Telecommunication Systems (2020) 75:245–257
https://doi.org/10.1007/s11235-020-00681-x

Benchmarking stateless NAT64 implementations with a standard tester

G. Lencse[1]

Published online: 15 June 2020

Abstract
RFC 5180, the IPv6 update of RFC 2544, declared IPv6 transition technologies out of its scope. RFC 8219 defined a benchmarking methodology for IPv6 transition technologies including stateless NAT64 (more properly called SIIT) in the category of single translation solutions. Whereas several research papers have dealt with the performance of different stateful NAT64 implementations, none of them used RFC 8219 compliant measurements or addressed stateless NAT64 implementations. In this paper, we show, how stateless NAT64 implementations can be benchmarked carrying out the most important tests recommended by RFC 8219 without a special purpose NAT64 Tester, using simply an RFC 2544/RFC 5180 compliant legacy Tester. We carry out benchmarking measurements to examine the performance of three free software NAT64 implementations, namely: Jool, TAYGA and map646. All the details of our measurements are disclosed and their results are presented in the paper.

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

1 Introduction

In the current phase of transitioning from IPv4 to IPv6 it is becoming more common that Internet Service Providers (ISPs) would like to use only IPv6 in their core and access networks, while some content providers still assign only IPv4 addresses to their servers. The combination of DNS64 [1] and stateful NAT64 [2] can be a proper solution to facilitate the communication of an IPv6-only client and an IPv4-only server. Further issue arises from the fact that certain applications can use only IPv4.[1] 464XLAT [4] is a good and widely used solution for this issue. 464XLAT uses stateless translation from IPv4 to IPv6 (NAT46) at the client side and stateful translation from IPv6 to IPv4 (NAT64) at the core network.

Although it is less common, but it is also an increasing tendency that content providers would like to use solely IPv6 in their internal network, whereas they still need to provide dual stack access to their services. This situation can be handled by stateless translation from IPv4 to IPv6 (NAT46). One of the earliest such solution was done as a part of the WIDE project [5].

Whereas a connection through a stateful NAT64 gateway may be initiated exclusively from the IPv6 side, stateless NAT64 (more properly called SIIT [6]) translators have no such constraints. Many free software [7] (also called open source [8]) implementations exist for stateful or stateless NAT64 and some of them support both. Their stability and performance may be an important factor, when network operators are selecting from among them.

RFC 2544 [9] defines a benchmarking methodology for network interconnect devices to facilitate proper and unbiased performance measurements. Although it is theoretically IP version independent, it focuses on IPv4. RFC 5180 [10] provides a technology update (e.g. regarding the frame rates of contemporary media) and addresses IPv6 specificities, but it explicitly states that: "IPv6 transition mechanisms are outside the scope of this document". RFC 8219 [11] defines a benchmarking methodology for IPv6 transition technologies. Since the number of IPv6 transition technologies is rather high [12] it classifies them into a small number of categories and it defines the benchmarking methods for the categories not for the individual IPv6 transition technologies. Both state-

[1] For example, several IPv4-only applications are listed on slide 10 of [3].

✉ G. Lencse
lencse@hit.bme.hu

[1] Department of Networked Systems and Services, Budapest University of Technology and Economics, 2 Magyar Tudósok körútja, Budapest 1117, Hungary

ful NAT64 and stateless NAT64 belong to the category of single translation solutions, thus the same tests should be used for their benchmarking. The difference is that there are some additional tests defined for the stateful solutions, see Sect. 8 of RFC 8219 for the details.

The purpose of this paper is twofold:

- to develop a method for benchmarking stateless NAT64 implementations without a special purpose NAT64 Tester, using simply an RFC 2544/RFC 5180 compliant legacy Tester,
- to carry out the benchmarking of a few stateless NAT64 implementations.

The remainder of this paper is organized as follows. In Sect. 2, we give a short overview of research papers on the performance analysis of various NAT64 implementations. In Sect. 3, we highlight the benchmarking method for single translation solution defined in RFC 8219 and its most important differences from and similarities to RFC 2544/RFC 5180 tests. In Sect. 4, we examine different possibilities, how legacy RFC 2544/RFC 5180 compliant testers can be used for benchmarking stateless NAT64 implementations. In Sect. 5, we present the details of our benchmarking measurements and also disclose and evaluate our results. Section 6 concludes our paper.

2 A short survey of papers on the performance analysis of various NAT64 implementations

Several papers on the performance analysis of different stateful NAT64 implementations were published. The first group of papers measured together the performance of a given NAT64 implementation with that of a given DNS64 implementation. In [13] the performance of the TAYGA NAT64 implementation (and implicitly that of the TOTD DNS64 implementation) is compared to the performance of NAT44. In [14], the performance of the Ecdysis NAT64 implementation (and that of its own DNS64 implementation) is compared to the performance of the authors' own HTTP ALG. In [15], the performance of the Ecdysis NAT64 implementation (and implicitly that of its DNS64 implementation) is compared to the performance of both the NAT-PT and an HTTP ALG. In [16], we argued 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."

In [17], we have developed a method suitable for independent performance analysis and stability testing of NAT64 and DNS64 implementations. In [18], we have compared the performances of TAYGA and OpenBSD PF using this method. In [19] we repeated our measurements using also TCP and UDP over IP besides ICMP, which was used solely in [18]. Whereas the method we used was suitable for performance comparison and stability analysis, it has several limitations, e.g. its results depend on the whole test setup, not only on the performance of the DUT (Device Under Test), thus it is not suitable for benchmarking.

In [20], the performance of different IPv6 transition solutions were measured including the TAYGA and the Jool NAT64 implementations by means of one way delay and throughput. Not surprisingly, neither the measurements of this one comply with the requirements of the later published RFC 8219 as the throughput was measured by iperf.

We note that although TAYGA is a stateless NAT64 implementation it was always used together with iptables to provide stateful NAT64. Similarly, although Jool can do both stateful and stateless NAT64, it was used to provide a stateful NAT64 service.

Whereas we consider the above results important and useful, we point out that they do not include stateless NAT64 tests and none of them complies with RFC 8219, which was published later than any of the above mentioned papers.

We are aware of only one paper reported an RFC 8219 compliant stateless NAT64 testing tool [21]. However, this tool cannot be used for benchmarking in practice due to its low performance, this is why its author has started re-implementing it using DPDK [22].

3 Benchmarking method for NAT64 gateways

Although RFC 8219 discusses all single translation solutions together, now we focus on NAT64, under which we understand both stateful NAT64 and stateless NAT64 as well as stateless NAT46.

The first and most important difference from the RFC 2544 (RFC 5180) benchmarking is visible on the Test Setup in Fig. 1. The fact that the two ports use different IP versions has the consequences that legacy RFC 2544 (RFC 5180) compliant Testers, which expect the same IP version, fail to operate (unless an appropriate trick is used).

Except this very important difference, the majority of the measurement procedures were kept unchanged. They are: *throughput*, *frame loss rate*, *back-to-back frames*, *system recovery* and *reset*. The measurement procedure for *latency* has been redefined to achieve higher accuracy, and further

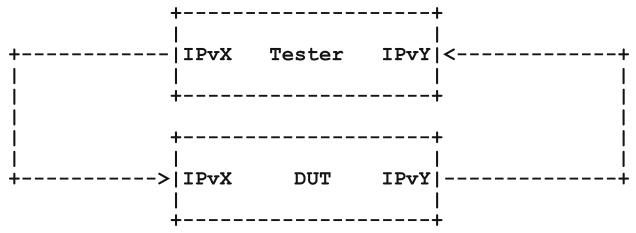

Fig. 1 Single DUT test setup [11]

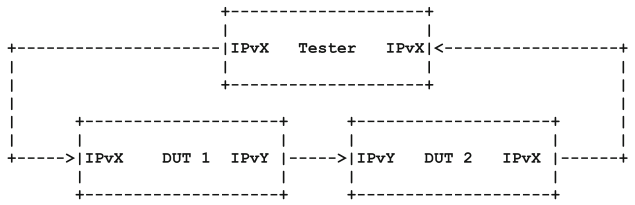

Fig. 2 Dual DUT test setup [11]

measurement procedures for *packet delay variation* and *inter packet delay variation* have been added. For more details, please refer to Section 7 of RFC 8219.

