

# Performance Analysis of SIIT Implementations: Testing and Improving the Methodology

G. Lencse<sup>1</sup> · K. Shima<sup>2</sup>

## 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.

**Keywords** Benchmarking · IPv6 deployment · IPv6 transition solutions · SIIT · Stateless NAT64 · Performance analysis

## 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 a few SIIT implementations and thus provide network operators with ready to use benchmarking results.

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 reports the design and implementations of such tester [9]. However, it has never been publicly released due to its insufficient performance. Rather, it was re-implemented using DPDK (Intel Data Plane Development Kit [10]) 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. Unfortunately, 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, but we plan to reimplement it from scratch in C++ using a proper object oriented design and plan to make it publicly available from GitHub.

Right before our current effort, we have examined the possibility of benchmarking stateless NAT64 implementations using legacy RFC 2544 [11] / RFC 5180 [12] compliant Testers. We have reported our results in [13].

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. In section 3, first, we design the test and traffic setup, next, we

<sup>1</sup> Department of Networked Systems and Services, Budapest University of Technology and Economics, 2 Magyar Tudósok körútja, H-1117 Budapest, Hungary, E-mail: lencse@hit.bme.hu

<sup>2</sup> IJ Innovation Institute Inc., Iidabashi Grand Bloom, 2-10-2 Fujimi, Chiyoda-ku, Tokyo 102-0071, Japan, E-mail: keiichi@ijjlab.net

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 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 concludes our paper.

## 2 Benchmarking Tests for SIIT Gateways

RFC 8219 addresses single translation technologies in general, but now we focus on SIIT. 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 recommended test setup is very simple: it consists of the Tester and the DUT (Device Under Test), see Fig. 1. Although the arrows are unidirectional, the traffic may be bidirectional. RFC 8219 requires testing with bidirectional traffic and makes testing with unidirectional traffic optional.

The recommended measurement procedures usually came from RFC 2455. The following ones are kept unchanged: *throughput*, *frame loss rate*, *back-to-back frames*, and *system recovery*. The measurement procedure for *latency* has been redefined to achieve higher accuracy, and further measurement procedures for *packet delay variation*, *inter packet delay variation*, and *reset* have been added. For more details, please refer to Section 7 of RFC 8219.

It also means that the strict absolutely zero packet 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 seconds long test. (In practice, the highest such rate is determined by a binary search, where the initial upper limit of the interval is the maximum frame rate of the media.)

As for the recommended frames sizes, they have been mainly kept, but somewhat amended. They 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%,

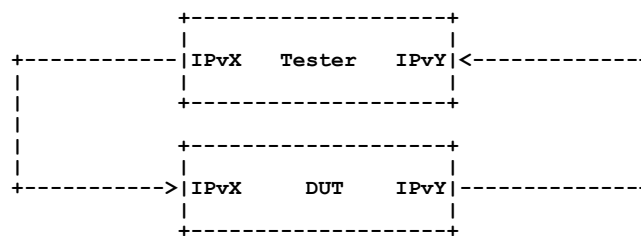


Fig. 1 Single DUT test setup [7]

90%, 50% and 10% are recommended for the proportion of the translated traffic, where the rest should be native IPv6 traffic.

As for timeout time, RFC 8219 mentions it only concerning DNS64 testing, but it *makes no explicit recommendation for an appropriate timeout time* in the case of any other types of measurements. According to our interpretation, timeout is defined implicitly as follows. As we have already mentioned, several measurement procedures defined in RFC 2544 were kept unchanged. Section 23 of RFC 2544 contains a *general trial description*. It says that 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 seriously 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 *number of repetitions of the tests*. RFC 8219 mentions at four different places that the tests must be repeated at least 20 times. They 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 [14] 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 evaluation of the test data must be done with an understanding of generally

