280 bytes long frames.

3.6.3. Map646

The frame loss rate results of map646 measured on TS1 are shown in Fig. 12. They are very similar to that of TAYGA with an exception that here a high error bar appears also at 500 kfps rate with 1280 bytes.

3.7. Tests with mixed traffic

Similarly to the frame loss rate tests, we have also used two selected frame sizes, 128 and 1280 bytes.

RFC 8219 requires the usage of 100%, 90%, 50%, and 10% translated traffic and the remainder should be native IPv6 traffic. In addition to that, we found that it was worth using also 75% and 25% as translated traffic, as well as native IPv6 traffic as reference.

3.7.1. Jool

The throughput results of Jool using mixed traffic measured on TS1 are shown in Fig. 13. It can be observed that the throughput with 25% and 10% translated traffic is higher than the throughput of IPv6 routing with 0% translated traffic. The explanation is very simple: the translated and the native IPv6 traffic made two different flows, and thus they were processed by two distinct CPU cores. To set up a simple model for the throughput of the mixed traffic, let us use the following notations:

T_t throughput of the translated traffic

T_n throughput of the native IPv6 traffic

T_m throughput of the mixed traffic

α proportion of the translated traffic, where $\alpha \in [0,1]$, and the proportion of the native IPv6 traffic is $1-\alpha$.

Then the mixed throughput can be expressed as follows:

$$T_m(\alpha) = min\left(\frac{T_t}{\alpha}, \frac{T_n}{1-\alpha}\right) \quad (1)$$

The dispersion of the results is low, and the throughput with 1280 bytes long frames is somewhat lower than with 128 bytes frames.

3.7.2. TAYGA

The throughput results of TAYGA using mixed traffic measured on TS1 are shown in Fig. 14. Here only the throughput of the 1280 bytes long frames with 10% translated traffic is higher than the throughput of the native IPv6 traffic, which can be easily explained by Eqn. (1) and the actual values of T_t and T_n. As for the high error bar here, we have checked the raw results and found their distribution to be bimodal: 12 of the 20 results were higher than 650 kfps, and 7 of them were below 460 kfps.

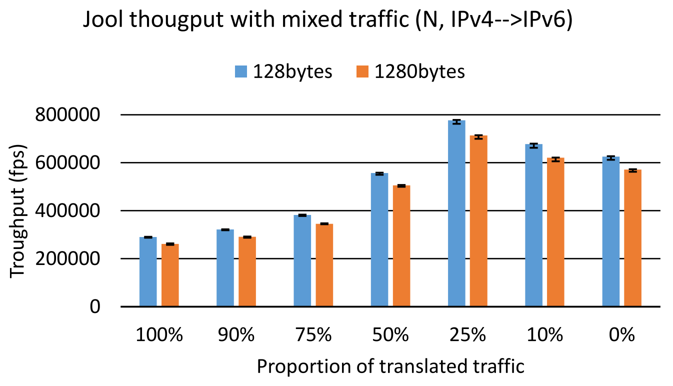

Fig. 13. Throughput results of Jool with mixed traffic (the rest is native IPv6), TS1.

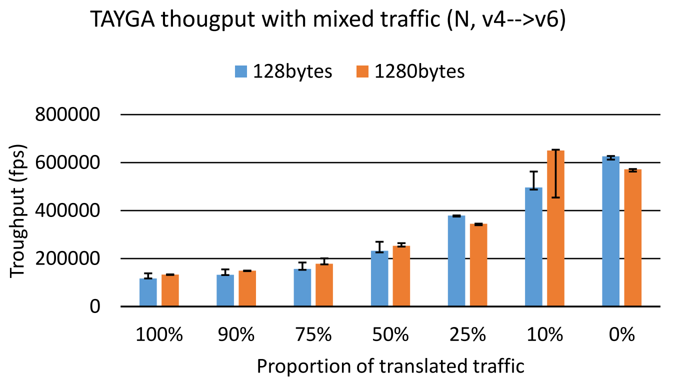

Fig. 14. Throughput results of TAYGA with mixed traffic (the rest is native IPv6), TS1.

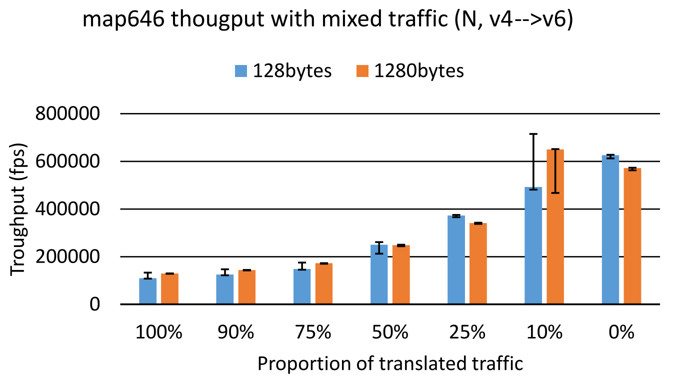

Fig. 15. Throughput results of map646 with mixed traffic (the rest is native IPv6), TS1.

3.7.3. Map646

The throughput results of map646 using mixed traffic measured on TS1 are shown in Fig. 15. The high error bars at 10% translated traffic show that even 10% translated traffic may significantly influence the overall results of a measurement. It was possible because the loss of even a single frame results in the failure of the complete test.

