Received: 9 February 2022 | Revised: 9 July 2022 | Accepted: 10 September 2022

DOI: 10.1002/dac.5354

RESEARCH ARTICLE

WILEY

Towards the scalability comparison of the Jool implementation of the 464XLAT and of the MAP-T IPv4aaS technologies

Gábor Lencse | **Norbert Nagy**

Department of Networked Systems and Services, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, Budapest, Hungary

Correspondence
Gábor Lencse, Department of Networked Systems and Services, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, 1111 Budapest, Hungary.
Email: lencse@hit.bme.hu

Summary

464XLAT and MAP-T are both IPv4-as-a-Service technologies aiming to provide IPv4 service to the customers, while ISPs use only IPv6 in their access and core networks. The ISPs need to consider several different aspects, like scalability, compatibility with the applications, and efficiency of public IPv4 address sharing, when selecting the most appropriate IPv4aaS technology for their specific application scenario. The aim of our current effort is to make an initial step towards the comparison of the scalability of the 464XLAT and of the MAP-T IPv4aaS technologies. We point out the gap between the most desirable measurement method and the available testing tools. We measure the scale-up of the performance of the Jool implementation of 464XLAT and MAP-T as a function of the number of active CPU cores up to 16 cores using an available testing tool called dns64perf++. We fully disclose our detailed measurement setup and our results, as well as the analysis of our results pointing out the bottleneck during the measurements. We also outline the directions of our future research

KEYWORDS

464XLAT, IPv4aaS, Jool, MAP-T, scalability

1 | INTRODUCTION

The IETF (Internet Engineering Task Force) has standardized several *IPv6 transition technologies* to support the transition from IPv4 to IPv6 in various communication scenarios.[1] Among them, the *IPv4aaS* (IPv4-as-a-Service) technologies aim to support the scenario, when ISPs (Internet service providers) use only IPv6 in their access and core networks (e.g., to reduce their costs), while they also facilitate the usage of IPv4 for their customers. RFC 8585[2] recommends that CE (customer edge) IPv6 transition routers should support all the following five IPv4aaS technologies: 464XLAT (combination of stateful and stateless translation), DS-Lite (dual-stack lite), lw4o6 (lightweight 4over6), MAP-E (mapping of address and port with encapsulation), and MAP-T (mapping of address and port with translation). Thus, network operators may select the technology that they find the most appropriate for their specific application scenario. This selection

is an important decision, because each technology has its own advantages and disadvantages, as discussed by Lencse et al.[3] The technologies have several differences and similarities regarding different aspects.

As for the solution used for ISP network traversal, they can be categorized into two groups:

- 464XLAT and MAP-T use double translation (first NAT46 from IPv4 to IPv6 and then NAT64 from IPv6 to IPv4).
- DS-Lite, lw4o6, and MAP-E use 4-in-6 encapsulation (IPv4 packets are encapsulated into IPv6 packets and then they are de-encapsulated).

They are also different regarding their property if their operation is stateful or stateless in the operator's network:

- 464XLAT and DS-Lite are stateful.
- lw4o6 MAP-E and MAP-T are stateless.

We note that the latter are stateful at the customer edge, but it is considered less problematic regarding scalability. Several people believe that stateful operation in the service provider network results in poor scalability, whereas others believe that a good implementation can efficiently mitigate the problem of scalability (e.g., by hashing).[3] Therefore, their scalability is also an important aspect.

The aim of our current effort is to make the first step towards the comparison of the scalability of the 464XLAT[4] and of the MAP-T[5] IPv4aaS technologies. To that end, we measure, how the performance of the Jool open source 464XLAT and MAP-T implementation scales up with the number of active CPU cores.

The rest of this paper is organized as follows. In Section 2, we give an introduction to 464XLAT and MAP-T. In Section 3, we overview the possible test setups for the scalability measurements elaborating their advantages and disadvantages and select the one that currently can be used, then disclose our detailed measurement setup and our results, as well as the analysis of our results pointing out the bottleneck during the measurements, and we discuss the results. Finally, we outline the directions of our future research in Section 4.

2 | SHORT INTRODUCTION TO 464XLAT AND MAP-T

2.1 | Introduction to the operation of 464XLAT

The architecture of 464XLAT[4] is somewhat different in its wireline and wireless scenarios. We follow the wireline one and explain the operation of 464XLAT on the basis of Figure 1. Regarding the traffic of an IPv4 client, the *CLAT* (customer-side translator) performs a stateless NAT46, and only IPv6 packets travel in the access and core network of the ISP. The traffic destined to an IPv4 server is routed to the *PLAT* (provider-side translator), which performs a stateful