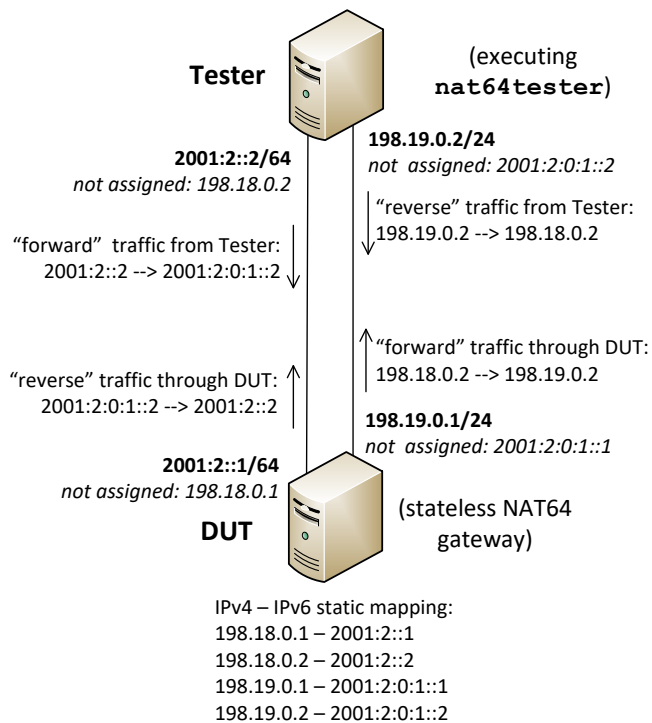


Fig. 2 Traffic for benchmarking stateless NAT64 gateways

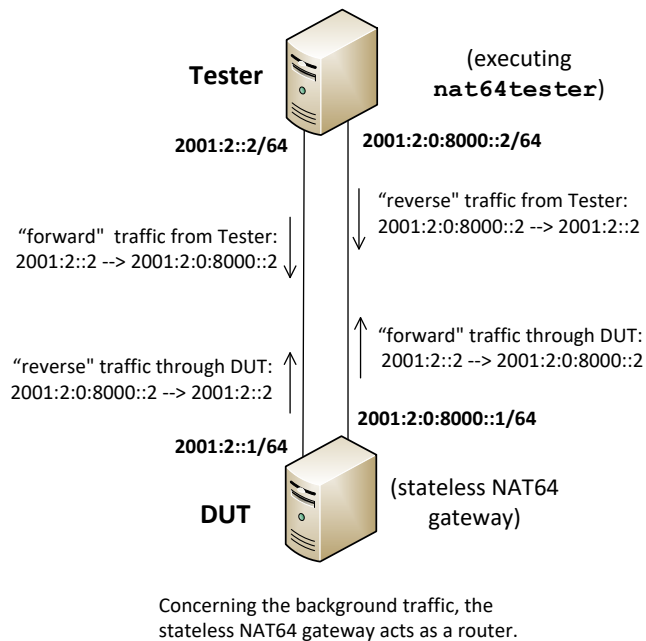


Fig. 3 Background traffic for benchmarking stateless NAT64 gateways

accepted testing practices regarding repeatability, variance and statistical significance of small numbers of trials.” As for repeatability, RFC 2330 [15] 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.

### 3 Benchmarking Measurements with a Stateless NAT64 Tester

#### 3.1 Test and Traffic Setup

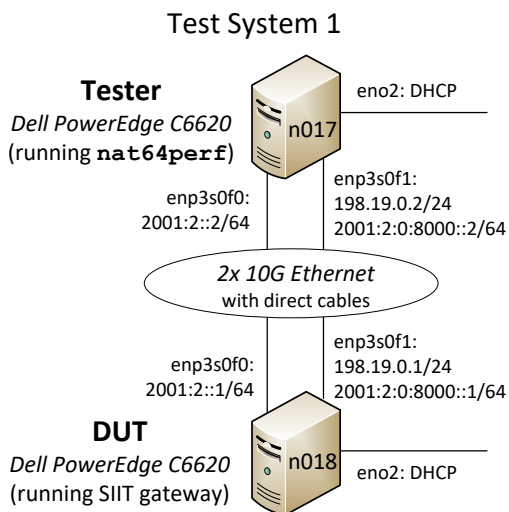
Following the requirements of RFC 8219, we have designed the test and traffic setup for benchmarking stateless SIIT gateways using EAM (Explicit Address Mapping) [16]. The traffic to be translated is shown in Fig. 2. Only IPv6 addresses are assigned to the left side network interfaces of both the Tester and the DUT, and similarly, their network interfaces on their right side have only IPv4 addresses. 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. The native IPv6 traffic, which we call as “background” traffic is shown in Fig. 3. Concerning this traffic, the DUT acts as a router.

#### 3.2 NAT64 Tester in a Nutshell

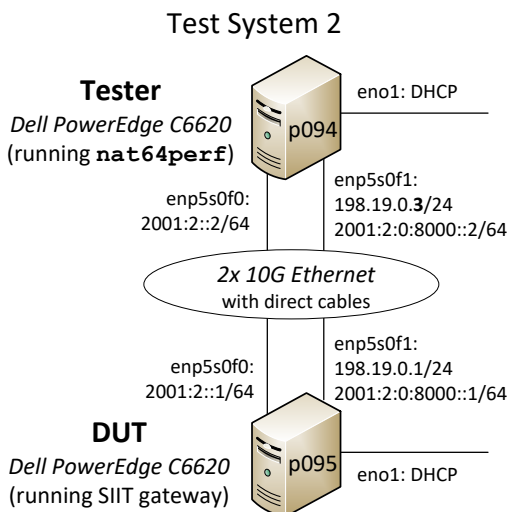
We give a very brief summary of the functional design of the DPDK based NAT64 tester (called **nat64tester**). Following the high level design of Dániel Bakai’s excellent DNS64 tester called **dns64perf++** [17], **nat64tester** performs only one test, and the binary search, as well as the further repetitions 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 *packet loss*. 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



**Fig. 4** Test System 1 built up by N nodes, with a fixed 2GHz CPU clock frequency DUT



**Fig. 5** Test System 2 built up by P nodes, with a variable (1.2-3GHz) CPU clock frequency DUT

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, thus it requires direct cable connections between the Tester and the DUT, and static ARP/ND table entries has to be set manually.

### 3.3 Measurement Environment

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

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

**Table 1** The Building Elements of the Test Systems for Basic Tests

Test System	Tester node	Tester speed	DUT node	DUT speed	DUT active cores
TS1	n017	2-2.8GHz	n018	2GHz	0-7
TS2	p094	1.2-3GHz	p095	1.2-3GHz	0-31

port X520 network adapters.

- The P nodes are Dell PowerEdge R430 servers with two 2.1GHz Intel Xeon E5-2683 v4 CPUs having 16 cores each, 384GB 2400MHz 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. 4), we switched off hyper-threading in both computers and set the clock frequency of the DUT to 2GHz (fixed), because we knew from our previous benchmarking experience [18] 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 (see Fig. 4) 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 2.8GHz, 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. 5) 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 1.2GHz to 3GHz, 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 Fig. 4 and Fig. 5. 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 reason we presented in [19]. We have made a survey of papers on NAT64 performance measurements and existing free software stateless NAT64 implementations in [13]. 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) [20], Debian package version: 0.9.2-6+b1
- Jool 4.0.1 (released on April 26, 2019) [21]
- map646 (GitHub latest commit cd93431 on Mar 31, 2016) [22]

We note that in our previous paper [13], 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, how we set the different SIIT implementations.