We note that, similarly to its predecessors, RFC 8219 requires bidirectional traffic for testing and allows also unidirectional tests. For us, it also means that we do not need to distinguish stateless NAT64 and stateless NAT46.

Section 8 of RFC 8219 describes additional tests for stateful IPv6 transition solutions, which can be applied to stateful NAT64. For us, the point is that the basic tests of the stateful NAT64 are the same as that of the stateless NAT64.

4 How legacy testers can be reused for benchmarking stateless NAT64 gateways?

4.1 Which tests can be and should be reused?

Throughput and frame loss measurement procedures are the same and they are both very important, thus they definitely should be reused. As the latency measurement procedure has been changed, if the old one is reused, its results may give some valuable insight, but they are not comply with the RFC 8219. Measurement procedures for PDV and IPDV measurements as well as stateful tests are missing from the legacy Testers. Procedures for back-to-back frames, system recovery and reset test should exist, but in our understanding, they are seldom used, thus we do not deal with them. Therefore, we focus on throughput and frame loss rate measurements.

4.2 Reusing legacy testers with the dual DUT setup

4.2.1 Feasibility in theory

Although RFC 8219 recommends the Dual DUT setup for double translation and encapsulation technologies, it can provide us a viable solution for reusing the legacy Testers for benchmarking stateless NAT64 gateways. As it is shown in Fig. 2, the sequence of two translations that are the inverse of each other restores the original IP version. As for stateless NAT64, two equally good solutions are possible:

1. If we use NAT64 first and NAT46 after that, then the Tester must use IPv6.
2. If we use NAT46 first and NAT64 after that, then the Tester must use IPv4.

Of course, the dual DUT setup has several limitations and hindrances, for example:

- In the simplest case, two identical DUTs are used, which hides any possible asymmetric behavior (e.g. due to an implementation bug) and causes somewhat extra cost (as two DUTs are needed).
- One may try using two different DUTs, but it is not trivial how to make one of them the bottleneck for sure, due to possible unknown asymmetric behavior.
- It excludes the stateful NAT64 devices from testing. (Even the stateless tests may not be performed, because they do not allow connection establishment in the direction from IPv4 to IPv6.)

However, the most important advantage of the dual DUT setup is deliberate: the two most important tests may be performed using an existing legacy Tester.

4.2.2 Feasibility in practice

In order to test if this setup can be implemented in real life, we have put together a test system in a virtual environment using Debian GNU/Linux operating system. From the two, theoretically equally good solutions, we have chosen the second one, because it has the important practical advantage that even an old, IPv4-only Tester is enough, whereas the first one requires a Tester with IPv6 capabilities.

We have tested three stateless NAT64 implementations, TAYGA [23], Jool [24] and map646 [25]. We were successful with the first two ones, but failed with the third one. (We explain the reason of the failure at the end of this subsection.)

RFC 7757 [26] extended stateless NAT64 with EAM (Explicit Address Mapping), thus removed the constraints caused by the limited applicability of IPv4-convertible IPv6 addresses [27]. We have used this approach in our test systems. Figures 3 and 4 show the test setups with virtual machines using TAYGA and Jool, respectively. Whereas