3.7.4. Comparison and discussion

Although the three figures look somewhat different (Jool showed its maximum throughput value at 25% translated traffic, whereas the other two implementations did it at 10%) they all followed the same rule shown in Eqn. (1). We note that this behavior is the consequence of the conditions that we used a single translated flow and a single background flow and multiple CPU cores. Using only a single CPU core or high number of flows, the throughput of the mixed traffic would follow the rule shown in Eqn. (2), that is, the mixed throughput would be the weighted harmonic mean of the throughput of the translated traffic and the throughput of the native IPv6 traffic.

$$T_m(\alpha) = \frac{1}{\frac{\alpha}{T_t} + \frac{1-\alpha}{T_n}} \tag{2}$$

3.8. Challenging the throughput test of RFC 2544

The throughput results obtained by RFC 2544 testers do not necessarily accord with the experiences of the users. On the one hand, the absolutely zero loss criterion *may be too strict*, as some low frame loss rates (e.g. 10^{-4} or 10^{-5}) do not prevent communication. Indeed, some benchmarking professionals use 99.999% throughput (or 0.001% loss) as their “zero loss” criterion [35]. However, on the other hand, we contend that the RFC 2544 benchmarking procedure *is way too lax*. Frames are sent for 60 s and received for 62 s. We understand, that the additional 2 s timeout was probably set to surely receive also the very last frames. However, it also means that the very first frame has theoretically 62 s timeout. This is completely unacceptable from the user point of view. Both TCP and real-time UDP applications will time out much–much earlier than that! In order to assess the throughput experienced by the users, one should use *per packet timeout*. We are aware that RFC 2544 was published in 1999 and its procedures could rely on the *then available technologies* only. In 2008, RFC 5180 updated some of the technology dependent parts, for example, it defined the maximum frame rate for 10Gbps Ethernet, however, it left the measurement procedures unchanged. In 2017, RFC 8219 defined benchmarking methodology for IPv6 transition technologies. It has redefined the procedure for measuring latency to achieve more accurate results, and added procedures for measuring packet delay variation and inter packet delay variation, but the throughput and frame loss rate measurement procedures were still kept unchanged.

We strongly argue that in 2019, the available technology makes it possible to use per packet timeout and we highly recommend it. We demonstrate its feasibility in Section 4. Similarly, we also believe that our results presented above have sufficiently demonstrated that RFC 2544 throughput and frame loss rate tests must be updated regarding the number of repetitions, the summarizing function and the way of expressing the dispersion of the results.

4. Benchmarking measurements with dns64perf++

Unfortunately, the missing functions of `nat64tester` limited our investigations. In this section, we use another testing tool, `dns64perf++`, to perform further tests with the following aims:

- to demonstrate the feasibility of the measurements using per packet timeout,
- to perform tests using high number of flows (which is unfortunately not supported by `nat64tester`),
- to benchmark map646 using bidirectional traffic.

First, we give a short introduction to the benchmarking tool, then we disclose the test setup, next continue on with various self-tests, and then start testing the SIIT implementations.

4.1. The applied testing tool

The `dns64perf++` testing tool was designed for benchmarking DNS64 servers [27]. Originally, it used only two threads, one for sending the queries and one for receiving the replies. Its performance was about 250,000 queries per second executed by a 3200MHz Intel Core i5-4570 CPU [27]. Later it was enabled to use n times two threads (n threads for sending queries and n threads for receiving replies) and thus we could use it up to 3.3 Mqps (3.3 million queries per second) rate for benchmarking authoritative DNS servers [36]. It has two significant advantages over `nat64tester`:

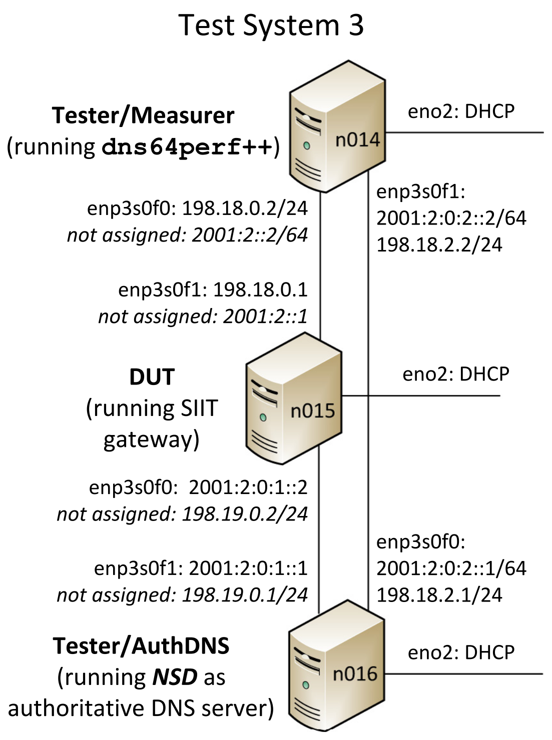

Fig. 16. Test System 3 is built up by N nodes, with a fixed 2 GHz CPU clock frequency DUT.

- It can individually identify every single DNS reply and check, if it arrived within the predefined timeout time after the corresponding query was sent.
- It can use a high number of different source port numbers for the queries and thus facilitate two very important things:
  - the distribution of the interrupts caused by the incoming packets to all the CPU cores, which is a precondition for receiving several million packets per second [37] and which we can use for testing with multiple flows,
  - the distribution of the queries among multiple threads or processes of the DNS server, if it uses the `so_reuseport` socket option, thus providing us with a high speed responder for our tests.

From the SIIT gateway point of view, the DNS queries and replies are UDP packets, which can be translated as any other packets. The `dns64perf++` program is used as a Tester, which sends and receives packets, the content of which is redundant from the DUT's point of view. Of course, it has some limitations, too. One of them is the fixed size of the queries and their replies. In general, this could be a serious limitation, but in our case, this limitation is not at all a significant one, as our previous tests showed no significant difference in the achieved frame rates for different frame sizes. The limitation caused by the usage of the authoritative DNS server is twofold: its maximum reply rate limits the maximum frame rate of the SIIT measurements, and the time necessary for the DNS server to produce a reply reduces the accuracy of the SIIT measurement concerning timeout.