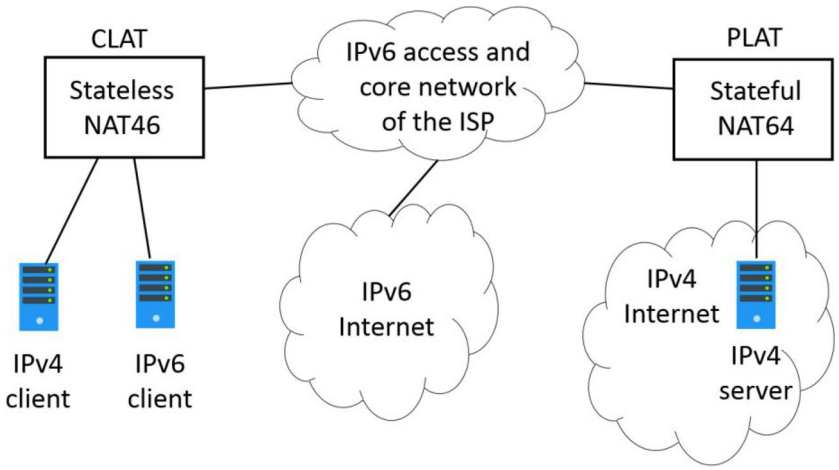

FIGURE 1 The architecture of 464XLAT

NAT64, and the IPv4 packets reach the IPv4 server. Regarding the traffic of an IPv6 client, the CLAT device acts as an IPv6 router. In the wireless scenario, the CLAT is implemented by the wireless device, for example, smartphone.

Let us see how the translation of the packets of the IPv4 applications works. To that end, we extend the example of RFC 6877[4] with port numbers to gain more insight into how translations happen. Let us suppose that the IPv4 client has the 192.168.1.2 IPv4 address and it communicates with the IPv4 web server at 198.51.100.1 IPv4 address. So, the IPv4 packet sent by the client has the following addresses and port numbers:

src IPv4: 192.168.1.2, src port: 50000
dst IPv4: 198.51.100.1, dst port: 80

For the translation of the source IP address and the destination IP address, the CLAT uses its client side IPv6 prefix (2001:db8:aaaa::/96) and the PLAT IPv6 prefix (2001:db8:1234::/96), respectively. Thus, after the stateless NAT64 translation (also called SIIT[6]), the IPv6 packet has the following addresses:

src IPv6: 2001:db8:aaa::192.168.1.2, src port: 50000
dst IPv6: 2001:db8:1234::198.51.100.1, dst port: 80

The PLAT uses the same prefixes, plus it has a public IPv4 address pool, let it be now: 192.0.2.1–192.0.2.100. The PLAT performs the stateful NAT64 translation and registers the five-tuple (source IP address, source port number, destination IP address, destination port number, and protocol number) into its connection tracking table. The IPv4 packet sent out by PLAT has the following addresses:

src IPv4: 192.0.2.1, src port: 10000
dst IPv4: 198.51.100.1, dst port: 80

The IPv4 only server receives the packet and sends the reply to the public IPv4 address of the PLAT. The PLAT finds the appropriate five-tuple in its connection tracking table, performs the stateful NAT64 translation in the reverse direction, and sends the IPv6 packet to the CLAT. The CLAT performs the stateless NAT46 translation in the reverse direction by using the last 32 bits of the IPv6 addresses as the IPv4 addresses. And finally, the reply packet from the IPv4 server reaches the IPv4 client. This is a simple and straightforward process, especially compared to the complexity of the operation of MAP-T.

We note that although we replaced the source port number with a new one in the above example, the modern NAT64 implementations follow the extended algorithm, which replaces the source port number with a new one only in the case, if keeping the original one would result in a clash with an already existing five-tuple in the connection tracking table that belongs to a different connection (please see an example in table 2 of our paper[7]). It also means that 464XLAT can use the source port numbers really sparingly and thus, it can share the public IPv4 addresses among the subscribers very efficiently. The price of this is the stateful operation of the PLAT.

2.2 | Introduction to the operation of MAP-T

We explain the operation of MAP-T on the basis of Figure 2, which looks very similar to Figure 1. It is so because MAP-T solves the same problem as 464XLAT; however, MAP-T uses just the "opposite" approach for IPv4 address sharing than 464XLAT. MAP-T divides the source port number range of a public IPv4 address into fixed size parts, called as port sets, and it assigns a specific port set to each subscriber. Of course, applications are not aware of the port sets, and they may use any source port numbers from the dynamic port range.[8] When an IPv4 client sends a packet, the CE router first performs a stateful NAT44 to transform the source port number into the port set assigned to the given CE. Then, the CE performs a stateless NAT46, during which the source port number is algorithmically mapped into the source IPv6 address. This is the "trick" of MAP-T: It thus eliminates the "port routing" of the A + P (address plus port) approach,[9] and normal IPv6 routing is performed in the operator's network. As each CE has its own port set, the *BR* (border relay) router can do stateless NAT64 using an algorithmic mapping between IPv6 addresses and IPv4 addresses + port numbers.