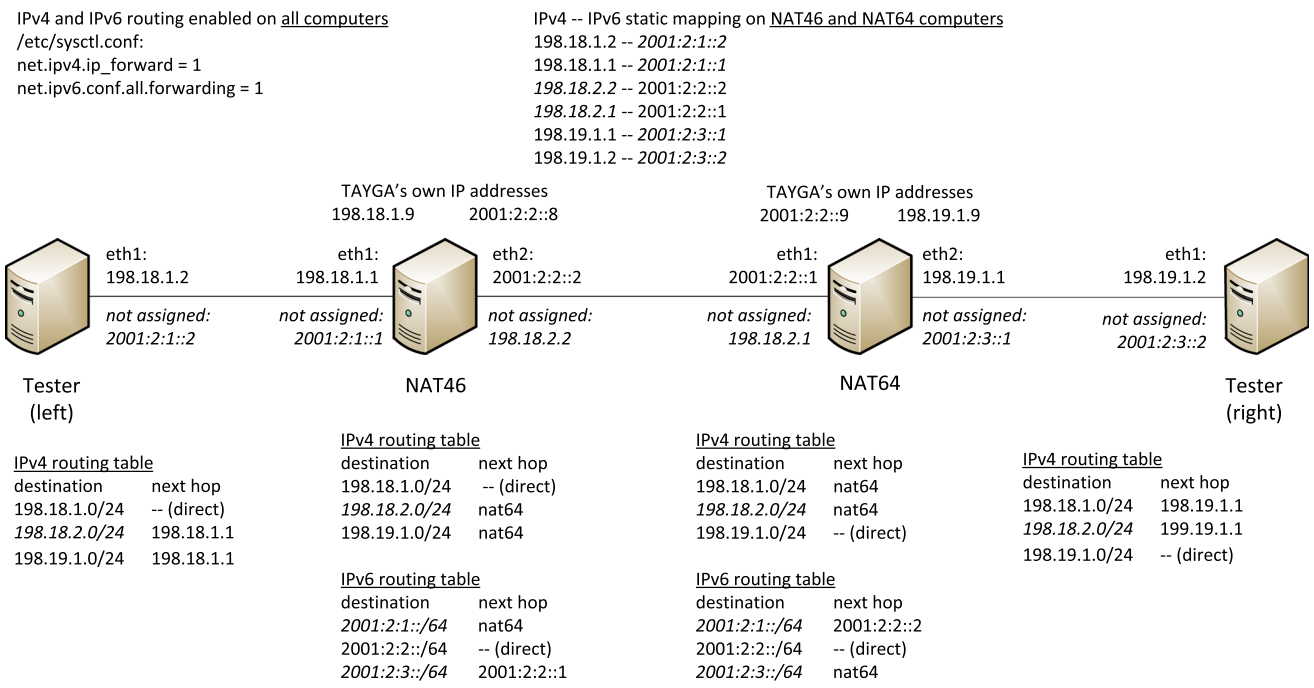

Fig. 3 Dual DUT test setup with virtual machines using TAYGA

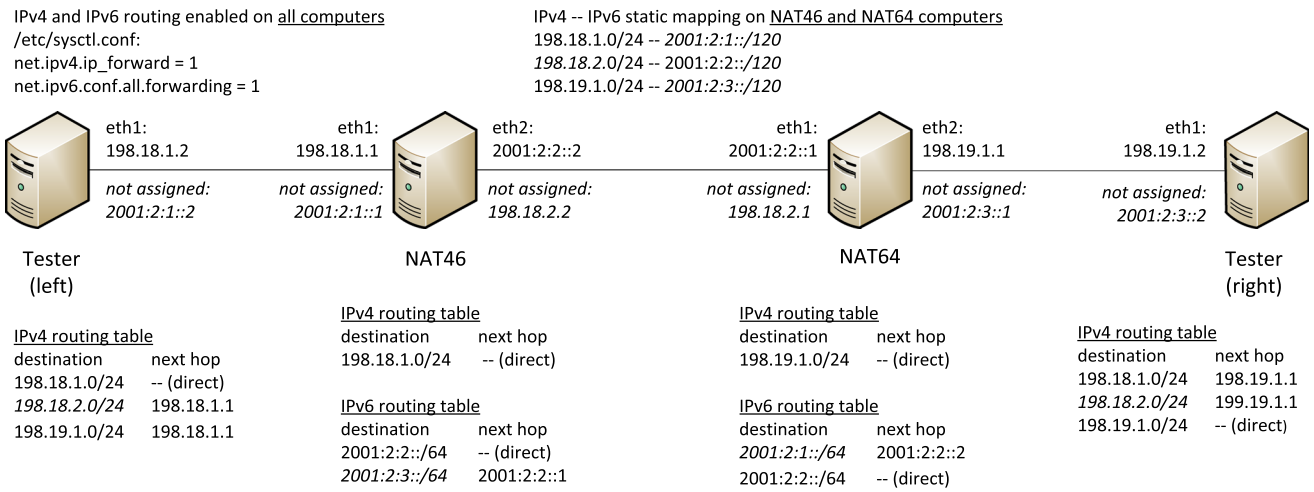

Fig. 4 Dual DUT test setup with virtual machines using Jool

there are some slight differences between the two setups due to particularities of TAYGA and Jool, the two figures show the same solution. The key of this solution is that every single network interface has either an IPv4 or an IPv6 address, which is really assigned to it, and they also "have" IP addresses from the other IP version, which are not assigned to them, but are used to represent them, when they need to be referred to using the other IP version. These peers are mapped to each other by EAM. The two ports of the Tester are represented by two virtual machines called "Tester (left)" and "Tester (right)". They are assigned only IPv4 address. The IPv6 addresses in italic font are not assigned to them, they are used to represent their IPv4 addresses in the IPv6 domain. Similarly, an IPv6 network connects the NAT46 and NAT64 gateways, thus the interfaces are assigned only IPv6 addresses, which have their corresponding IPv4 peers (written in italic font).

Routing was set manually to forward the packets according to the routing tables shown in Figs. 3 and 4.

We have checked that the first and fourth computers (representing the left and right side ports of the Tester) were mutually accessible from each other (using the `ping` Linux command).

For those, who would like to follow the operation, we put here an excerpt of the message flow resulted by a ping command issued at Tester (left) and targeted to Tester (right) captured by `tshark` at various interfaces of the NAT46 and NAT64 virtual machines. (They are from the TAYGA version, as TAYGA uses a nat64 virtual interface, which resulted in more details, thus an easier traceability of what happened.)

```
NAT46 eth1    198.18.1.2 -> 198.19.1.2   ICMP Echo request
NAT46 nat64   198.18.1.2 -> 198.19.1.2   ICMP Echo request
NAT46 nat64  2001:2:1::2 -> 2001:2:3::2  ICMPv6 Echo request
NAT46 eth2   2001:2:1::2 -> 2001:2:3::2  ICMPv6 Echo request
NAT64 eth1   2001:2:1::2 -> 2001:2:3::2  ICMPv6 Echo request
NAT64 nat64  2001:2:1::2 -> 2001:2:3::2  ICMPv6 Echo request
NAT64 nat64   198.18.1.2 -> 198.19.1.2   ICMP Echo request
NAT64 eth2    198.18.1.2 -> 198.19.1.2   ICMP Echo request
NAT64 eth2    198.19.1.2 -> 198.18.1.2   ICMP Echo reply
NAT64 nat64   198.19.1.2 -> 198.18.1.2   ICMP Echo reply
NAT64 nat64  2001:2:3::2 -> 2001:2:1::2  ICMPv6 Echo reply
NAT64 eth1   2001:2:3::2 -> 2001:2:1::2  ICMPv6 Echo reply
NAT46 eth2   2001:2:3::2 -> 2001:2:1::2  ICMPv6 Echo reply
NAT46 nat64  2001:2:3::2 -> 2001:2:1::2  ICMPv6 Echo reply
NAT46 nat64   198.19.1.2 -> 198.18.1.2   ICMP Echo reply
NAT46 eth1    198.19.1.2 -> 198.18.1.2   ICMP Echo reply
```

Thus, we have successfully constructed a test setup for benchmarking stateless NAT64 gateways using legacy Testers having only IPv4 capabilities.

Now, let us return to map646. Its test setup was very much similar to that of TAYGA, therefore we do not repeat it. By studying its behavior using `tshark` captures, we have observed that map646 applied our EAM static mapping rules to the destination IPv4 address only, whereas it completely ignored them concerning the source IPv4 address, and it rather synthesized an IPv4-embedded IPv6 address [27] using the ::/96 prefix (as we have not specified a prefix) plus the 32 bits of the source IPv4 address. We could circumvent this behavior, but, consistently to this behavior, map646 also ignores the mapping rule for the destination address, when it performs the translation from IPv6 to IPv4 and it expects an IPv4-embedded IPv6 address.

We have contacted its author, Keiichi Shima, who confirmed that map646 was willfully designed so (to avoid handling IPv6 neighbor discovery proxy operation, which would have made the code more complex), as it was developed to be a NAT46 gateway solution for the WIDE project, for which this behavior was completely satisfactory, however, it also means that map646 may not be used in the Dual DUT setup.

4.3 Reusing legacy testers with the single DUT setup

RFC 8219 recommends the single DUT setup for single translation technologies.

4.3.1 Single DUT test setup using virtual machines

First, let us see the test setups with virtual machines for benchmarking the three before mentioned stateless NAT64 implementations. Figures 5, 6 and 7 show the test setups with virtual machines using TAYGA, Jool, and map646, respectively. We believe that the test setups for TAYGA and Jool are easy to follow after the understanding of their Dual DUT test setups. Map646 is somewhat similar to TAYGA in the sense, that it also uses a pseudo interface, tun646, however, its operation is rather different from that of TAYGA. To reveal the difference, let us compare their message flows. First, let us see an excerpt of the message flow of TAYGA resulted by a ping command issued at Tester (right) and targeted to Tester (left) captured by `tshark` at various interfaces of the NAT64 virtual machine.

```
eth2     198.19.0.2 -> 198.18.0.2     ICMP 98 Echo request
nat64    198.19.0.2 -> 198.18.0.2     ICMP 84 Echo request
nat64  2001:2:0:1::2 -> 2001:2::2     ICMPv6 104 Echo request
eth1   2001:2:0:1::2 -> 2001:2::2     ICMPv6 118 Echo request
eth1      2001:2::2 -> 2001:2:0:1::2  ICMPv6 118 Echo reply
nat64     2001:2::2 -> 2001:2:0:1::2  ICMPv6 104 Echo reply
nat64   198.18.0.2 -> 198.19.0.2      ICMP 84 Echo reply
eth2    198.18.0.2 -> 198.19.0.2      ICMP 98 Echo reply
```

This is similar to the message flow of TAYGA with the Dual DUT setup. However, the message flow of the same ping command looks differently with map646.

```
eth2      198.19.0.2 -> 198.18.0.2   ICMP 98 Echo request
tun646    198.19.0.2 -> 198.18.0.2   ICMP 84 Echo request
tun646    64::c613:2 -> 2001:2::2    ICMPv6 104 Echo request
eth1      64::c613:2 -> 2001:2::2    ICMPv6 118 Echo request
eth1       2001:2::2 -> 64::c613:2   ICMPv6 118 Echo reply
tun646     2001:2::2 -> 64::c613:2   ICMPv6 104 Echo reply
tun646    198.18.0.2 -> 198.19.0.2   ICMP 84 Echo reply
eth2      198.18.0.2 -> 198.19.0.2   ICMP 98 Echo reply
```

As we have mentioned before, when doing the translation from IPv4 to IPv6, map646 uses EAM only concerning the destination IPv4 address. In our example, it synthesized an IPv4-embedded IPv6 address using the specified 64::/96 prefix as the IPv6 source address. Similarly, in the IPv6 to IPv4 direction, map646 uses EAM only for the source address, and it expects an IPv4 embedded-IPv6 address as the destination address, otherwise is does not work. Our example demonstrated that this functionality is enough to provide an IPv4 access to IPv6 only servers (the `ping` command was successful). This limitation also means that map646 cannot be tested in the Dual DUT setup, but (in itself) it should not prevent us from benchmarking a map646 gateway according to the Single DUT setup with bidirectional traffic.