4.2. Measurement environment

We used some of the N nodes of StarBED, but actually different ones than before to enable concurrent testing.

In our paper about the benchmarking methodology for DNS64 benchmarking [10], we have elaborated that the two subsystems of the Tester may be implemented by two physical computers, when high performance is needed. We have followed this approach during building *Test System 3* shown in Fig. 16. The DUT can be found in the middle of the system: `n015` was used to execute the tested SIIT implementations. The `dns64perf++` program was executed by `n014` using its CPU cores 0–7 for executing the 8 sending threads and cores 8–15 for executing the 8 receiving threads. Based on our authoritative DNS server benchmarking results [36], the NSD authoritative DNS server was installed on the `n016` node to provide authoritative DNS service. It was set to use all 16 cores of the computer.

In accordance with our previous measurements, this system was set to be protected from any possible disturbance that could cause scattered measurement results. The CPU clock frequencies of both `n015` and `n016` were set to fixed 2 GHz, the computers were interconnected by direct cables, and only cores 0–7 of `n015` were online (to avoid NUMA issues). Although turbo mode was enabled on `n014`, the power budget allowed to raise its CPU clock frequency up to 2.4 GHz only, when all its cores were used. We have summarized the most important building elements of Test System 3 in Table 2.

The IP addresses were set according to Fig. 16. The network traffic through the SIIT gateway was now initiated from the IPv4 side, so that map646 could also be tested with bidirectional traffic.

The direct connection between the two subsystems of the Tester served the purposes of the *self-test of the Tester* [10]. In short, it was used to check, up to what query rates the Tester could be used. We have performed these tests using both IPv4 and IPv6 for carrying the DNS requests and replies, as we knew from our earlier experience that the achievable query per second rate was significantly lower with IPv6.

In addition to that, we have also performed *routing tests*, when the DUTs were used as IPv4 or IPv6 routers. The routing tests had three purposes:

- to check and demonstrate that the paths for the SIIT measurements are working properly and having no bottleneck,
- to assess their maximum performances in IPv4 and IPv6 packet forwarding,
- to test and demonstrate, how the number of flows influence the performance of the system.

For the routing tests, all the IP addresses shown in Fig. 16 (including those typeset in italic) were assigned to the NICs. All the tests (including self-test, routing tests and the SIIT measurements) were performed using 64,000 different source ports by `dns64perf++` (8000 ports per thread by 8 sending threads).

To distribute the interrupts evenly among the CPU cores, the following commands were used:

```
ethtool -N enp3s0f0 rx-flow-hash udp4 sdfn
ethtool -N enp3s0f0 rx-flow-hash udp6 sdfn
ethtool -N enp3s0f1 rx-flow-hash udp4 sdfn
ethtool -N enp3s0f1 rx-flow-hash udp6 sdfn
```

Static IPv4 and IPv6 routes were set in the computers acting as the two subsystems of the Tester. The settings of the three tested SIIT implementations were only slightly different than in Section 3, thus we do not repeat them.

First, we performed the routing tests with excluding the source ports from the `rx-flow-hash` (by using `sd` instead of `sdfn`), thus there were only two flows (due to the bidirectional traffic).

We note that `dns64perf++` measures the number of successfully resolved *queries per second*, and we used this unit during the calibration of the test system. We switched over to *frames per second*, when benchmarking the different SIIT implementations in order to have our results comparable with the previous ones. The conversion is very simple: each resolved query means two frames: the first one carried the query and the second one carried the answer for the query.

Table 2
The building elements of Test System 3.

Test system	Tester/Measurer			Tester/AuthDNS			Device Under Test (DUT)			Connection of the elements
	Node	Speed	Active cores	Node	Speed	Active cores	Node	Speed	Active cores	
TS3	n014	2–2.4 GHz	0–15	n016	2 GHz	0–15	n015	2 GHz	0–7	Direct cables

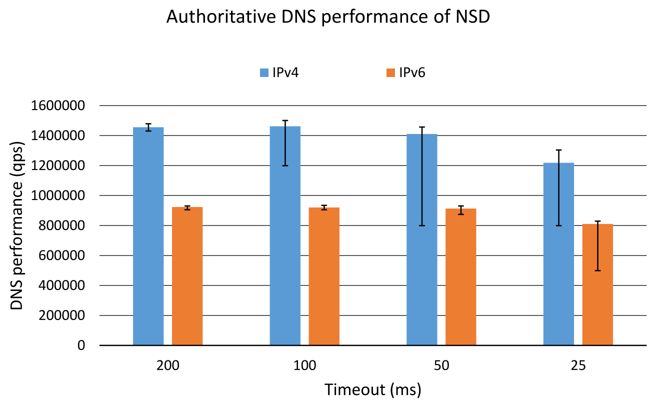

Fig. 17. Self-test results as a function of timeout time.

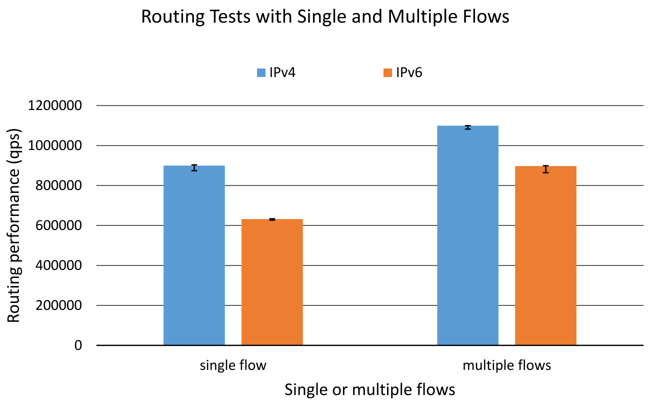

Fig. 18. Results of the routing tests (beware that the unit is qps and not fps).