#### 3.4.1 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

```

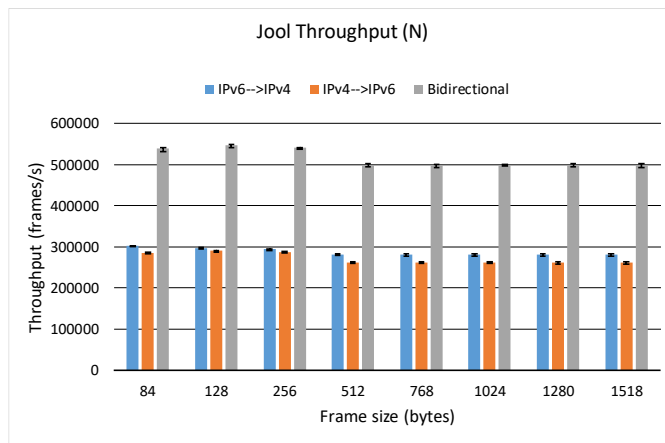


Fig. 6 Throughput results of Jool, TS1

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

```

#### 3.4.2 Jool

Unfortunately, Jool is not yet a part of the Debian Linux distribution. Its compilation and installation is described in detail in its documentation [21], 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
iptables -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

```

#### 3.4.3 Map646

Map646 was downloaded from [22]. 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. (We have disclosed its root causes in [13].)

### 3.4.4 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
```

## 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.

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. 6. 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 [13], 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 (from IPv6 to IPv4), reverse (from IPv4 to IPv6) and bidirectional traffic are 296,972fps (frames per second) 290,234fps, and 547,848fps, respectively. The observation that the bidirectional throughput is 5.62% less

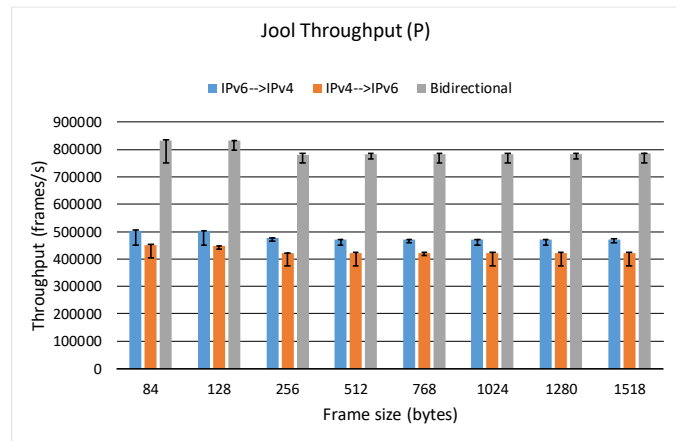


Fig. 7 Throughput results of Jool, TS2

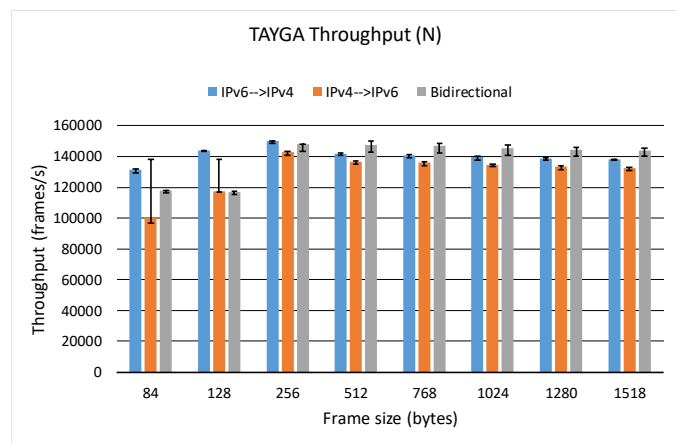


Fig. 8 Throughput results of TAYGA, TS1

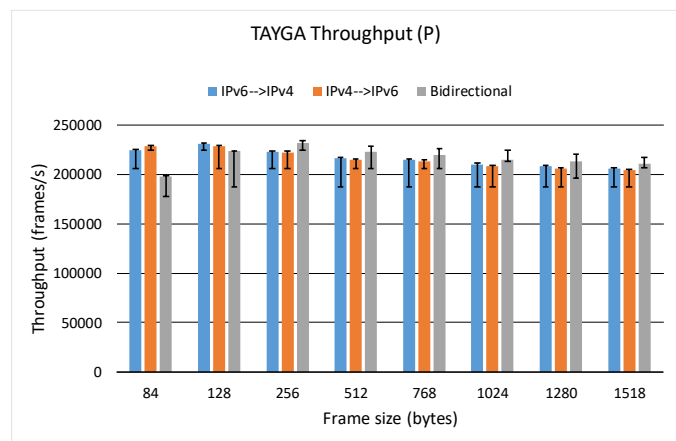


Fig. 9 Throughput results of TAYGA, TS2

than the double of the minimum of the unidirectional throughput (580,468fps) 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. 7. 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<sup>3</sup>, 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 frequency.

### 3.5.2 TAYGA

The throughput results of TAYGA produced by TS1 are shown in Fig. 8. There are visible problems at 84bytes and 128bytes 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 96,777fps and the 99th percentile is 138,183fps at 84bytes 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 256bytes to 1518bytes frame sizes are nearly constant, showing very small degradation with the increase of the frame size. (Considering that TAYGA works in user-space, we easily could accept even higher degradation than that.)