A `ping6 64::198.19.0.2` command issued at Tester (left) using the IPv4 embedded-IPv6 address of Tester (right) works perfectly resulting the following message flow.

```
eth1     2001:2::2 -> 64::c613:2   ICMPv6 118 Echo request
tun646   2001:2::2 -> 64::c613:2   ICMPv6 104 Echo request
tun646  198.18.0.2 -> 198.19.0.2   ICMP 84 Echo request
eth2    198.18.0.2 -> 198.19.0.2   ICMP 98 Echo request
eth2    198.19.0.2 -> 198.18.0.2   ICMP 98 Echo reply
tun646  198.19.0.2 -> 198.18.0.2   ICMP 84 Echo reply
tun646  64::c613:2 -> 2001:2::2    ICMPv6 104 Echo reply
eth1    64::c613:2 -> 2001:2::2    ICMPv6 118 Echo reply
```

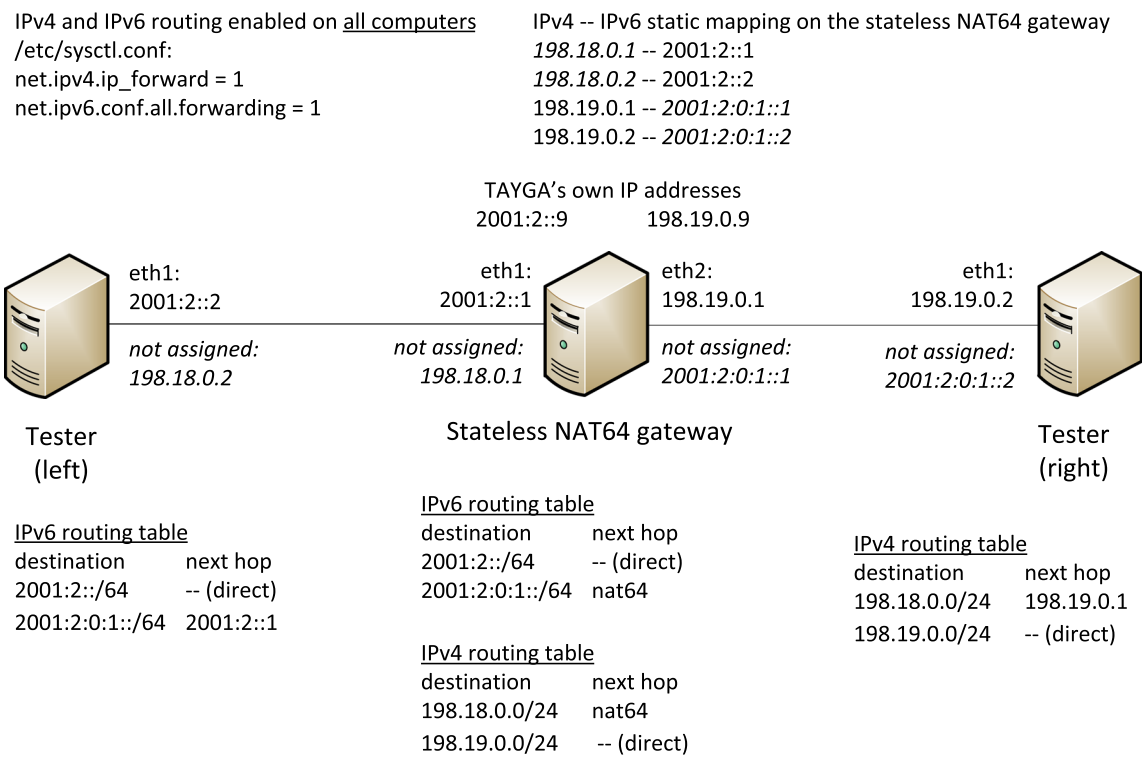

Fig. 5 Single DUT test setup with virtual machines using TAYGA

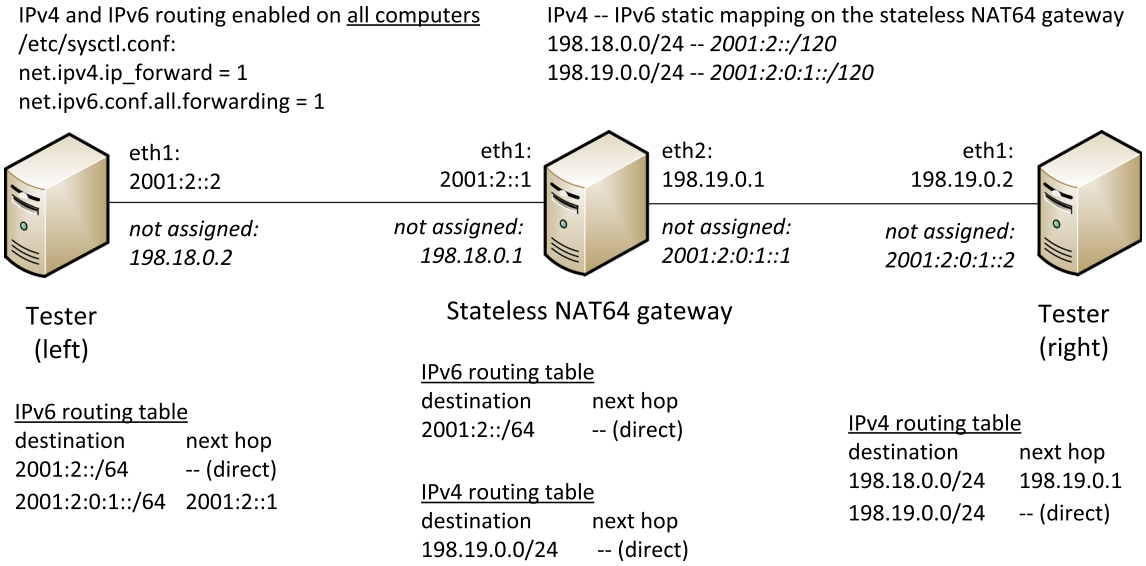

Fig. 6 Single DUT test setup with virtual machines using Jool

Thus, map646 may be benchmarked according to the Single DUT setup, but it visibly requires different network settings at the Tester than TAYGA and Jool.

4.3.2 Application of a legacy tester

A legacy Tester must comply with the following two requirements so that it may be used for benchmarking stateless NAT64 implementations according to the Single DUT setup:

1. It must support the RFC 2544 (RFC 5180) tests for both IPv4 and IPv6.

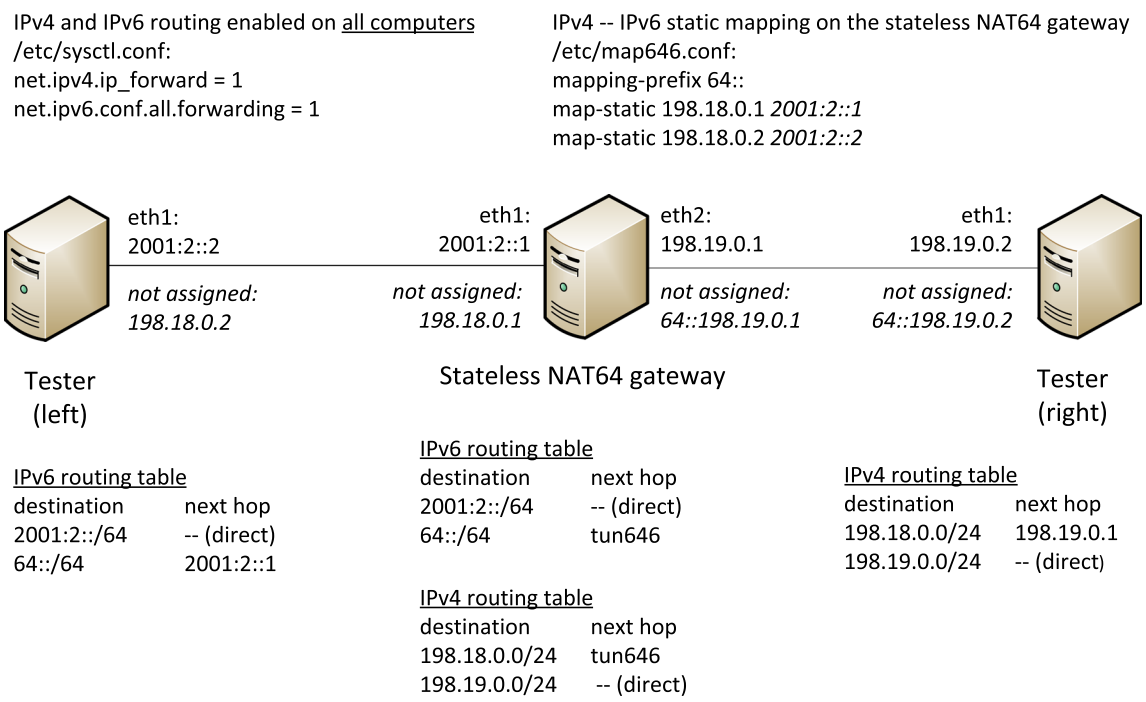

Fig. 7 Single DUT test setup with virtual machines using map646

2. It must allow the user not to connect all its "logical ports" physically. (It is needed for our "trick".)

As different Testers may use different technical terms, from now on we use that of a Spirent SPT-N4U Tester, which we used for our measurements.