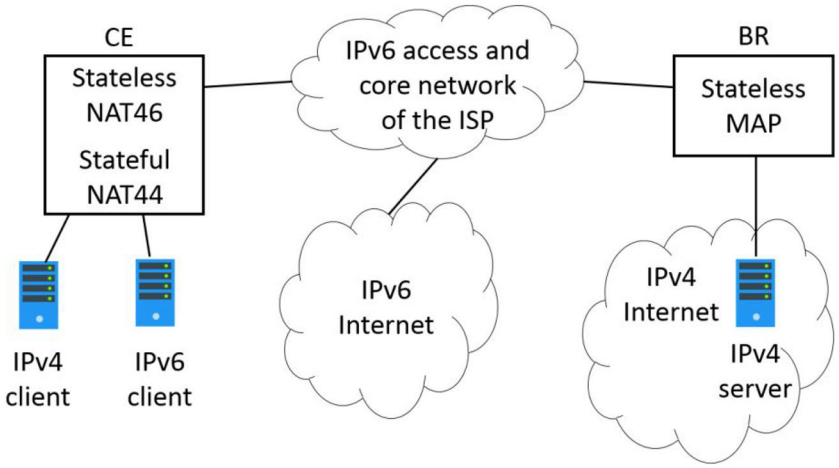

FIGURE 2 The architecture of MAP-T

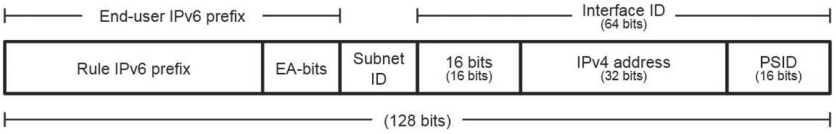

FIGURE 3 MAP address format[10]

On the one hand, the predefined size of port sets results in less efficient public IPv4 address sharing, as the possible maximum port number consumption of the applications is very different from their average port number consumption.[7] However, on the other hand, it also guarantees that a malicious user may not exhaust the source port number range (and thus the public IPv4 address pool) of the BR.

As the operation of MAP-T is quite complex, we limit our introduction to a simple example taken from Jool's MAP-T summary.[10] Let us suppose that we have 256 public IP addresses (192.0.2.0/24) and we choose 2048 as the size of the port sets. Thus, we can serve 256 * 65,536/2048 = 8192 customers. The *PSID* (port set identifier) is used to identify the port sets as follows:

PSID = 0 identifies port range 0–2047,
PSID = 1 identifies port range 2048–4095,

and so on, finally,

PSID = 31 identifies port range 63,488–65,535.

We note that in this simple approach, the entire source port range is used; the system ports[8] are also part of the very first port set.

In our example, the CEs are identified by the last 8 bits of the public IPv4 address plus the 5-bit long PSID. In MAP-T terminology, these 13 bits are called as *EA-bits* (embedded address bits). Now, let us choose the CE identified by the hexadecimal number "41." In the 8 + 5 = 13-bit long binary format, it looks like: 00000010|00001, where the decimal 2 is the IPv4 suffix and the decimal 1 is the PSID.

We need further information to build a MAP-T system. This is what type of IPv6 addresses may occur, and how they are handled.

The IPv6 addresses referring to hosts in the MAP domain are called MAP addresses, and they follow the MAP address format shown in Figure 3. For us, it is enough to deal with the end-user IPv6 prefix part of it. (The subnet ID is used only, when IPv4 addresses are not at all shared, rather each subscriber gets more than a single public IPv4 address. And the interface ID contains some redundant information, which is not needed for the operation of MAP-T.)

IPv6 addresses referring to outside of the MAP domain may be native IPv6 addresses or RFC 6052[11] compliant IPv4-embedded IPv6 addresses, which refer to some IPv4 hosts.

Formally, two triplets are defined. One of them is called as *BMR* (basic mapping rule), and it expresses "how to assemble MAP addresses out of IPv4 addresses and vice versa."[10] It contains the rule IPv6 prefix, the rule IPv4 prefix, and the length of the EA-bits. In our example, they are 2001.db8:ce::/51, 192.0.2.0/24 and 13, respectively.

The other "triplet" is the *DMR* (default mapping rule), which contains only a single valid field: an IPv6 prefix used to prepare the IPv4-embedded IPv6 prefix to represent the IPv4 hosts. In our example, it is the NAT64 well-known prefix (64:ff9b::/96).