The throughput results of TAYGA produced by TS2 are shown in Fig. 9. 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 84bytes and 128bytes frame sizes is hardly noticeable. (Only the bidirectional throughput shows some deviation: the median value at 84bytes is visibly less, than it should be and the error bar at 128bytes is higher than at any other places). We consider this phenomenon to be a good example, 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. 10. Similarly to TAYGA, there are visible problems at 84bytes and 128bytes frame sizes: the throughput

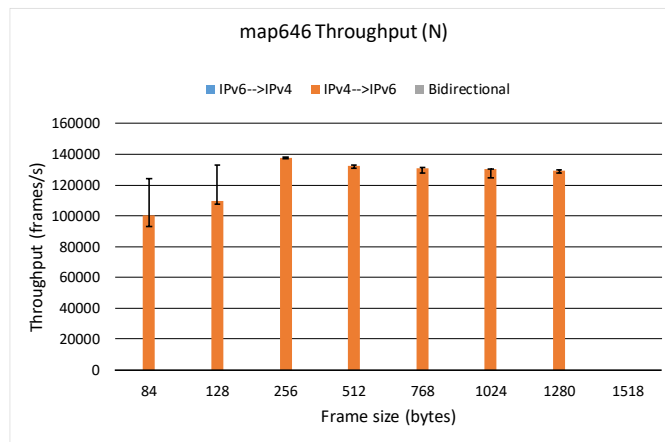


Fig. 10 Throughput results of map646, TS1

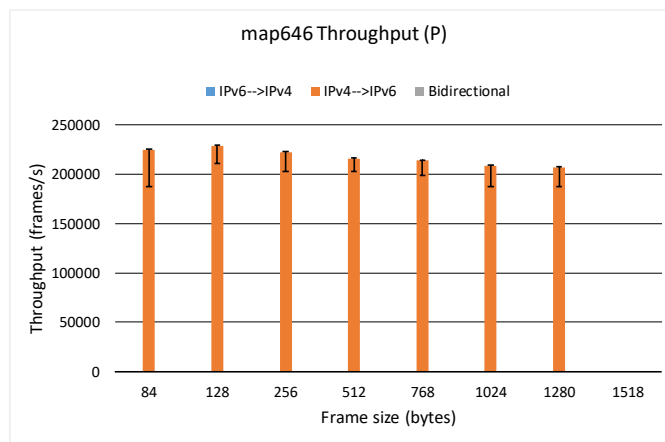


Fig. 11 Throughput results of map646, TS2

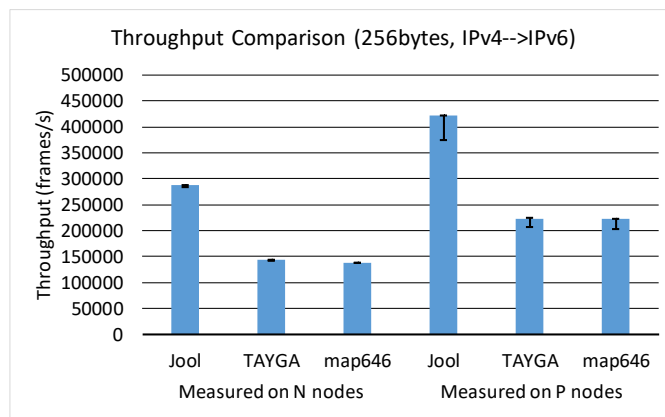


Fig. 12 Throughput comparison with 256bytes frames, IPv4 to IPv6 direction

is visibly lower than at 256bytes, and the error bars are very high, indicating significantly scattered measurement results.

The throughput result at 1518bytes is missing due to the same reason discussed in [13].

Otherwise, the results from 256bytes to 1280bytes 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

<sup>3</sup> In fact, we could observe only the load caused by software interrupts, as Jool works in the Linux kernel.

shown in Fig. 11. Unlike with TS1, here the median throughput at 84bytes and 128bytes frame sizes is very similar to the throughput at other frame sizes, only the somewhat higher error bar at 84bytes reminds us the problem observed on TS1.

### 3.4.5 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 256bytes 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. 12. 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 256bytes 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. Therefore, we have chosen two frame sizes, 128bytes and 1280bytes 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 RFC2544, 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 128bytes frame size is 8,445,945fps, 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 50,000fps to 500,000fps using 50,000fps steps.

#### 3.6.1 Jool

The frame loss rate results of Jool measured on TS1 are shown in Fig. 13. In agreement with Fig. 6, no frame loss occurred from 50kfps to 250kfps. From 300kfps rate, an increasing frame loss rate can be observed. For any given frame rate, the frame loss rate with 1280bytes frames were higher than with 128byte frames, which also complies with the results shown in Fig. 6: 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 300kfps, the number of frames