Figure 8 shows the Single DUT test setup using a Dell PowerEdge T630 NAT64 server as DUT and a Spirent SPT-N4U Tester with and MX2-10G-S8 card, which we used for benchmarking TAYGA and Jool.

To provide the Tester with an "acceptable situation", two unidirectional flows were set up. IPv6 packets were sent from virtual device 1 to virtual device 4 and IPv4 packets were sent from virtual device 3 to virtual device 2. However, only virtual device 1 and virtual device 3 were actually connected to the DUT. The IPv6 packets from the Tester were translated to IPv4 by the stateless NAT gateway and they arrived to the connected virtual device 3 (instead of the unconnected virtual device 4). Similarly, the IPv4 packets from the Tester were translated to IPv6 and they arrived to the connected virtual device 1 (instead of the unconnected virtual device 2). With this trick, we could achieve that the Tester sent and received the appropriate IP version packets for benchmarking the stateless NAT64 gateway.

As for map646, the same structure was used with a slightly different addressing as shown in Fig. 9.

The details of the benchmarking measurements are discussed in Sect. 5.

5 Benchmarking measurements

5.1 Frame size considerations

RFC 8219 recommends the following frame sizes for testing Ethernet devices: 64, 128, 256, 512, 768, 1024, 1280, 1518 bytes. It also mentions that 84 bytes should be used for single-translation transition technologies (e.g., NAT64) in the IPv6 to Pv4 direction. It is so, because the translation from IPv6 to IPv4 decreases the frame size by 20 bytes to 64 bytes, which is the minimum allowed frames size for Ethernet. Of course, the phenomenon exits in the opposite direction, too: 1498 bytes long IPv4 frames will be converted to 1518 bytes long IPv6 frames during the NAT46 translation, which is the maximum allowed frames size for Ethernet. As RFC 8219 does not say anything about which IP version should use the above specified frames sizes, we decided that we correct the first or last frame size value, when needed, and keep the other values untouched for *sending*, which means that the frame sizes of the received frames differed from the listed ones. Table 1 shows the frame sizes we planned to use for benchmarking. (The above listed and the modified frame sizes are typeset in bold and italic fonts, respectively.)

Another consequence of the translation is that the traffic volume measured in bytes is changed by the translation in both directions. To make our results unambiguous, we either need to express our results in *number of frames per second* or, if we use *number of bytes per second*, then we must also mention the *IP version* besides the frame size, too. We

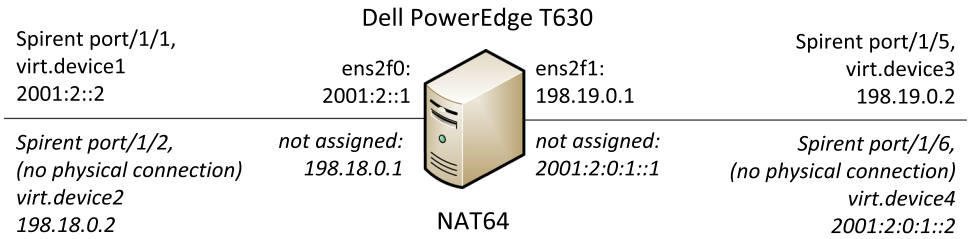

Fig. 8 Single DUT test setup with a Spirent SPT-N4U Tester for benchmarking TAYGA and Jool

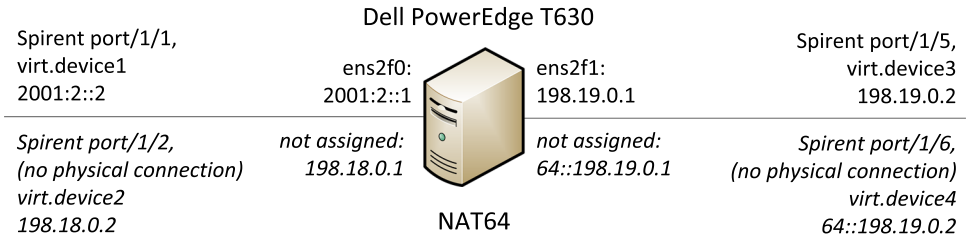

Fig. 9 Single DUT test setup with a Spirent SPT-N4U Tester for benchmarking map646

Table 1 Frame Sizes used for Benchmarking NAT64 Gateways

NAT64	IPv6	*84*	**128**	**256**	**512**	**768**	**1024**	**1280**	**1518**
	IPv4	**64**	*108*	*236*	*492*	*748*	*1004*	*1260*	*1498*
NAT46	IPv4	**64**	**128**	**256**	**512**	**768**	**1024**	**1280**	*1498*
	IPv6	*84*	*148*	*276*	*532*	*788*	*1044*	*1300*	**1518**

have chosen the first option, and specified the load always in *frames per second*.

5.2 Parameters and settings

The measurements were performed according to the setups shown in Figs. 8 and 9. The Spirent SPT-N4U Tester had an MX2-10G-S8 card. The most important parameters of the Dell PowerEdge T630 server were:

- 2× Intel Xeon E5-2698 v3 CPUs
- 8× 16 GB RDIMM, 2133 MT/s, Dual Rank, ×4 Data Width memory modules
- 2× 10 Gbps SFP + ethernet ports

We have switched off Hyper Threading in the BIOS setup, because it could have caused scattered results according to our previous benchmarking experience [28]. Thus, the Linux operating system displayed 32 CPU cores. As for NUMA situation, CPU cores 0–15 belonged to NUMA node 0, and the other ones to NUMA node 1. Memory was distributed evenly between the two nodes and all the I/O devices (NICs, HDD, etc.) were connected to node 0.

For the repeatability of our measurements, we also document the software versions:

- TAYGA 0.9.2 [23]
- Jool 3.5.7 [24]
- map646 (GitHub latest commit cd93431 on Mar 31, 2016) [25]

As for the parameters of the benchmarking measurements, the *Trial Duration* was set to 60 s. Binary search was used with *Rate lower limit* 0.001% and *Rate upper limit* 100%, and the *Resolution* was set to 0.01%, which we considered a good compromise between speed and accuracy.

RFC 8219 requires bidirectional throughput tests with absolutely 0 frame loss. We used this one, but we note that the Tester offers a possibility to set non-zero frame loss rate.

The parameters set in the *RFC 2544 Throughput Parameters* dialog box of the Spirent Tester apply to all frames to be sent: the user cannot specify distinct values for IPv6 and IPv4. Therefore, for bidirectional tests, we were not able to use the distinct frame size values for IPv6 and IPv4 frames presented in the first and last column of Table 1. Thus, for bidirectional tests, we used only the 128, 256, 512, 768, 1024, 1280 bytes frame sizes, which were the same for IPv6 and IPv4.

Besides the required bidirectional tests, we also performed unidirectional tests to gain further insight into the operation of the tested NAT64 implementations. With these tests, we used the frames sizes shown in the first and third rows of Table 1 as IPv6 and IPv4 frame sizes, respectively.