4.3. Self-test measurements

Similarly to the self-test measurements of the DNS64 benchmarking tests [10], the aim of these measurements was to determine the performance of the Tester, in order to ensure that the DUT be the bottleneck during all further measurements. However, now we did not have a predefined timeout value, rather we had to select a suitable one, which was high enough to ensure as high as possible query rates on the one hand, however, on the other hand, it had to be low enough for performing meaningful SIIT measurements. The only hint we had from our previous experience [36] was that 100 ms was likely workable from the authoritative DNS server point of view.

We have performed self-test measurement using 200 ms, 100 ms, 50 ms, and 25 ms timeout values and both IPv4 and IPv6 as transport protocol for carrying the DNS messages. The results are shown in Fig. 17. (The upper limit of the binary search was set to 1,600,000 qps and 1,000,000 qps for IPv4 and IPv6, respectively, thus they did not limit the results.) To avoid that the unsatisfactory performance of the AuthDNS subsystem impact the measurement results, the 1st percentiles (that is the minimum values) must be taken into consideration as a limiting factor for the further tests. Considering also the results of our preliminary routing tests, we had to choose the 100 ms timeout for the routing tests to ensure the necessary performance for demonstrating multi-flow operation. It means that our test system is surely usable up to 1,100,000 qps and 900,000 qps, when it is used with IPv4 and IPv6 as transport protocol for the queries.

4.4. Routing tests

For the routing tests, we used the 100 ms timeout chosen before, not adding any value to compensate the latency of the IPv4 or IPv6 router, which was involved in both directions. The results of the routing tests are shown in Fig. 18. We note that the single flow results reflect the IPv4 and IPv6 routing performance of nodes, however, the multiple flow results were limited by the setting of the upper limit of the binary search (1,100,000 qps and 900,000 qps for IPv4 and for IPv6, respectively). The multiple flow results are presented to demonstrate that the test systems able to provide higher performance, when multiple flows are used. By this test, we have also checked that our test system can be used up to 1,100,000 queries per second with IPv4, which will be enough for the later tests.

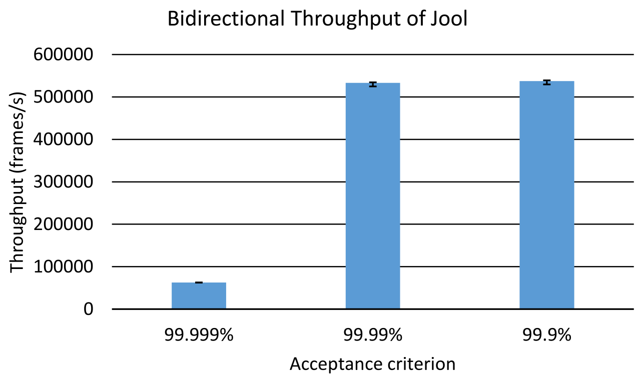

Fig. 19. Bidirectional throughput of Jool with different acceptance criteria using DNS traffic and 200 ms individual timeout.

The conclusion of these tests for our current effort is that both measurement systems can be used up to high enough rates for testing the different SIIT implementations.

The conclusion of these tests in general is that *benchmarking of a router is now possible with using individual timeout for every single frame*. If it can be done up to such rates using `dns64perf++`, which uses TCP/IP socket interface, then it can be done up to significantly higher rates using a DPDK-based Tester.

4.5. Frame loss rate consideration

During preliminary experiments, we have found that the test environment using `dns64perf++` produces a small amount of frame loss that has a significant impact on the results because the measurement procedure does not allow any loss of packets. To remedy this situation, we introduced an idea of *acceptable loss rate criterion* to achieve comparable performance results among the three SIIT implementations. The results of Jool with three different acceptance criteria (99.999%, 99.99%, and 99.9%) are shown in Fig. 19. (From now on, we use fps unit for easier comparison with the results in Section 3. Although we note that the comparability is only approximate due several reasons, e.g. different frame sizes, different timeout, different hardware instances,[6], etc.) Whereas the 99.999% acceptance criterion resulted in very low rate, the 99.99% acceptance criterion resulted in rates similar to that measured with the RFC 8219 compliant method, and 99.9% resulted in no further increase. We have performed the tests also with TAYGA and map646 and found that their throughput did not increase significantly using 99.9% as acceptance criterion compared to 99.99%. Therefore, we have chosen 99.99% as acceptance criterion for our further investigations. By doing so, we do not want to propose it as a general acceptance criterion, we have done so only to facilitate the performance comparison of the implementations.

4.6. Choice of timeout value

We expect that 50 ms should be more than enough timeout for a SIIT gateway. Calculating with 50 ms in each directions, we consider that $2 * 50$ ms + 100 ms = 200 ms timeout must be surely high enough for the SIIT tests. The results shown in Fig. 19 were received using 200 ms timeout value. That means the parameter combination of the timeout value of 200 ms and the acceptance criterion of 99.99% gives us comprehensive and comparable results of the three SIIT implementations. We used 200 ms timeout time for all further tests.

4.7. Performance comparison with bidirectional traffic using single and multiple flows

For performance comparison of the three SIIT implementations, we use their throughput results produced with 99.99% acceptance criterion, as shown in Fig. 20. On the left side of the figure, the "single flow" results are to be interpreted as "single flow per direction", the two directions mean two flows, of course. Although these results are not fully comparable with the single directional RFC 8219 compliant ones on Fig. 9 (for the reasons mentioned before), it is deliberate that Jool benefited from the fact that it could use two different cores for processing the packets of the two directions, whereas the others could not. As expected, the bidirectional throughput of Jool with 200 ms per frame timeout is visibly lower than the double of its unidirectional one shown in Fig. 9 with a global 2s timeout, similarly to the two other implementations, the throughput of which are not doubled.