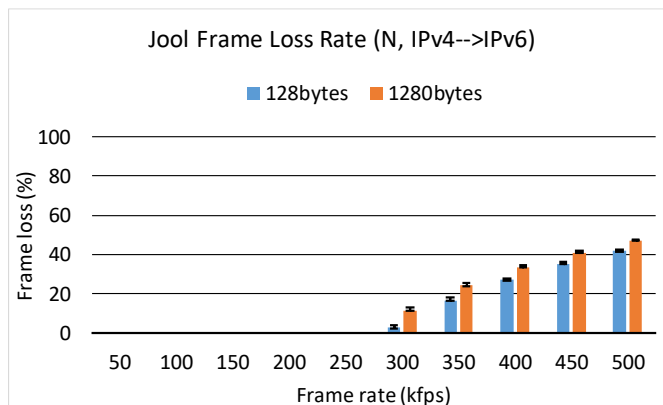


Fig. 13 Frame loss rate results of Jool, TS1

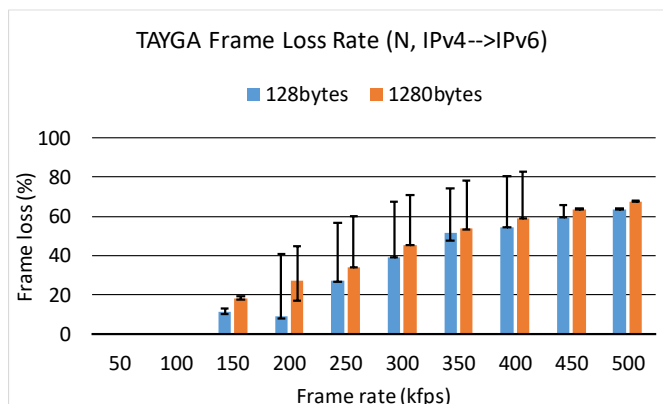


Fig. 14 Frame loss rate results of TAYGA, TS1

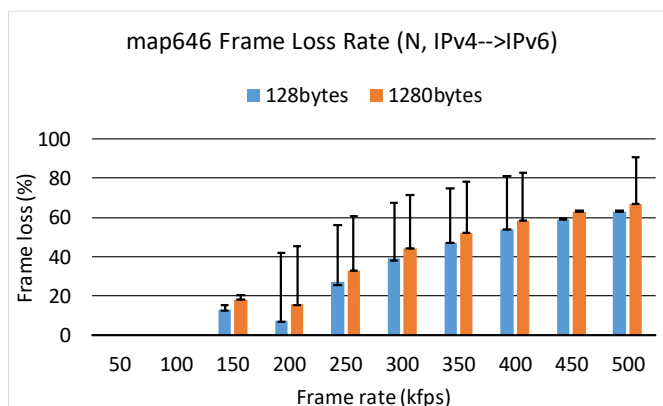


Fig. 15 Frame loss rate results of map646, TS1

transmitted during the 60s long tests were approximately constant, about 17.5 million and 15.9 million with 128bytes and 1280bytes 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. 14. Whereas the median values roughly comply with what could be expected on the basis of the previous results



(frame loss appears from 150,000fps, 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 200kfps to 400kfps 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 128bytes shown in Fig. 8. However, the throughput results on the same figure are very stable at 1280bytes. Yet the frame loss rate (and thus also the throughput) shows high fluctuations under serious overload conditions with 1280bytes long frames.

### 3.6.3 Map646

The frame loss rate results of map646 measured on TS1 are shown in Fig. 15. They are very similar to that of TAYGA with an exception that here a high error bar appears also at 500kfps rate with 1280bytes.

## 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. 16. 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
- $\alpha$ : 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 1280bytes long frames is somewhat lower than with 128bytes frames.

### 3.7.2 TAYGA

The throughput results of TAYGA using mixed traffic measured on TS1 are shown in Fig. 17. Here only the throughput of the 1280bytes 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 650kfps, and 7 of them were below

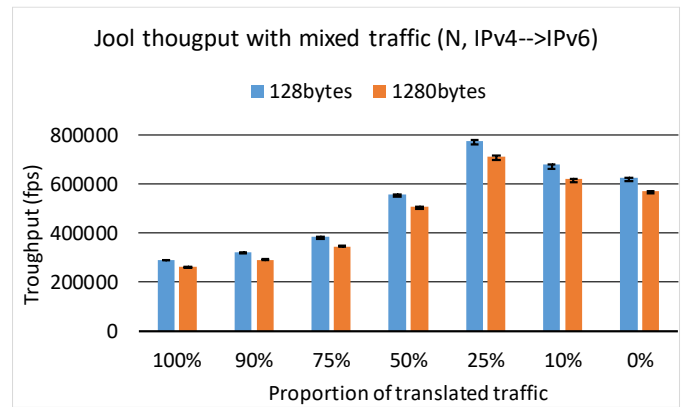


Fig. 16 Throughput results of Jool with mixed traffic (the rest is native IPv6), TS1

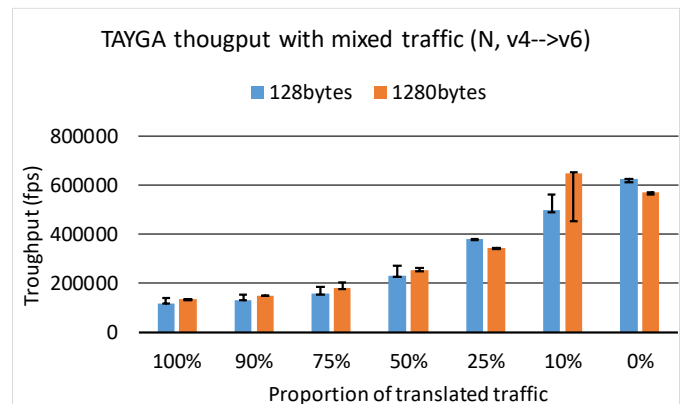


Fig. 17 Throughput results of TAYGA with mixed traffic (the rest is native IPv6), TS1

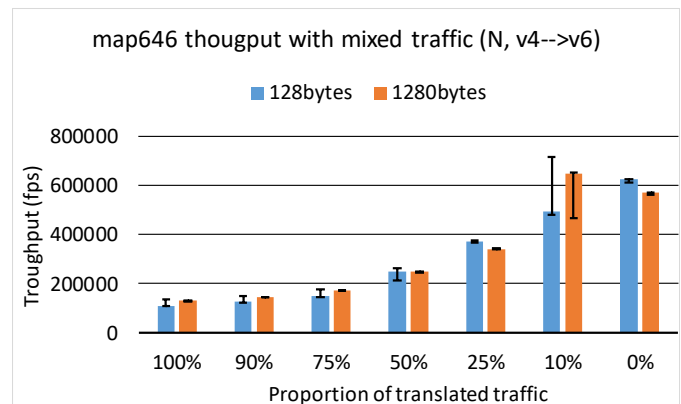
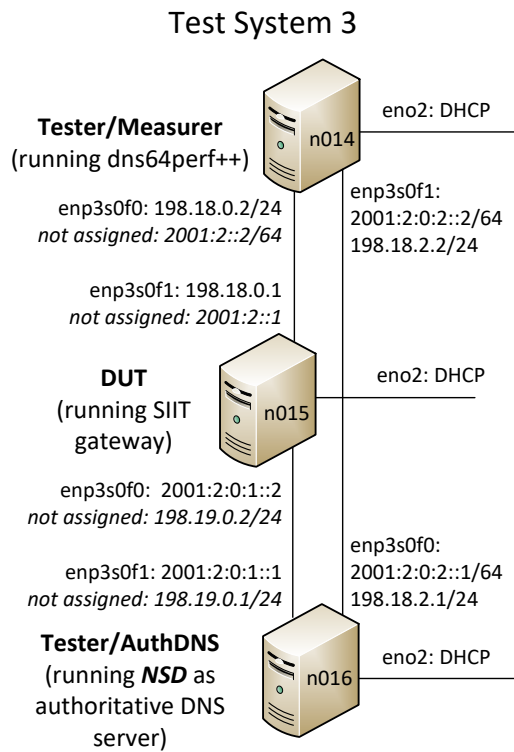