As for frames loss rate tests, theoretically the entire frame rate range of the media (10 Gbps Ethernet) should have been tested using at most 10% granularity. However, considering the results of the throughput tests, we have chosen reduced ranges with higher resolution to produce meaningful results.

5.3 Throughput results

First, we present and discuss the results of the three tested implementations separately and, we compare them after that.

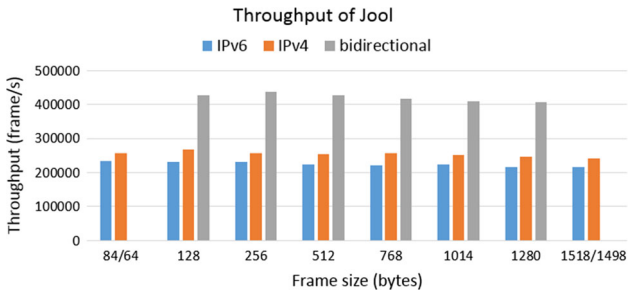

Fig. 10 Throughput of Jool as a function of frame size and traffic type

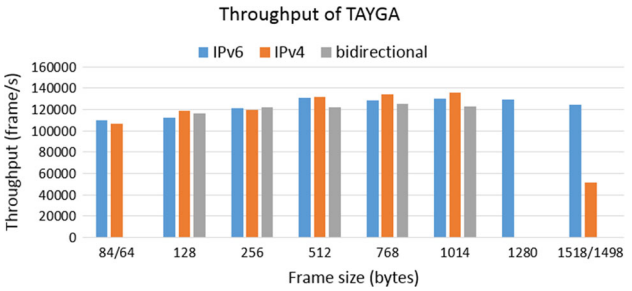

Fig. 11 Throughput of TAYGA as a function of frame size and traffic type

5.3.1 Jool

The throughput results of Jool are shown in Fig. 10. The horizontal axis shows the frame size. Frame sizes 84 and 1518 belong to frames sent by virtual device 1, similarly, frame sizes 64 and 1498 belong to frames sent by virtual device 3. The other frame sizes are common. And all frame sizes are to be interpreted as the sizes of the *sent* frames. (Sizes of the received frames were 20 bytes shorter and 20 longer due to NAT64 and NAT46 translation, respectively.) In the legend, IPv6 and IPv4 indicate the types of the sent frames, when unidirectional traffic was used.

We can observe that the number of frames per second shows a very slight degradation as the frame size increases, being the decrease so small that the number of frames per second could be called roughly independent from the frame size. (For the explanation of the slight fluctuations, please refer to the frame loss tests in Sect. 5.4.1.) This observation can be explained by the fact that NAT64/NAT46 translation affects only the IP header, and the bottleneck is surely the processing capacity of the CPU not the transmission capacity of the 10 Gbps Ethernet. The fact that the unidirectional throughput was about 230,000 fps for IPv6 and about 250,000 fps for IPv4, whereas bidirectional throughput is about 430,000fps, which means 215,000 fps for each directions, complies with our observations during preliminary testing that only a single CPU core was working, when unidirectional traffic was used, but Jool could utilize two CPU cores, when bidirectional traffic was used.

5.3.2 Tayga

The throughput results of TAYGA are shown in Fig. 11. (The notations of the figure are to be interpreted as that of Fig. 10.) The results of the IPv4 unidirectional and bidirectional tests with 1280 bytes long frames seem to be missing. It is so, because the throughput tests reported 0 packet per second in both cases. (The repeated test gave the same results.) To investigate the issue, we have checked the detailed results of the unidirectional IPv4 test, and we have found that all the frames were lost at 1280 bytes frame size independently from the load conditions. We have found this phenomenon very strange, especially that the unidirectional IPv4 test with 1498 bytes long frames was successful, though the throughput was about halved compared to other frame sizes. To find the root causes of this behavior, we have checked the traffic at the DUT with `tshark`. Figure 12 shows a fraction of the traffic captured at the `nat64` interface using 1280 bytes long frames. It is visible that TAYGA has fragmented the incoming IPv4 packet (no. 86) into two IPv6 packets (no. 87 and 88), which were not recognized as valid by the Tester, and thus it sent back an ICMPv6 error message (no. 89), which was then translated to an ICMPv4 error message (no. 90) by TAYGA. The situation is partially different with 1498 bytes long frames, as shown in Fig. 13, which contains the packets belonging to two consecutive test frames. (The packets of the second one are displayed to show that the first one did not result in an ICMPv6 error message.) Fragmentation also happens here, but now the Tester accepts the fragmented test frames. Of course, fragmentation had its computational cost, hence the maximum achievable rate decreased to about less than one half.

Considering the other packet rates, the measured throughput of TAYGA is roughly independent from the frame size.

The fact that the bidirectional throughput of TAYGA looks approximately the same as its unidirectional throughput (which means twice one half per direction), complies with our observation that TAYGA could utilize only a single CPU core, even when bidirectional traffic was used.

5.3.3 Map646

The throughput results of map646 are shown in Fig. 14. (The notations of the figure are to be interpreted as that of Fig. 10.) Here, the results of the IPv4 test with 1498 bytes frame size seem to be missing, since all frames of this size were lost at any packet rate. It is also caused by fragmentation.

We have attempted to test the performance of map646 in the IPv6 to IPv4 direction, however, we have got the following error message from map646:

```
86 1.768027036    198.19.0.2 → 198.18.0.2   IPv4 1262 Unknown (253)
87 1.768068056 2001:2:0:1::2 → 2001:2::2    IPv6 1280 IPv6 fragment (off=0 more=y ident=0x00007814 nxt=253)
88 1.768076843 2001:2:0:1::2 → 2001:2::2    IPv6 58 Unknown IP Protocol: Unknown (253)
89 1.768489419     2001:2::2 → 2001:2:0:1::2 ICMPv6 1280 Parameter Problem (unrecognized Next Header type encountered)
90 1.768529993    198.18.0.2 → 198.19.0.2   ICMP 576 Destination unreachable (Protocol unreachable)
```

Fig. 12 A `tshark` capture of the traffic at the nat64 interface of TAYGA, using 1280 bytes test frames

```
46 1.821626681    198.19.0.2 → 198.18.0.2   IPv4 1480 Unknown (253)
47 1.821668296 2001:2:0:1::2 → 2001:2::2    IPv6 1280 IPv6 fragment (off=0 more=y ident=0x00007cca nxt=253)
48 1.821677746 2001:2:0:1::2 → 2001:2::2    IPv6 276 Unknown IP Protocol: Unknown (253)
49 1.943062342    198.19.0.2 → 198.18.0.2   IPv4 1480 Unknown (253)
50 1.943103537 2001:2:0:1::2 → 2001:2::2    IPv6 1280 IPv6 fragment (off=0 more=y ident=0x00007ccb nxt=253)
51 1.943112350 2001:2:0:1::2 → 2001:2::2    IPv6 276 Unknown IP Protocol: Unknown (253)
```

Fig. 13 A `tshark` capture of the traffic at the nat64 interface of TAYGA, using 1498 bytes test frames

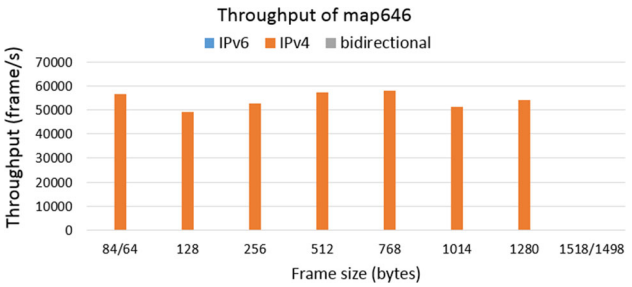

Fig. 14 Throughput of map646 as a function of frame size and traffic type

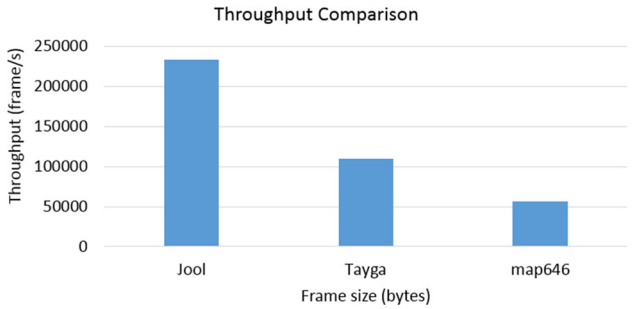

Fig. 15 Throughput of Jool, TAYGA, and map646 in the IPv4 to IPv6 direction with 64 bytes frame size

```
map646: Extention header 59 is not
supported.
```

The value 59 in the Next Header field of an IPv6 datagram means "No Next Header". As it can be seen from the captures in Figs. 12 and 13, the Tester sent raw IP packets with protocol type 253.[2] However, map646 expected one of TCP, UDP or ICMPv6. We know it from Keiichi Shima, that the aim of the checking of the Next Header field was to prevent incorrect translation of extension headers. (In the IPv4–IPv6 direction no such checking was necessary, as no extension headers may occur in the IPv4 packets, thus testing was possible in that direction.) For us, the point is that we could not perform the test in the IPv6 to IPv4 direction.