The results of the multi-flow measurements are shown on the right side of Fig. 20. They were measured with the same settings as the single flow ones, the only difference was that we enabled the source and destination ports in the "rx-flow-hash" on the DUT. (To facilitate easy visual comparison, we plotted them together.) Due to using a high number of flows, the performance of Jool increased radically. Interestingly, both TAYGA and map646 benefited to some extent from the high number of flows. We presume that the performance increase was caused by the distribution of interrupts of the incoming packets to all the active CPU cores, because we have checked that the CPU utilization of TAYGA and map646 did not exceed 100%, the performance of a single CPU core.

From general point of view, our most important result is that we could benchmark the selected SIIT implementations with bidirectional traffic with both single and multiple flows using the `dns64perf++` tool.

[6] In [28] we have pointed out significant performance differences in the authoritative DNS performances of the test systems built up of different N nodes. (For example, the median, minimum and maximum values were 168,623 qps, 167,935 qps and 168,961 qps for one test system and 163,904 qps, 163,447 qps and 164,361 qps for another test system, when NSD was executed by a single CPU core. Please note that these intervals are non-overlapping.) This was the reason why we did not parallelize the testing of different DNS64 implementations using different nodes. Since then, we always benchmark all tested implementations by using the very same nodes for each of them.

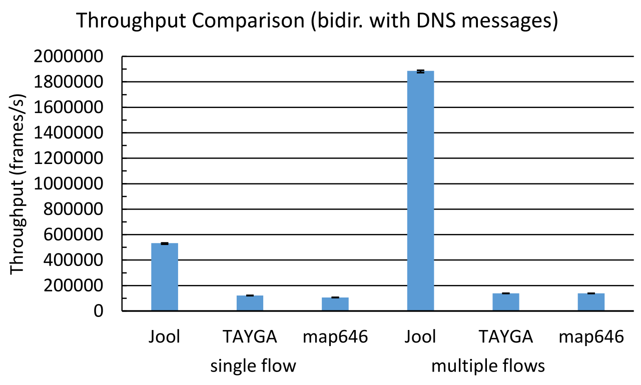

Fig. 20. Throughput comparison with bidirectional DNS traffic, using 200 ms timeout and 99.99% acceptance criterion.

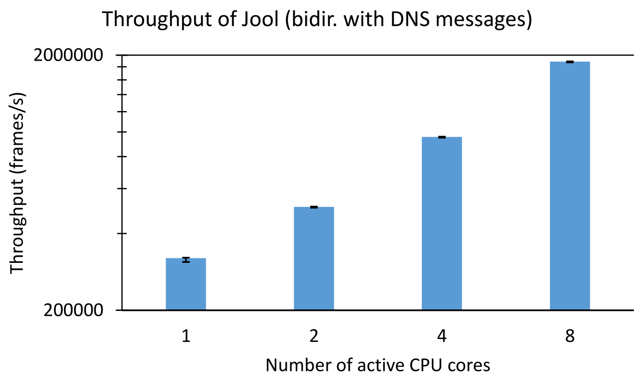

Fig. 21. The throughput of Jool as a function of the number of CPU cores using a high number of flows.

Table 3
The throughput of Jool as a function of the number of CPU cores using a high number of flows.

Num. CPU cores	1	2	4	8
Median (fps)	319980	508506	956322	1886325
Increase of median	–	1.59	1.88	1.97
1st percentile (fps)	309238	504744	949472	1872762
99th percentile (fps)	321592	509848	958336	1890758

4.8. The scale up of jool

We have examined, how the performance of Jool scales up with the number of active CPU cores. To keep the number of measurements relatively low, the number of CPU cores were always doubled compared to the previous case starting from 1. The results are shown in Fig. 21. (Beware, we used logarithmic scale!) To facilitate a more precise analysis, we have included the results also in Table 3, which we have completed with an additional row: "the increase of the median". It expresses the ratio of the current throughput and the throughput with half as many cores. It shows that the scale up from a single core to two cores is far from linear (the increase is only 59%), which we attribute to the cost of multi-core operation, but after that Jool scaled up very well (by an increase of 88% and then 97%). Unfortunately, the performance of TS3 was not enough to perform a measurement using all 16 cores of the DUT, but as far as we could test, Jool scaled up very well. Thus, we recommend its usage in high performance systems with multi-core computers as gateways.

5. Discussion and plans for future research

According to our understanding, the aim of benchmarking is to "accurately measure some standardized performance characteristics in order to obtain reasonable and comparable results" [38]. As for *network interconnect devices*, RFC 2544 compliant commercial testers have been serving this purpose for two decades. As for *IPv6 transition technologies*, RFC 8219 defined several benchmarking procedures, and we could test the viability of only a fraction of them, namely *throughput* and *frame loss rate*. RFC 8219 has taken both of them from RFC 2544 without changes. Being aware that they are matured and widely used, yet we believe that the time has come, when it is worth considering their update. We have pointed out several possibilities to improve them. Our recommendations for consideration are:

1. Checking the timeout individually for each frame.
2. The requirement of multiple repetitions of the tests.
3. An optional non-zero frame loss acceptance criterion.

As for the first one, we have shown that checking the timeout individually for every single frame is reasonable (and the results are more useful than those with a "global timeout", which can actually be anything form 2 s to 62 s concerning a given frame), and we have demonstrated its feasibility using a TCP/IP socket interface based tester. Our new DPDK-based tester, `siitperf` [14] complies with the requirements of RFC 8219, and it can optionally use per frame timeout with throughput and frame loss rate tests [15].

As for the second one, we have shown, that high deviations are possible both in throughput and in frame loss results, thus the repetition of the tests is a must. No new measurement procedure is necessary, only the requirements should be changed. We are aware that this change may drastically increase the time of testing. Therefore, we do not recommend always 20 repetitions, but rather a kind of adaptive method, which can stop after a few tests, if the results are consistent, but performs more tests if they are scattered. We plan to develop an algorithm, which can be easily implemented (e.g. in a shell script) and can determine on the fly, when testing may be finished. As for summarizing function, we recommend median, whereas 1st and 99th percentiles can be used to account the variation of the results.