Fig. 18 Throughput results of map646 with mixed traffic (the rest is native IPv6), TS1

460kfps.

### 3.7.3 Map646

The throughput results of map646 using mixed traffic measured on TS1 are shown in Fig. 18. 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.



**Fig. 19** Test System 3 built up by N nodes, with a fixed 2GHz CPU clock frequency DUT

### 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.

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

### 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 packet 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 [23]. However, on the other hand, we contend that the RFC 2544 benchmarking procedure *is way too lax*. Frames are sent for 60 seconds and received for 62 seconds. We understand, that the additional 2 seconds timeout was probably set to surely receive also the very last frames. However, it also means that the very first frame has theoretically 62 seconds

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 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++

In this section, we do 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 [17]. 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. 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 3.3Mqps (3.3 million queries per second) rate for benchmarking authoritative DNS servers [24]. It has two significant advantages over **nat64tester**:

- 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

**Table 2** The Building Elements of Test System 3

Test System	Tester/Measurer			Tester/AuthDNS			DUT (Device Under Test)			Connection of the elements
	node	speed	active cores	node	speed	active cores	node	speed	active cores	
TS3	n014	2-2.4GHz	0-15	n016	2GHz	0-15	n015	2GHz	0-7	Direct cables

per second [25] 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 [14], 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. 19. 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 previous results [24], 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 2GHz, 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 2.4GHz 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. 19. 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* [14]. 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. 19 (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 interface rx-flow-hash udp4 sdfn
ethtool -N interface rx-flow-hash udp6 sdfn
```

(The interfaces were `enp3s0f0` and `enp3s0f1`.)

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, and 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.

## 4.3 Self-test Measurements

Similarly to the self-test measurements of the DNS64 benchmarking tests [14], 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 [24] was that 100ms was likely workable from the authoritative DNS server point of view.

We have performed self-test measurement using 200ms, 100ms, 50ms, and 25ms timeout values and both IPv4 and IPv6 as transport protocol for carrying the DNS messages. The results are shown in Fig 20. (The upper limit of the binary search was set to 1,600,000qps and 1,000,000qps 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 100ms 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 1,100,000qps and 900,000qps, 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 100ms 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. 21. 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 (1,100,000qps and 900,000qps 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.

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 Choice of Timeout Value and Parameter Study

We expect that 50ms should be more than enough timeout for a SIIT gateway. Calculating with 50ms in each directions, we consider that  $2 \cdot 50\text{ms} + 100\text{ms} = 200\text{ms}$  timeout must be surely high enough for the SIIT tests.

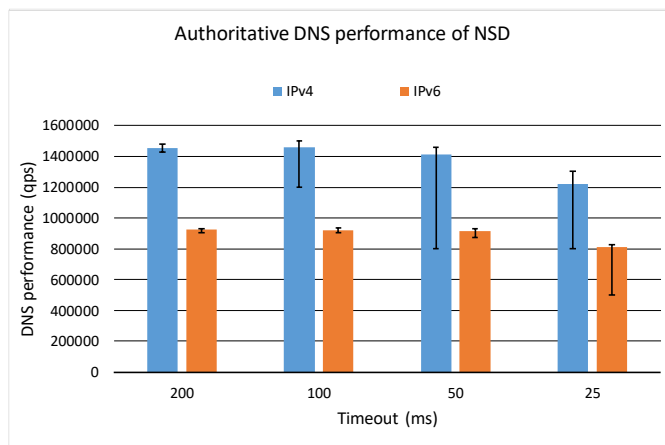


Fig. 20 Self-test results as a function of timeout time

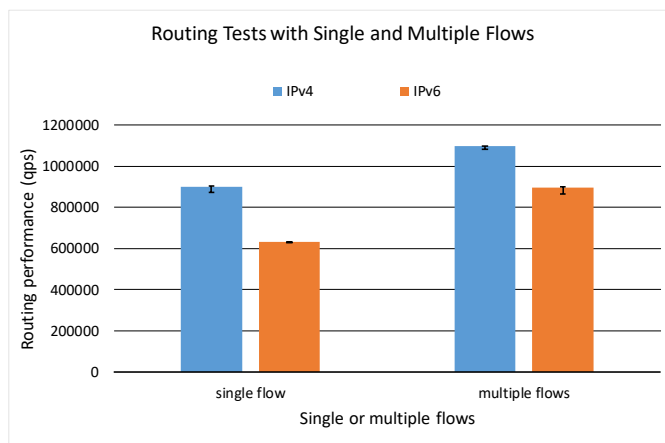


Fig. 21 Results of the routing tests (beware that the unit is qps and not fps)

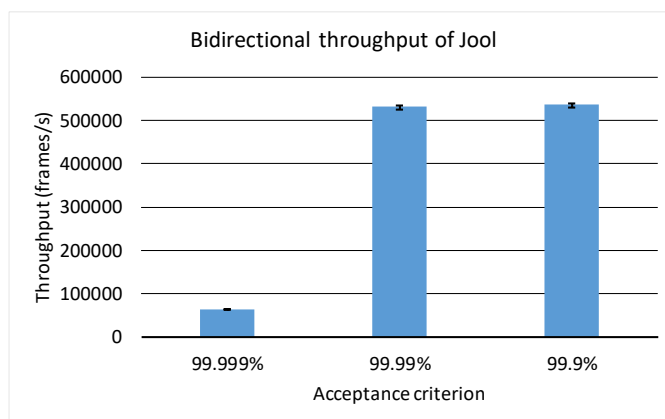


Fig. 22 Bidirectional throughput of Jool with different acceptance criteria using DNS traffic and 200ms individual timeout

First, we tested Jool, and received an unexpectedly low result, namely 3,177qps, which is 6354fps. We observed that all 20 measurements produced the very same results and several binary search steps failed due to a very low rate loss (less than 0.01%). We have performed the test also with 400ms timeout, but the results did not change. We have received similarly low results with the two other SIIT implementations. We