Otherwise the throughput of map646 shows some fluctuations, but is also roughly independent from the frame size.

5.3.4 Comparison

As the throughput of all three tested implementations is independent from the frame size, and we could test map646 only in the IPv4–IPv6 direction, we compare their IPv4 throughput using 64 bytes long test frames. Figure 15 shows the results. We note that the results were not surprising for us, as Jool is the latest implementation of them, which is still actively developed and is works in the kernel space [24]. TAYGA works in user space and it was "intended to provide production-quality NAT64 service for networks where dedicated NAT64 hardware would be overkill" [23], however, TAYGA is no more developed. Map646 was developed for a single purpose to serve as stateless NAT46 gateway solution for the WIDE project [5] and it was published as free software for public benefit out of courtesy.

5.4 Frame loss results

For the comparability of the results, Jool and TAYGA were tested under the same conditions: their frame loss rate was measured with IPv6 to IPv4 traffic, and the frame rate was increased from 50,000 to 800,000 in 50,000 fps steps. However, these limits were inappropriate for map646, which was tested using IPv4 to IPv6 traffic, and the frame rate was increased from 40,000 to 100,000 fps in 10,000 fps steps.

5.4.1 Jool

The frame loss rate of Jool as a function of frame rate and frame size is shown in Fig. 16. For all frame sizes, the frame loss rate is between 0.001 and 0.004% at 250,000 fps frame

[2] The 253 IP protocol field value was reserved for experimentation and testing purposes by RFC 3692.

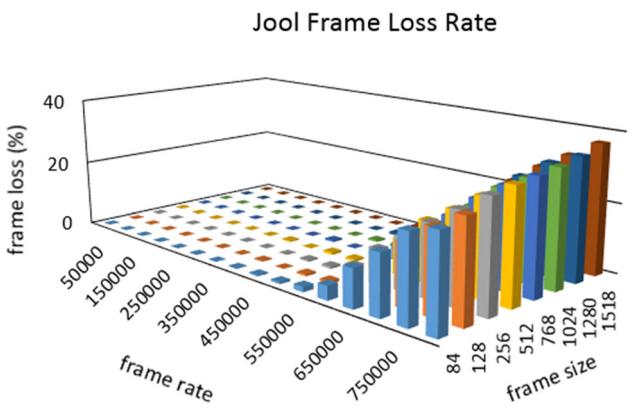

Fig. 16 Frame loss rate of Jool as a function of frame rate and frame size, using IPv6 to IPv4 traffic

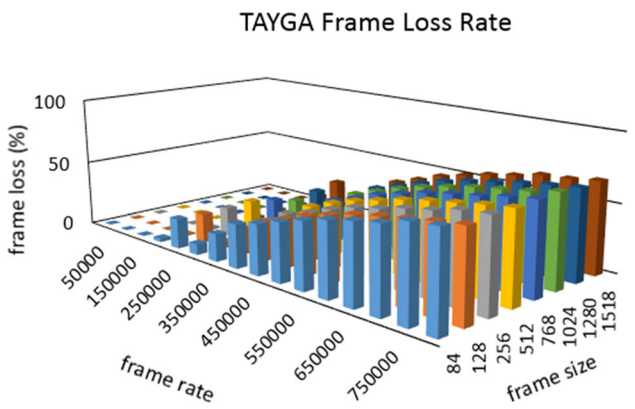

Fig. 17 Frame loss rate of TAYGA as a function of frame rate and frame size using IPv6 to IPv4 traffic

rate, it is below 0.01% and at 300,000 fps, and it is still below 1% at 500,000 fps, but it raises sharply from 550,000 fps frame rate.

Our results with very low frame loss rate up to significantly higher frame rates than the throughput measurement results taught us two very important lessons:

1. As RFC 2544 based throughput tests use a binary search to find the highest frame rate, where all frames are transmitted (that is, there is no frame loss), our results explain, why the results of multiple repetitions may differ significantly: tests sometimes fail due to a the loss of a very small number of frames. (And this observation also explains our observation in Sect. 5.3.1 that the throughput results at different frame sizes fluctuate.)
2. Users may experience significantly higher throughput (e.g. download speed) than the RFC 2544 throughput result, if TCP selective acknowledgment is enabled and the end to end delay is low enough.

5.4.2 Tayga

The frame loss rate of TAYGA as a function of frame rate and frame size is shown in Fig. 17. As a function of the frame rate, frame loss firs appears at 150,000 fps, where it is under 0.3% with all frame sizes. Frame loss rate suddenly jumps to about 20% at 250,000 fps, but significantly falls back at 300,000 fps, form where it continuously rises, and it is about 65% at 800,00 fps.

5.4.3 map646

The frame loss rate of map646 as a function of frame rate and frame size is shown in Fig. 18. We note that because of the IPv4–IPv6 direction, here the smallest frame size was 64 bytes and the largest one was 1498 bytes, however, we have omitted the results with 1498 bytes frames, as they were all

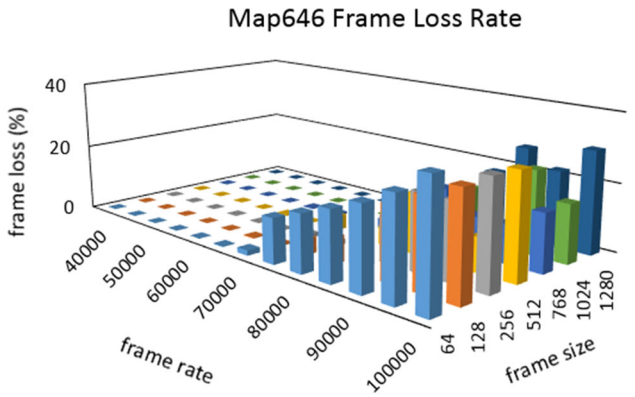

Fig. 18 Frame loss rate of map646 as a function of frame rate and frame size using IPv4–IPv6 traffic

100%, and thus the scaling of the figure would have been inappropriate for the other values.

The results of map646 are rather fluctuated, thus we can not characterize the behavior of the results with relevant statements that apply for all frame sizes. Less than 0.1% frame losses occurred occasionally under 60,000 fps. Non-zero, but $<0.33\%$ frame loss usually appeared at 60,000 fps. Frame loss achieved 13.2% at 75,000 fps with 64 bytes long frames, but it was under 5% with all other frames sizes. Frame loss usually achieved 50% at 100,000 fps, but it was only about 17% with 768 and 1024 bytes long frames.

5.4.4 Comparison

Whereas the behavior of the frame loss of Jool was consequent, the frame loss rates of both TAYGA and map646 have shown inconsistent behavior under certain parameter combinations. Considering also the throughput values, we consider that Jool is the most matured stateless NAT64 implementation from among the examined ones.