Let the IPv4 client has the 192.168.0.4 IPv4 address, and let it communicate with the web server at 203.0.113.56 IPv4 address. Now, let us follow the translation of the IPv4 packet sent by the client:

src IPv4: 192.168.0.4, src port: 12345
dst IPv4: 203.0.113.56, dst port: 80

The CE first performs a stateful NAT44 to transform the source IPv4 address and source port number as required, that is, the source IPv4 address must be 192.0.2.2 and the source port number must be in the 2048–4095 range. Now, let it be 3000. Then, the CE uses the BMR to transform the source IPv4 address and port number into the source IPv6 address, and it simply prepares the IPv4-embedded IPv6 destination address using the prefix given in the DMR and the IPv4 destination address. Thus, the IPv6 packet sent out by the CE to the BR has the following addresses and port numbers:

src IPv6: 2001:db8:ce:41:0:c000:0202:1, src port: 3000
dst IPv6: 64:ff9b::203.0.113.56, dst port: 80

The packet arrives at the BR, because the 64:ff9b::/96 prefix is routed there. The BR uses the *FMR* (forwarding mapping rule) to transform the received IPv6 packet back to IPv4. In our case, the FMR is the same as the BMR, thus the IPv4 packet will have the following IPv4 addresses and port numbers:

src IPv4: 192.0.2.2, src port: 3000
dst IPv4: 203.0.113.56, dst port: 80

This packet arrives to the IPv4 web server, and its reply arrives back to the BR. Then, everything happens in the reverse direction as expected: The BR translates the source IPv4 address using the DMR into an IPv4 embedded IPv6 address. And it translates the destination IPv4 address into a MAP address using the FMR (now, the same as BMR). The packet arrives back to the proper CE, because the end-user IPv6 prefix is routed there. Now, the CE translates the destination IPv6 address according to the BMR and the source IPv6 address according to the DMR (that is, it uses the last 32 bits of the IPv6 address as the IPv4 address), and finally, the stateful NAT44 is also performed in the reverse direction.

For a more detailed explanation, please refer to the Jool MAP-T summary.[10]

3 | SCALABILITY MEASUREMENTS

3.1 | Method for scalability testing

Scalability may be examined at different levels. Now, we focus on the level, where the number of CPU cores is increased to achieve higher performance. We are aware that it may also be necessary to increase the number of devices at a coarser granularity level, but it is now beyond the scope of our investigations.

Now, we follow the same approach as we did when we measured the performance of different DNS64 server or authoritative DNS server implementations as a function of the number of active CPU cores in Lencse and Kadobayashi[12,13]; that is, we double the number of active CPU cores of the *DUT* (device under test) and execute the performance measurements using 1, 2, 4, 8, and 16 CPU cores.

As for the performance measurement method, RFC 8219[14] has defined the benchmarking methodology for IPv6 transition technologies. For double translation technologies, the dual DUT setup, shown in Figure 4, may be used. The advantage of this setup is that the two translators, CLAT and PLAT or CE and BR, may be benchmarked together; thus, no special tester is required (see below). The limitation of this setup is that it hides the potential asymmetric behavior: As the performance of the two devices is measured together, it is not revealed, which of them the actual bottleneck is. For this reason, RFC 8219 recommends benchmarking the devices also separately according to the single DUT setup, shown in Figure 5. However, this setup would require a special tester; for example, when benchmarking the BR of MAP-T, on the left side, the tester needs to send specially crafted IPv6 packets having MAP-T addresses, whereas on the right side, it needs to receive them as IPv4 packets (and it needs to do the opposite in the other direction as testing with bidirectional traffic is required by RFC 8219). As we do not have such special tester for MAP-T, the single DUT test setup was not a viable solution for us.

As for the testing tool, we have chosen `dns64perf++`, which was written as an RFC 8219 compliant benchmarking tool for DNS64 servers,[15] and it was successfully used for that purpose by Lencse and Kadobayashi[12] and later (after the increase of its performance by enabling it to use multiple sending and receiving threads), it was also suitable for benchmarking authoritative DNS servers.[13] Moreover, it was also used as an additional testing tool to measure the scale-up of the performance of the different examined SIIT implementations as a function of the number of active CPU cores.[16] In that paper, we used `dns64perf++` as a packet sending and receiving application together with an authoritative DNS server, the task of which was to send replies for the DNS queries. (The content of the replies was redundant; the DNS server was practically used as an "echo server.") The devices were calibrated so that DUT running the examined SIIT implementation be the bottleneck.

Now, we follow the same approach: Using the roles from Figures 1 and 2, the IPv4 client and the IPv4 server are used to execute the `dns64perf++` and the authoritative DNS server program, respectively. Their performance is set to the maximum by setting all CPU cores online. And the performance of the CLAT and PLAT (or CE and BR) devices is limited by the number of active CPU cores. (We clarify the details in the following sections.)