considered that they were *completely unsuitable* for the comparison of performance of the three SIIT implementations.

Then we used a different method, which allowed some low lost rate as *acceptance criterion*. It means if at least 99.999% of the queries were replied within timeout time, then the test was considered as *successful*, otherwise it was considered as *failed*. Also 99.99% and 99.9% values were used as acceptance criterion. (The previously chosen 200ms timeout was used for these and all further tests.) The results of Jool are shown in Fig. 22. (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<sup>4</sup>, 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 Performance Comparison with Bidirectional Traffic using Single and Multiple Flows

We use the 99.99% acceptance criterion results of the three SIIT implementations for comparison, as shown in Fig. 23. 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. 12 (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, its bidirectional result with 200ms per frame timeout are visibly lower than the double of its unidirectional one shown in Fig.12 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. 23. 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

<sup>4</sup> In [18], 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 168,623qps, 167,935qps and 168,961qps for one test system and 163,904qps, 163,447qps and 164,361qps 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.

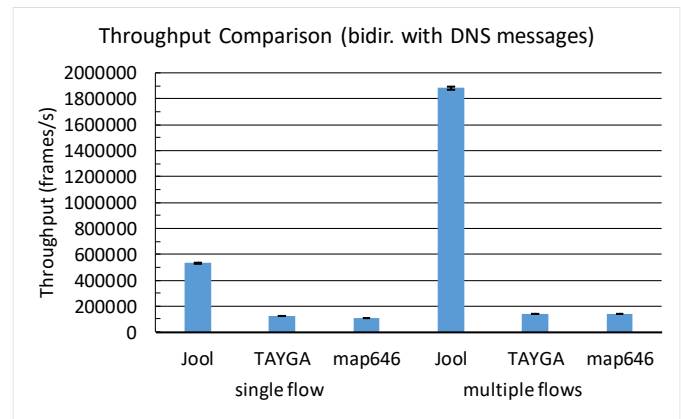


Fig. 23 Throughput comparison with bidirectional DNS traffic, using 200ms timeout and 99.99% acceptance criterion

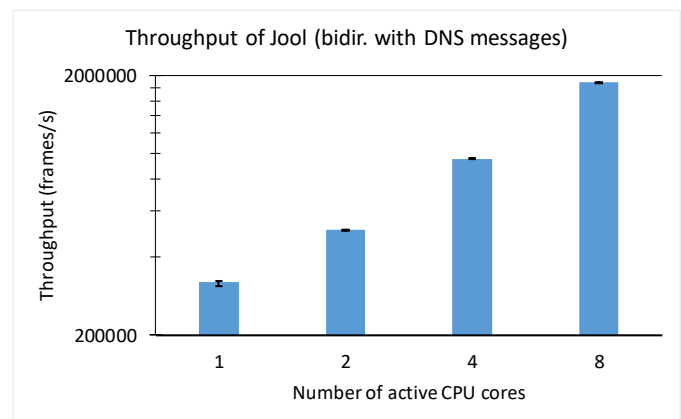


Fig. 24 The throughput of Jool as a function of the number of CPU cores using a high number of flows

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

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.

#### 4.7 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. 24. (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” [26]. 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 from 2 seconds to 62 seconds concerning a given frame), and we have demonstrated its feasibility using a TCP/IP socket interface based tester. We are intending to develop a DPDK-based tester, which complies with the requirements of RFC 8219, and can optionally use individual timestamps.

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 [27]. 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 [18] and [24], and the non-zero loss acceptance criterion is used in the practice of benchmarking professionals for a long time [23].) 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 the introduction, we are planning to re-implement the DPDK based tester in C++ using a proper object oriented design, which will also facilitate its easy extension with new functions. The handling of individual timestamps for each frame required by the newly recommended throughput and frame loss rate tests, may also be used for the packet delay variation and inter packet delay variation tests recommended by RFC 8219, thus a proper design may make the coding work more economic.

We plan to retest the three examined SIIT implementations with the new tester to be able to compare the results of the newly recommended tests with the results of the original ones.

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 Conclusion

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 found that the performance of Jool scaled up well with the number of active CPU cores, and also significantly outperformed TAYGA and map646.

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, 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.

## Acknowledgements

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.

## References

- [1] Bao, C., Li, X., Baker, F., Anderson, T., & Gont, F. (2016). IP/ICMP translation algorithm, IETF RFC 7915, DOI: 10.17487/RFC7915
- [2] Bagnulo, M., Matthews, P., & Beijnum, I. (2011). Stateful NAT64: Network address and protocol translation from IPv6 clients to IPv4 servers, IETF RFC 6146, DOI: 10.17487/RFC6146
- [3] Bagnulo, M., Sullivan, A., Matthews, P., & Beijnum, I. (2011). DNS64: DNS extensions for network address translation from IPv6 clients to IPv4 servers, IETF RFC 6147, DOI: 10.17487/RFC6147
- [4] Mawatari, M., Kawashima, M., & Byrne, C. (2013). 464XLAT: Combination of stateful and stateless translation, IETF RFC 6877, DOI: 10.17487/RFC6877