As for the third one, we see contradicting arguments and interests. We are aware that the market of network interconnect devices has several stakeholders with different interests. For example, some high-end devices can operate at full line speed without frame loss, whereas others cannot [39]. As researchers, we have our own special interests. On the one hand, we need devices (e.g. switches, NICs) that comply with the absolute zero loss criterion, to be able to perform benchmarking tests. However, on the other hand, we have experienced that in some cases, the absolute zero loss criterion may prevent us from achieving practically usable results. (Our above experience with the SIIT implementations in Section 4 is only one example. We had the same experience in some cases in [28] and [36], and the non-zero loss acceptance criterion is used in the practice of benchmarking professionals for a long time [35].) Our recommendation is to keep the absolutely zero loss criterion as the compulsory test and make the higher than zero loss acceptance criterion test as a recognized optional throughput test. Standardizing its reporting format would make its results more transparent (by the compulsory indication of the allowed frame loss rate of the acceptance criterion).

As we have mentioned in Section 2.2, we have re-implemented the DPDK-based tester in C++ using a proper object oriented design. The handling of individual timestamps for each frame required by the newly recommended throughput and frame loss rate tests, is also used for the packet delay variation tests recommended by RFC 8219, thus we could make our coding work more economic with a proper design [15].

By using `siitperf`, we plan to compare the results of the traditional throughput and frame loss rate tests and the results of the newly recommended ones using per frame timeout.

We also plan to test the viability of the further measurements procedures recommended by RFC 8219 and supported by `siitperf`, namely: latency and packet delay variation.

Another interesting direction of research is to examine the issue of the scattered results of TAYGA and map646 with small frame size. Perhaps the analysis of the kernel level packet processing overhead may help us to find its root cause.

6. Reproducibility

We have made available the raw measurement results and the measurement scripts to support reproducibility [40]. The hardware parameters were given in Section 4.2. The setup and configuration of the tested SIIT implementations are disclosed in Appendix.

7. Conclusion

We have tested the viability of the benchmarking methodology for IPv6 transition technologies defined by RFC 8219 in the case of single translation technologies, and successfully demonstrated the feasibility of throughput and frame loss rate tests.

As for potential methodological problems in RFC 8219, we have pointed out that the "global timeout" defined originally by RFC 2544 is improper, and recommended the individual checking of the timeout for every single frame. We have also demonstrated its feasibility.

We have also pointed out the need for multiple tests, and recommended an adaptive algorithm for determination of the number of tests necessary, which is yet to be developed.

We have also shown that sometimes it is worth using a non-zero frame loss acceptance criterion, which we recommended to be a recognized optional test.

As for providing network operators with ready to use benchmarking results, we have performed the throughput and frame loss rate benchmarking tests required by RFC 8219 to analyze the performance of Jool, TAYGA and map646 and we have found that the performance of Jool scaled up well with the number of active CPU cores, and Jool also significantly outperformed TAYGA and map646.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Gábor Lencse: Conceptualization, Methodology, Software, Validation, Investigation, Writing - original draft, Writing - review & editing, Visualization. **Keiichi Shima:** Resources, Writing - review & editing, Supervision, Project administration, Funding acquisition.

Acknowledgments

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

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.

Appendix. Setup and configuration of the tested siit implementations

Tayga

Tayga is a part of the Debian Linux distribution, and its installation also prepares the necessary `nat64` pseudo network interface. We have made the following changes to its `/etc/tayga.conf` configuration file:

```
ipv4-addr 198.19.0.9
ipv6-addr 2001:2::9
map 198.18.0.1 2001:2::1
map 198.18.0.2 2001:2::2
map 198.19.0.1 2001:2:0:1::1
map 198.19.0.2 2001:2:0:1::2
```

As we wanted to use the same configuration for all implementations, the last line for p095 was as follows:

```
map 198.19.0.3 2001:2:0:1::3
```

In addition to that, we had to change two settings in the `/etc/default/tayga` file as follows:

```
RUN="yes"
CONFIGURE_NAT44="no"
```

After that, we could start it by the standard way under Debian Linux:

```
/etc/init.d/tayga start
```

Jool

Unfortunately, Jool was not a part of the Debian Linux distribution at the time of our measurements. Its compilation and installation was described in detail in its documentation [33], which we followed. Jool does not have a configuration file, its parameters were set by its user interface program, and the packets were redirected to Jool by `iptables` rules. We used the following commands:

```
/sbin/modprobe jool_siit
jool_siit instance add "benchmarking" --iptables
jool_siit -i "benchmarking" eamt \
 add 2001:2::/120 198.18.0.0/24
jool_siit -i "benchmarking" eamt \
 add 2001:2:0:1::/120 198.19.0.0/24
ip6tables -t mangle -A PREROUTING -s 2001:2::/120 \
 -d 2001:2:0:1::/120 -j JOOL_SIIT \
 --instance "benchmarking"
iptables -t mangle -A PREROUTING -s 198.19.0.0/24 \
 -d 198.18.0.0/24 -j JOOL_SIIT \
 --instance "benchmarking"
jool_siit -i "benchmarking" eamt display
```

Map646

Map646 was downloaded from [34]. It needed a minor update, because some changes were made to the library structure of the include files for JSON. (It means that the `json` library no more exists in `/usr/include`, but there are two different libraries for C and C++.)