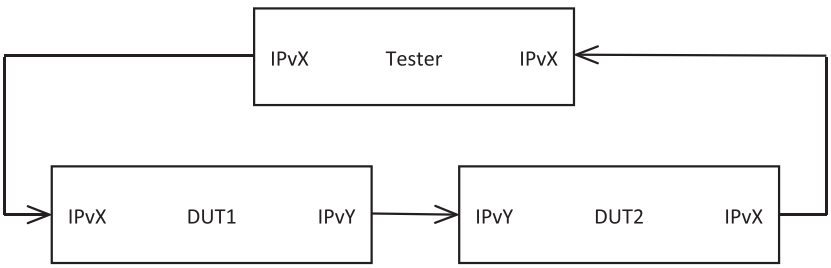

FIGURE 4 The dual DUT test setup of RFC 8219[14]

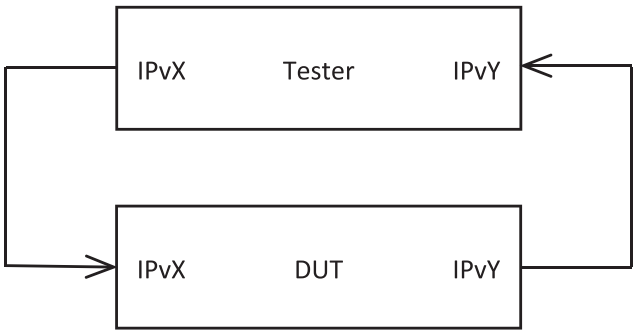

FIGURE 5 The single DUT test setup of RFC 8219[14]

3.2 | Building up the test system

The testbed for benchmarking measurements was built up using the resources of NICT StarBED, Japan. Four Dell PowerEdge R430 servers were used. As for their relevant parameters, each of them had two 16-core Intel Xeon E5-2683 CPUs, 384 GB 2400 MHz DDR4 RAM, and a dual port Intel X540 10 GbE NIC. Hyper-threading was switched off from the BIOS of the computers to achieve more stable measurement results.[13]

The Debian Linux operating system was upgraded to version 10.11 with kernel version 4.19.

The CPU clock frequency was set to fixed 2.1 GHz using the `tlp` package and the following command line in the `/etc/default/grub` file:

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet idle=poll"
```

The relevant settings of the/etc/default/tlp configuration file were:

```
CPU_SCALING_GOVERNOR_ON_AC=performance
CPU_SCALING_MIN_FREQ_ON_AC=2100000
CPU_SCALING_MAX_FREQ_ON_AC=2100000
```

The same test system was used for 464XLAT and MAP-T; thus first, we present their common settings, and then we give those settings that were different for the two variants for testing 464XLAT and MAP-T. In addition to that, first, we built a third variant of the test system for baseline measurements (please refer to Section 3.3), where IPv4 packet forwarding was enabled on the entire path between the IPv4 client and the IPv4 server as shown in Figure 6. The p097, p098, p099, and p100 computers were interconnected by putting their network interfaces into the proper VLANs using the "sbeasy" management tool provided us by NICT StarBED.

The network interface of the IPv4 client was set up, and a static route was set towards the IPv4 server using the following commands:

```
ip link set enp5s0f0 up
ip addr add 192.168.0.4/24 dev enp5s0f0
ip route add 203.0.113.0/24 via 192.168.0.1
```

The network interface of the IPv4 server was set up, and static routes were set towards the IPv4 client using the following commands:

```
ip link set enp5s0f0 up
ip addr add 203.0.113.56/24 dev enp5s0f0
```

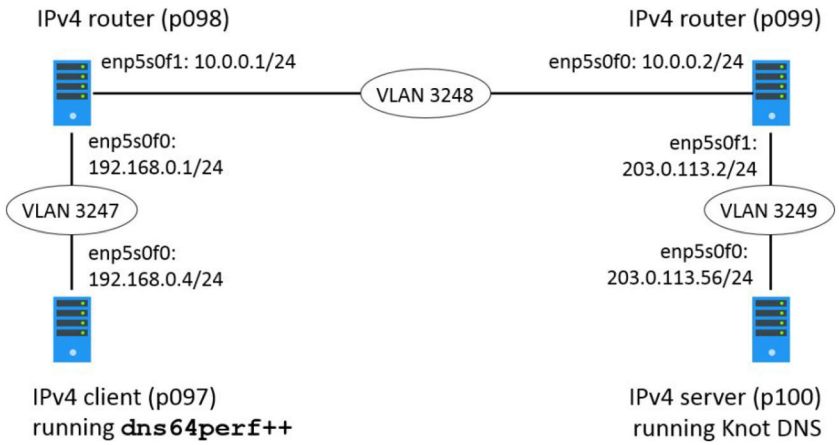

FIGURE 6 Test setup for baseline measurements