- [5] Shima, K., Ishida, W., & Sekiya, Y. (2012). Designing an IPv6-oriented datacenter with IPv4-IPv6 translation technology for future datacenter operation. In: I. Ivanov, M. van Sinderen, F. Leymann, T. Shan (eds) *Cloud Computing and Services Science*. (CLOSER 2012). Porto, Portugal, Apr. 2012. pp. 39–53, *Communications in Computer and Information Science*, vol. 367. Springer, DOI: 10.1007/978-3-319-04519-1\_3
- [6] Free Software Foundation, The free software definition, <http://www.gnu.org/philosophy/free-sw.en.html> Accessed May 1, 2019.
- [7] Georgescu, M., Pislaru L., & Lencse, G. (2017). Benchmarking methodology for IPv6 transition technologies, IETF RFC 8219, DOI: 10.17487/RFC8219
- [8] Lencse, G., & Kadobayashi, Y., (2019). Comprehensive survey of IPv6 transition technologies: A subjective classification for security analysis, *IEICE Transactions on Communications*, to be published in E102-B(10), October 2019. DOI: 10.1587/transcom.2018EBR0002
- [9] Bálint, P. (2017). 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, Jul. 5-7, pp. 74–78. DOI: 10.1109/TSP.2017.8075940
- [10] Scholz, D. (2014). A look at Intel’s dataplane development kit, *Proc. Seminars Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM)*, Munich, Germany, pp. 115–122, DOI: 10.2313/NET-2014-08-1\_15
- [11] Bradner, S., & McQuaid, J. (1999). Benchmarking methodology for network interconnect devices, IETF RFC 2544, DOI: 10.17487/RFC2544
- [12] Popoviciu, C., Hamza, A., Van de Velde, G., & Dugatkin, D. (2018). IPv6 benchmarking methodology for network interconnect devices, IETF RFC 5180, DOI: 10.17487/RFC5180
- [13] Lencse, G. (2019). Benchmarking stateless NAT64 implementations with a standard tester, under second review in *Telecommunication Systems*, <http://www.hit.bme.hu/~lencse/publications/STS-2019-SLNAT64-Legacy-revised.pdf>
- [14] Lencse, G., Georgescu, M., & Kadobayashi, Y. (2017). Benchmarking methodology for DNS64 servers, *Computer Communications*, 109(1), pp. 162–175, DOI: 10.1016/j.comcom.2017.06.004
- [15] Paxson, V., Almes, G., Mahdavi, J., & Mathis, M. Framework for IP Performance Metrics, IETF RFC 2330, DOI: 10.17487/RFC2330
- [16] Anderson, T., Potter, A. L. (2016). Explicit address mappings for stateless IP/ICMP Translation, IETF RFC 7757, DOI: 10.17487/RFC7757
- [17] Lencse, G., Bakai, D. (2017). Design and implementation of a test program for benchmarking DNS64 servers, *IEICE Transactions on Communications*, vol. E100-B, no. 6. pp. 948-954, DOI: 10.1587/transcom.2016EBN0007
- [18] Lencse, G., & Kadobayashi, Y. (2018). Benchmarking DNS64 implementations: Theory and practice, *Computer Communications*, 127(1), pp. 61–74, DOI: 10.1016/j.comcom.2018.05.005
- [19] Lencse, G., & Répás, S. (2016). 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
- [20] Lutchansky, N. (2011). TAYGA: Simple, no-fuss NAT64 for Linux, <http://www.litech.org/tayga/> Accessed May 1, 2019.
- [21] NIC Mexico, (2019). Jool: SIIT and NAT64, <http://www.jool.mx/en/about.html> Accessed May 1, 2019.
- [22] Shima, K. (2010). A one to one address mapping program for translation from IPv4 packets to IPv6 packets and vice versa, source code of map646, <https://github.com/keiichishima/map646>
- [23] Tolly, K. (2001) The real meaning of zero-loss testing, *IT World Canada*, <https://www.itworldcanada.com/article/kevin-tolly-the-real-meaning-of-zero-loss-testing/33066> Accessed May 1, 2019.
- [24] Lencse, G. (2019). Performance analysis of authoritative DNS servers, under review in *Computer Networks*, <http://www.hit.bme.hu/~lencse/publications/ECN-2019-AuthDNS-for-review.pdf>
- [25] Majkowski, M. (2015). How to receive a million packets per second, *Cloudflare Blog*, <https://blog.cloudflare.com/how-to-receive-a-million-packets/> Accessed May 1, 2019.
- [26] Lencse, G. (2017) “Benchmarking methodology for IPv6 transition technologies”, IJ Lab seminar, Tokyo, Oct. 10, 2017, <https://seminar-materials.ijlab.net/ijlab-seminar/ijlab-seminar-20171010.pdf> Accessed May 1, 2019.
- [27] Tolly, E. (2016). “Mellanox spectrum vs. Broadcom strataXGS tomahawk 25GbE & 100GbE performance evaluation”, Tolly Test Report #216112, <http://www.mellanox.com/related-docs/tolly/tolly-report-performance-evaluation-2016-march.pdf> Accessed May 1, 2019.



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

He works 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 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 IJ Innovation Institute, Japan from April 2 to July 2, 2019, where his research topic was the performance analysis of SIIT implementations.

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



**Keiichi Shima** is a deputy director at the Research Laboratory of IJ 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.