We used the following settings in its `/etc/map646.conf` configuration file:

```
mapping-prefix 64:ff9b::
map-static 198.18.0.1 2001:2::1
map-static 198.18.0.2 2001:2::2
```

It also means that map646 could only be tested with unidirectional traffic in the IPv4 to IPv6 direction. It is so, because map646 was specifically designed for the WIDE cloud operation [5], and it does not support Explicit Address Mapping (defined by RFC 7757 [26]). More specifically, map646 cannot configure the (virtual) IPv6 addresses of the right hand side of the network topology as specified in Fig. 1. The (virtual) IPv6 addresses of the right hand side network must be a kind of IPv4-embedded IPv6 address (defined by RFC 6052 [41]) in a map646 operation. That is the reason, why we cannot perform bidirectional and reverse directional tests like with Jool and TAYGA.

Manual static ARP and ND settings

The static ARP and ND table entries were set manually in the DUTs. The settings of n018 were as follows:

```
ip neighbor add 2001:2::2 lladdr a0:36:9f:13:fe:28 \
 dev enp3s0f0 nud permanent
ip neigh add 198.19.0.2 lladdr a0:36:9f:13:fe:2a \
 dev enp3s0f1 nud permanent
ip neighbor add 2001:2:0:8000::2 lladdr \
 a0:36:9f:13:fe:2a dev enp3s0f1 nud permanent
```

The settings of p095 were as follows:

```
ip neighbor add 2001:2::2 lladdr a0:36:9f:c5:fa:1c \
 dev enp5s0f0 nud permanent
ip neigh add 198.19.0.3 lladdr a0:36:9f:c5:fa:1e \
 dev enp5s0f1 nud permanent
ip neighbor add 2001:2:0:8000::2 lladdr \
 a0:36:9f:c5:fa:1e dev enp5s0f1 nud permanent
```

References

[1] C. Bao, X. Li, F. Baker, T. Anderson, F. Gont, IP/ICMP Translation Algorithm, IETF RFC 7915, 2016, http://dx.doi.org/10.17487/RFC7915.

[2] M. Bagnulo, P. Matthews, I. Beijnum, Stateful NAT64: Network Address and Protocol Translation from IPv6 clients to IPv4 servers, IETF RFC 6146, 2011, http://dx.doi.org/10.17487/RFC6146.

[3] M. Bagnulo, A. Sullivan, P. Matthews, I. Beijnum, DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers, IETF RFC 6147, 2011, http://dx.doi.org/10.17487/RFC6147.

[4] M. Mawatari, M. Kawashima, C. Byrne, 464XLAT: Combination of Stateful and Stateless Translation, IETF RFC 6877, 2013, http://dx.doi.org/10.17487/RFC6877.

[5] K. Shima, W. Ishida, Y. Sekiya, Designing an IPv6-oriented datacenter with IPv4-IPv6 translation technology for future datacenter operation, in: Cloud Computing and Services Science (CLOSER 2012), Porto, Portugal, 2012, pp. 39–53, http://dx.doi.org/10.1007/978-3-319-04519-1_3.

[6] Free Software Foundation, The free software definition, online URL http://www.gnu.org/philosophy/free-sw.en.html.

[7] M. Georgescu, L. Pislaru, G. Lencse, Benchmarking Methodology for IPv6 Transition Technologies, IETF RFC 8219, 2017, http://dx.doi.org/10.17487/RFC8219.

[8] G. Lencse, Y. Kadobayashi, Comprehensive survey of IPv6 transition technologies: A subjective classification for security analysis, IEICE Trans. Commun. E102-B (10) (2019) 2021–2035, http://dx.doi.org/10.1587/transcom.2018EBR0002.

[9] S. Bradner, J. McQuaid, Benchmarking Methodology for Network Interconnect Devices, IETF RFC 2544, 1999, http://dx.doi.org/10.17487/RFC2544.

[10] G. Lencse, M. Georgescu, Y. Kadobayashi, Benchmarking methodology for DNS64 servers, Comput. Commun. 109 (1) (2017) 162–175, http://dx.doi.org/10.1016/j.comcom.2017.06.004.

[11] V. Paxson, G. Almes, J. Mahdavi, M. Mathis, Framework for IP Performance Metrics, IETF RFC 2330, 1998, http://dx.doi.org/10.17487/RFC2330.

[12] P. Bálint, Test software design and implementation for benchmarking of stateless IPv4/IPv6 translation implementations, in: Proc. 40th International Conference on Telecommunications and Signal Processing (TSP 2017), Barcelona, Spain, 2017, pp. 74–78, http://dx.doi.org/10.1109/TSP.2017.8075940.

[13] D. Scholz, A look at Intel's dataplane development kit, in: Proc. Seminars Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM), Munich, Germany, 2014, pp. 115–122, http://dx.doi.org/10.2313/NET-2014-08-1_15.

[14] G. Lencse, Siitperf: an RFC 8219 compliant SIIT (stateless NAT64) tester, source code. URL https://github.com/lencsegabor/siitperf.