6 Further work and our future plans

During the review process of this paper we have performed benchmarking measurements using the before mentioned DPDK-based special purpose test software. The same implementations were tested, but Jool was the most current version, that is 4.0.1. Our results will be published in [29]. Unfortunately, the test program had several issues, which we could temporarily fix, but we plan to redesign and re-implement it in C++. We also plan to validate the new test program by comparing its results with the results produced by using a standard tester, thus our efforts described in this paper will be a great help for us in the validation.

7 Conclusion

We have pointed out that there was a gap in research papers concerning the benchmarking of stateless NAT64 (SIIT) implementations. We have demonstrated with virtual machines, how two kinds of test setups may be built for benchmarking stateless NAT64 implementations according to RFC 8219 without using a special purpose NAT64 Tester, rather by reusing legacy RFC 2544/RFC 5180 Testers. From among the tests defined in RFC 8219, the two most important ones, namely throughput and frame loss tests, can be performed by this way.

We have demonstrated the feasibility of benchmarking stateless NAT64 gateways according to the Single DUT setup by benchmarking three free software stateless NAT64 implementations: Jool, TAYGA and map646. We have found that Jool showed both the highest throughput and the most consistent behavior in the frame loss tests, thus we recommend Jool for new deployments. We hope that our results may contribute to the global deployment of the IPv6 protocol.

The measurement method described in this paper will also be useful in the validation of the planned special purpose NAT64 test software.

Acknowledgements Open access funding provided by Budapest University of Technology and Economics (BME). The author thanks István Pilisi for carrying out all the measurements and his employer the National Media and Information Authority, Budapest, Hungary for providing the equipment. The author thanks Keiichi Shima, IIJ Innovation Institute, Tokyo, Japan for his help in map646 related issues and also for reviewing the manuscript of this paper.

References

1. 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*. https://doi.org/10.17487/RFC6147.
2. Bagnulo, M., Matthews, P., & Beijnum, I. (2011). Stateful NAT64: network address and protocol translation from IPv6 clients to IPv4 servers. *IETF RFC 6146*. https://doi.org/10.17487/RFC6146.
3. Palet J (2017). Using 464XLAT in residential networks. RIPE *74*, Budapest, Hungary, May 8–12, 2017, slides of presentation. https://ripe74.ripe.net/presentations/151-ripe-74-ipv6-464xlat-residential-v2.pdf. Accessed March 1, 2019.
4. Mawatari, M., Kawashima, M., & Byrne, C. (2013). 464XLAT: combination of stateful and stateless translation. *IETF RFC 6877*. https://doi.org/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, and T. Shan (Eds.), *Cloud computing and services science. (CLOSER 2012), Porto, Portugal, Apr. 2012, Communications in computer and information science* (Vol. 367, pp. 39–53). Springer. https://doi.org/10.1007/978-3-319-04519-1_3.
6. Bao, C., Li, X., Baker, F., Anderson, T., & Gont, F. (2016). IP/ICMP translation algorithm. *IETF RFC 7915*. https://doi.org/10.17487/RFC7915.
7. Free Software Foundation, The free software definition. http://www.gnu.org/philosophy/free-sw.en.html. Accessed March 1, 2019.
8. Open Source Initiative, The open source definition. http://opensource.org/docs/osd Accessed March 1, 2019.
9. Bradner, S., & McQuaid, J. (1999). Benchmarking methodology for network interconnect devices. *IETF RFC 2544*. https://doi.org/10.17487/RFC2544.
10. Popoviciu, C., Hamza, A., Van de Velde, G., & Dugatkin, D. (2018). IPv6 benchmarking methodology for network interconnect devices. *IETF RFC 5180*. https://doi.org/10.17487/RFC5180.
11. Georgescu, M., Pislaru, L., & Lencse, G. (2017). Benchmarking methodology for IPv6 transition technologies. *IETF RFC 8219*. https://doi.org/10.17487/RFC8219.
12. Lencse, G., & Kadobayashi, Y. (2019). Comprehensive survey of IPv6 transition technologies: A subjective classification for security analysis. *IEICE Transactions on Communications*. https://doi.org/10.1587/transcom.2018EBR0002.
13. Llanto, K. J. O., & Yu, W. E. S. (2012). Performance of NAT64 versus NAT44 in the context of IPv6 migration. *Proceedings of the International MultiConference of Engineers, 1,* 638–645.
14. Monte, C. P., Robles, M. I., Mercado, G., Taffernaberry, C., Orbiscay, M., Tobar, S., et al. (2012). Implementation and evaluation of protocols translating methods for IPv4 to IPv6 transition. *Journal of Computer Science and Technology, 12*(2), 64–70.
15. Yu, S., & Carpenter, B. E. (2012). Measuring IPv4–IPv6 translation techniques, Department of Computer Science, Univ. Auckland, Auckland, New Zeeland, Technical Report 2012-001, 2012. https://www.cs.auckland.ac.nz/~brian/IPv4-IPv6coexistenceTechnique-TR.pdf Accessed March 1, 2019.
16. Lencse, G., & Répás, S. (2016). Performance analysis and comparison of four DNS64 implementations under different free operating

systems". *Telecommunication Systems, 63*(4), 557–577. https://doi.org/10.1007/s11235-016-0142-x.
17. Lencse, G., & Takács, G. (2012). Performance analysis of DNS64 and NAT64 solutions. *Infocommunications Journal, 4*(2), 29–36.
18. Lencse, G., & Répás, S. (2013). Performance analysis and comparison of the TAYGA and of the PF NAT64 implementations. In *Proceedings of the 36th international conference on telecommunications and signal processing*, Rome, Italy. https://doi.org/10.1109/tsp.2013.6613894.
19. Répás, S., Farnadi, P., & Lencse, G. (2014). Performance and stability analysis of free NAT64 implementations with different protocols. *Acta Technica Jaurinensis, 7*(4), 404–427. https://doi.org/10.14513/actatechjaur.v7.n4.340.
20. Quintero, A., Sans, F., & Gamess, E. (2016). Performance evaluation of IPv4/IPv6 transition mechanisms. *International Journal of Computer Network and Information Security, 2016*(2), 1–14. https://doi.org/10.5815/ijcnis.2016.02.01.
21. Bálint, P. (2017). Test software design and implementation for benchmarking of stateless IPv4/IPv6 translation implementations. In *Proceedings of the 40th international conference on telecommunications and signal processing* (*TSP 2017*) (pp. 74–78), Barcelona, Spain, Jul. 5–7. https://doi.org/10.1109/tsp.2017.8075940.
22. Scholz, D. (2014). A look at Intel's dataplane development kit. In *Proceedings seminars future internet (FI) and innovative internet technologies and mobile communications (IITM)*, Munich, Germany (pp. 115–122). https://doi.org/10.2313/net-2014-08-1_15.
23. Nathan Lutchansky, TAYGA: Simple, no-fuss NAT64 for Linux. http://www.litech.org/tayga/ Accessed March 1, 2019.
24. NIC Mexico, Jool: SIIT and NAT64. http://www.jool.mx/en/about.html. Accessed March 1, 2019.
25. K. Shima, map646, https://github.com/keiichishima/map646. Accessed March 1, 2019.
26. Anderson, T, & Potter, A. L. Explicit address mappings for stateless IP/ICMP Translatio. IETF RFC 7757. https://doi.org/10.17487/rfc7757.
27. Bao, C., Huitema, C., Bagnulo, M., Boucadair, M., & Li, X. (2010). IPv6 addressing of IPv4/IPv6 translators. *IETF RFC 6052*. https://doi.org/10.17487/RFC6052.
28. Lencse, G., Georgescu, M., & Kadobayashi, Y. (2017). Benchmarking methodology for DNS64 servers. *Computer Communications, 109*(1), 162–175. https://doi.org/10.1016/j.comcom.2017.06.004.
29. Lencse, G., & Shima, K. (2020). Performance analysis of SIIT implementations: Testing and improving the methodology. *Computer Communications*, *156*(1), 54–67. https://doi.org/10.1016/j.comcom.2020.03.034.

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 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, parallel discrete event simulation methodology and IPv6 transition methods.