[15] G. Lencse, Design and implementation of a software tester for benchmarking stateless NAT64 gateways, under review in IEICE Transactions on Communications. URL http://www.hit.bme.hu/~lencse/publications/IEICE-2019-siitperf-for-review.pdf.
[16] C. Popoviciu, A. Hamza, G.V. de Velde, D. Dugatkin, IPv6 Benchmarking Methodology for Network Interconnect Devices, IETF RFC 5180, 2008, http://dx.doi.org/10.17487/RFC5180.
[17] G. Lencse, Benchmarking Stateless NAT64 Implementations with a Standard Tester, unpublished, will be available from: http://www.hit.bme.hu/~lencse/publications/.
[18] K.J.O. Llanto, W.E.S. Yu, Performance of NAT64 versus NAT44 in the context of IPv6 migration, in: Proc. International Multiconference of Engineers and Computer Scientists 2012 (IMECS 2012), Hong Kong, Hongkong, 2012, pp. 638–645, URL http://www.iaeng.org/publication/IMECS2012/IMECS2012_pp638-645.pdf.
[19] 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, J. Comput. Sci. Technol. 12 (2) (2012) 64–70, URL http://sedici.unlp.edu.ar/handle/10915/19702.
[20] S. Yu, B.E. Carpenter, Measuring IPv4-IPv6 translation techniques, 2012, Computer Science Technical Reports (2012-001), Dept. of Computer Science, Univ. of Auckland, Auckland, New Zeeland. URL http://hdl.handle.net/2292/13586.
[21] G. Lencse, S. Répás, Performance analysis and comparison of different DNS64 implementations for Linux, OpenBSD and FreeBSD, in: Proc. IEEE 27th International Conference on Advanced Information Networking and Applications (AINA 2013), Barcelona, Catalonia, Spain, 2013, pp. 877–884, http://dx.doi.org/10.1109/AINA.2013.80.
[22] G. Lencse, G. Takács, Performance analysis of DNS64 and NAT64 solutions, Inf. J. 4 (2) (2012) 29–36, URL http://www.infocommunications.hu/documents/169298/404123/InfocomJ_2012_2_Lencse.pdf.
[23] G. Lencse, S. Répás, Performance analysis and comparison of the TAYGA and of the PF NAT64 implementations, in: Proc. 36th International Conference on Telecommunications and Signal Processing (TSP 2013), Rome, Italy, 2013, pp. 71–76, http://dx.doi.org/10.1109/TSP.2013.6613894.
[24] S. Répás, P. Farnadi, G. Lencse, Performance and stability analysis of free NAT64 implementations with different protocols, Acta Tech. Jaurinensis 7 (4) (2014) 402–427, http://dx.doi.org/10.14513/actatechjaur.v7.n4.340.
[25] A. Quintero, F. Sans, E. Gamess, Performance evaluation of IPv4/IPv6 transition mechanisms, Int. J. Comput. Netw. Inf. Secur. 2016 (2) (2012) 1–14, http://dx.doi.org/10.5815/ijcnis.2016.02.01.
[26] T. Anderson, A.L. Potter, Explicit Address Mappings for Stateless IP/ICMP Translation, IETF RFC 7757, 2016, http://dx.doi.org/10.17487/RFC7757.
[27] G. Lencse, D. Bakai, Design and implementation of a test program for benchmarking DNS64 servers, IEICE Trans. Commun. E100-B (6) (2017) 948–954, http://dx.doi.org/10.1587/transcom.2016EBN0007.
[28] G. Lencse, Y. Kadobayashi, Benchmarking DNS64 implementations: Theory and practice, Comput. Commun. 127 (1) (2018) 61–74, http://dx.doi.org/10.1016/j.comcom.2018.05.005.
[29] G. Lencse, S. Répás, Performance analysis and comparison of four DNS64 implementations under different free operating systems, Telecommun. Syst. 63 (4) (2016) 557–577, http://dx.doi.org/10.1007/s11235-016-0142-x.
[30] Cisco, End user license agreement, online https://www.cisco.com/c/en/us/about/legal/cloud-and-software/end_user_license_agreement.html.
[31] Juniper, End user license agreement, online https://support.juniper.net/support/eula/.
[32] N. Lutchansky, TAYGA: Simple, no-fuss NAT64 for Linux, 2011, online http://www.litech.org/tayga/.
[33] NIC Mexico, Jool: SIIT and NAT64, 2019, online http://www.jool.mx/en/about.html.
[34] K. Shima, A one to one address mapping program for translation from IPv4 packets to IPv6 packets and vice versa, 2010, source code https://github.com/keiichishima/map646.
[35] K. Tolly, The real meaning of zero-loss testing, 2001, IT World Canada https://www.itworldcanada.com/article/kevin-tolly-the-real-meaning-of-zero-loss-testing/33066.
[36] G. Lencse, Benchmarking authoritative DNS servers: Theory and practice, unpublished, will be available from: http://www.hit.bme.hu/~lencse/publications/.
[37] M. Majkowsky, How to receive a million packets per second, 2015, Cloudflare Blog https://blog.cloudflare.com/how-to-receive-a-million-packets/.
[38] G. Lencse, Benchmarking methodology for IPv6 transition technologies, 2017, IIJ Lab seminar https://seminar-materials.iijlab.net/iijlab-seminar/iijlab-seminar-20171010.pdf.
[39] Tolly Enterprises, Mellanox Spectrum vs. Broadcom StrataXGS Tomahawk 25GbE and 100GbE performance evaluation, Tolly Test Report #216112, 2016, URL http://www.mellanox.com/related-docs/tolly/tolly-report-performance-evaluation-2016-march.pdf.
[40] G. Lencse, Results and measurement scripts for paper Performance Analysis of SIIT Implementations: Testing and Improving the Methodology, online http://www.hit.bme.hu/~lencse/publications/ECC-2020-SIIT-perf-meas/.
[41] C. Bao, C. Huitema, M. Bagnulo, M. Boucadair, X. Li, IPv6 addressing of IPv4/IPv6 translators, IETF RFC 6052, 2010, http://dx.doi.org/10.17487/RFC6052.

Gábor Lencse received M.Sc. and Ph.D. 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 a Professor. He is also a part time Senior Research Fellow at the Department of Networked Systems and Services, Budapest University of Technology and Economics, Budapest, Hungary since 2005.

His research interests include the performance analysis of communication systems with current focus on IPv6 transition technologies.

He was a guest researcher at IIJ Innovation Institute, Tokyo, Japan from April 2 to July 2, 2019, where his research topic was the performance analysis of SIIT implementations.

Keiichi Shima is a deputy director at the Research Laboratory of IIJ Innovation Institute, Inc. His research field is the Internet, including designing and implementing communication protocols, computer 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.