```
ip route add 192.168.0.0/24 via 203.0.113.2
ip route add 192.0.2.0/24 via 203.0.113.2
```

(The last line was used for the MAP-T measurements only.)

The "feature/multiport" version of the `dns64perf++` program was used for the measurements, including the further modifications described in section III.A of Lencse.[13] It was executed on the IPv4 client computer, and it used 16 threads for sending DNS queries and another 16 threads for receiving the replies.

Knot DNS version 2.7.6-2 was installed on the IPv4 server computer, and its most important settings were (in the `/etc/knot/knot.conf` file):

```
listen: [127.0.0.1@53, 203.0.113.56@53, ::@53]
- domain:"dns64perf.test."
 file:"/zonefiles/db.dns64perf.test-AAAA"
```

The zone file was created in the same way as described by Lencse.[13]

The IPv4 client and IPv4 server computers always used all 32 cores, whereas the number of the active cores of the two other computers was set by the "`maxcpus=X`" Linux kernel command line parameter to 1, 2, 4, 8, or 16.

Receive-side scaling (RSS) was enabled on all computers using the proper variant(s) of the commands, depending on the interface and IP version.

```
ethtool -N enp5s0f{0,1} rx-flow-hash udp{4,6} sdfn
```

We note that even with these settings, the interrupts caused by the arriving packets are not evenly distributed among all the 32 CPU cores, but the first 16 cores receive the lion's share of the interrupts, and the second 16 cores receive an order of magnitude less interrupts. (We consider that it can be caused by a kernel bug.) This phenomenon does not influence our results, because we used at most 16 cores on the DUT computers.

3.3 | Baseline measurements

The network interfaces of the p098 and p099 computers were set according to Figure 6; IPv4 packet forwarding was enabled on both computers, and IPv4 static routes were also set as needed.

As the measurement method is described by Lencse,[13] we used a binary search to determine the highest rate at which nearly all DNS queries are resolved within the 0.25 s timeout time. We mean under "nearly all" that 0.01% loss (that is one from every 10,000) was tolerated. Please see its rationale both in Lencse and Lencse et al.[13,17]

As the number of different source port numbers used by `dns64perf++` may influence, how evenly the interrupts caused by the arrival of the packets are distributed among all the CPU cores, we always used 100 different source port numbers per sending thread. (It means 1600 different source port numbers, which fitted into the limits of MAP-T.)

In order to check the performance of the measurement environment and also to provide a basis for comparison, we measured the "baseline" performance of the test system with 1, 2, 4, 8, and 16 CPU cores.

An elementary measurement in a single "step" of the binary search always lasted for 60 s, and we gave a 10 s "rest" for the system between two consecutive steps of the binary search. We executed the binary search 10 times to get a reliable number of measurement results.

TABLE 1 Throughput of the Linux kernel IPv4 packet forwarding (number of resolved queries per second)

Number of active CPU cores	1	2	4	8	16
Median (qps)	420,055	732,441	1,469,081	2,028,763	2,009,385
Minimum (qps)	416,256	722,655	1,445,306	2,011,487	1,981,189
Maximum (qps)	420,322	734,943	1,479,493	2,031,865	2,032,509
Proportion: current/previous	-	1.74	2.01	1.38	0.99

The results are shown in Table 1. We use the median of the 10 results for the evaluation; the minimum and maximum values are included only to show that the results are quite consistent.

The fact that the 2-core result is only 74% higher than the single core result can be explained by two things:

- The multicore operation has its cost.
- The two cores belong to different NUMA nodes.

We note that in this computer, the cores 0, 2, 4, ..., 30 belong to NUMA node 0, and the cores 1, 3, 5, ..., 31 belong to NUMA node 1.

Then the 4-core result is approximately the double of the 2-core result. It can be explained by the fact that the two cores of the 2-core system already belong to two different NUMA nodes, and no further issues appeared in the 4-core system that could negatively influence its performance.

The 8-core result and the 16-core result were deliberately limited by the performance of the authoritative DNS server running on the IPv4 server.

We note that a small performance decrease (in the order of the measurement error) can be observed at 16 cores compared to eight cores. We repeated the measurements multiple times, and it was always so. As it was not significant regarding our measurement, we did not investigate its root cause.

The performance of the baseline test system proved to be enough: It has never limited the performance of the 464XLAT or the MAP-T systems.

3.4 | 464XLAT test system and measurements

As Jool supports MAP-T from its 4.2 version, we used its latest version 4.2.0-rc2, where "rc" stands for "release candidate." We did so, because we wanted to use the same version of Jool for both 464XLAT and MAP-T to facilitate a fair comparison.

The network interfaces of the p098 and p099 computers were set according to Figure 7, and IPv4 and IPv6 packet forwarding was enabled on both computers.

As for the CLAT device, the stateless module of Jool (called as `jool_siit`) was used. During its setup, the 64:ff9b::/96 NAT64 well-known prefix was used as the PLAT prefix, and explicit address mapping (EAM) was used to translate the 192.168.0.4 IP address of the IPv4 client to an IPv6 address. Finally, the route towards the 64:ff9b::/96 network was the set via the PLAT. The CLAT device was set using the following commands:

```
ip link set enp5s0f0 up
ip addr add 192.168.0.1/24 dev enp5s0f0
ip link set enp5s0f1 up
ip addr add 2001:db8:100::1/64 dev enp5s0f1
```

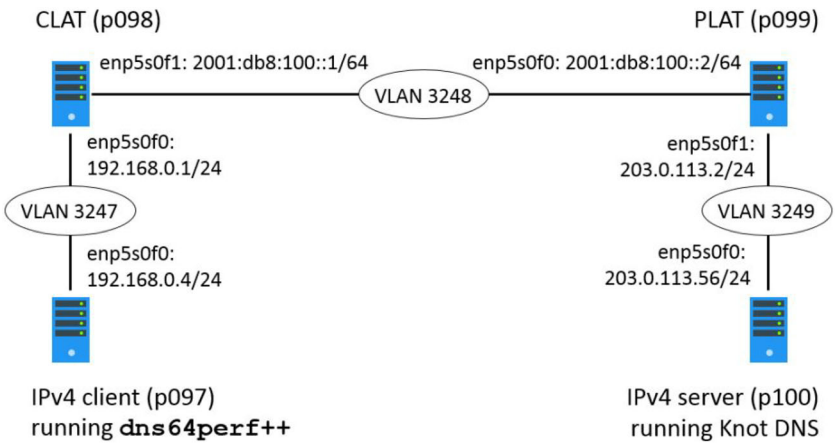

FIGURE 7 Test setup for 464XLAT

```
sysctl -w net.ipv4.conf.all.forwarding=1
sysctl -w net.ipv6.conf.all.forwarding=1
modprobe jool_siit
jool_siit instance add --netfilter --pool6 64:ff9b::/96
jool_siit eamt add 192.168.0.4 2001:db8:2::4
ip route add 64:ff9b::/96 via 2001:db8:100::2
```

As for the PLAT device, the stateful module of Jool (called as jool) was used. During its setup, the 64:ff9b::/96 NAT64 well-known prefix was used as the PLAT prefix, and the 203.0.113.2 source IP address was used for the stateful translation allowing Jool to use the entire 0–65,535 source port range. Finally, the route towards the network of the client was the set via the CLAT. The PLAT device was set using the following commands:

```
ip link set enp5s0f0 up
ip addr add 2001:db8:100::2/64 dev enp5s0f0
ip link set enp5s0f1 up
ip addr add 203.0.113.2/24 dev enp5s0f1
sysctl -w net.ipv4.conf.all.forwarding=1
sysctl -w net-ipv6.conf.all.forwarding=1
modprobe jool
jool instance add --netfilter --pool6 64:ff9b::/96
jool pool4 add 203.0.113.2 --udp 0-65535
ip route add 2001:db8:2::/64 via 2001:db8:100::1
```

The measurements were executed using 1, 2, 4, 8, and 16 cores at both the CLAT and the PLAT. The results are shown in Table 2. Our experience with the 1-2-4-core "baseline" results gives the explanation of the phenomenon that whereas the 2-core performance of 464XLAT is only 43% higher than its single core performance, its 4-core performance shows 79% increase compared to the 2-core performance: This is caused by the multicore operation and the NUMA architecture of the CPU.

As for the 8-core and 16-core results, the performance shows saturation: The performance increase drops from 20% to 4%. Thus, Jool shows rather limited scalability regarding the number of CPU cores.

As we mentioned before, the dual DUT setup hides the potential asymmetry of two DUTs. To find the bottleneck, we repeated the experiments at the median rates, and we measured the CPU utilization of the CLAT and PLAT. Technically, we used the `dstat` command to measure the idle time of the CPUs. And we performed the measurements 100 times and calculated the average for each second. We present the results only for the two extreme cases: with a single core and with 16 cores.

As for the single core system, the idle times of the CLAT and PLAT are shown in Figure 8 and in Figure 9, respectively. It is well visible that the bottleneck was the PLAT: It fully utilized the CPU capacity at the beginning of the 60 s of the test, while the CLAT had about 20% idle time.

As for the 16-core system, idle times of the CLAT and PLAT are shown in Figure 10 and in Figure 11, respectively. Now, the CLAT had more than 67% idle time, while the idle time of the PLAT cores was below 10% (and it showed some difference from core to core).

TABLE 2 Throughput of the Jool 464XLAT implementation (number of resolved queries per second)

Number of active CPU cores	1	2	4	8	16
Median (qps)	165,403	236,869	425,021	510,155	530,064
Minimum (qps)	163,280	236,185	420,311	499,999	529,881
Maximum (qps)	167,041	237,553	425,782	510,321	530,222
Proportion: current/previous	-	1.43	1.79	1.20	1.04

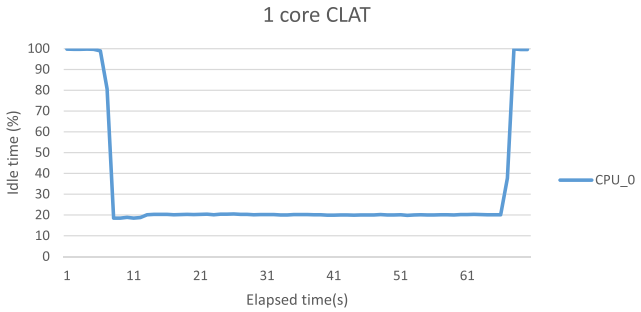

FIGURE 8 CPU idle time, 1 core, CLAT of 464XLAT

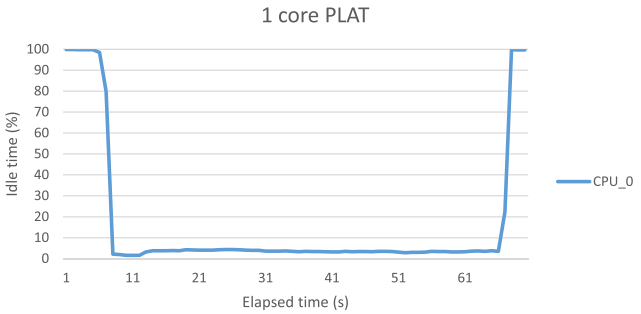

FIGURE 9 CPU idle time, 1 core, PLAT of 464XLAT

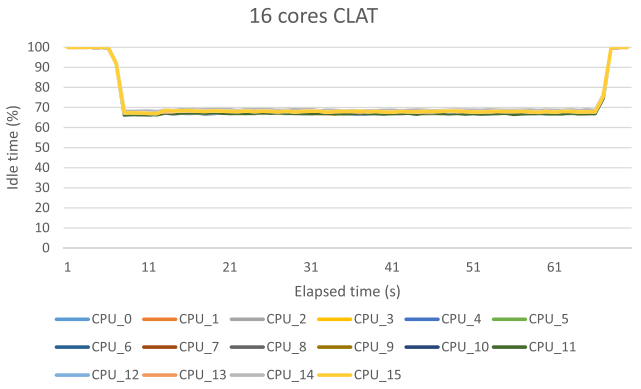

FIGURE 10 CPU idle time, 16 cores, CLAT of 464XLAT

3.5 | MAP-T test system and measurements

As we have mentioned before, Jool 4.2.0-rc2 was used as the MAP-T implementation. We followed the sample system that can be found in its documentation page.[10] We also followed the general practice of assigning 2048 source ports per user. It also means the 65,536/2048 = 32 users can share a single public IPv4 address, and thus the PSID has 5 bits. The 192.0.2.0/24 network was chosen as public IPv4 address pool of the BR. Thus, the size of the IPv4 suffix is 8 bits, and the number of EA bits is 8 + 5 = 13. From among the 8192 CEs, we have chosen the one that has the 192.0.2.8 IPv4 address and its PSID is the decimal 27, which denotes the port range of 55,296–57,343. Let us calculate its EA bits as the concatenation of the binary forms of 8 and 27:

"00001000" + "11011" = "0 0001 0001 1011," which is "11B" in hexadecimal format. This is the ending of the IPv6 address of the CE in Figure 12.

Due to the technical reasons explained in Jool,[10] we had to use namespaces in order to be able to implement both a stateful NAT44 and a stateless NAT64 by the same devices under Linux. We used the network namespace named "napt," and the enp5s0f0 was put into it. This namespace and the global namespace were connected by virtual Ethernet interfaces (`to_global` and `to_napt`) as shown in Figure 13.

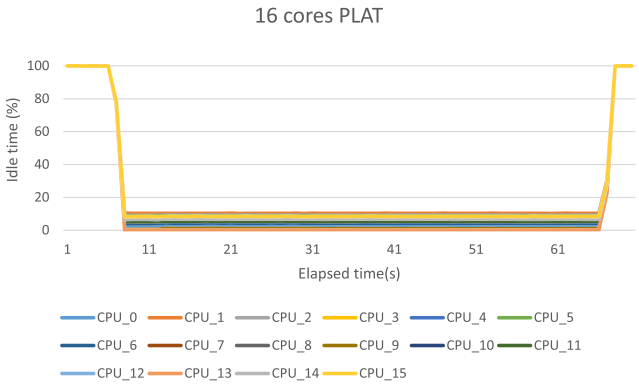

FIGURE 11 CPU idle time, 16 cores, PLAT of 464XLAT

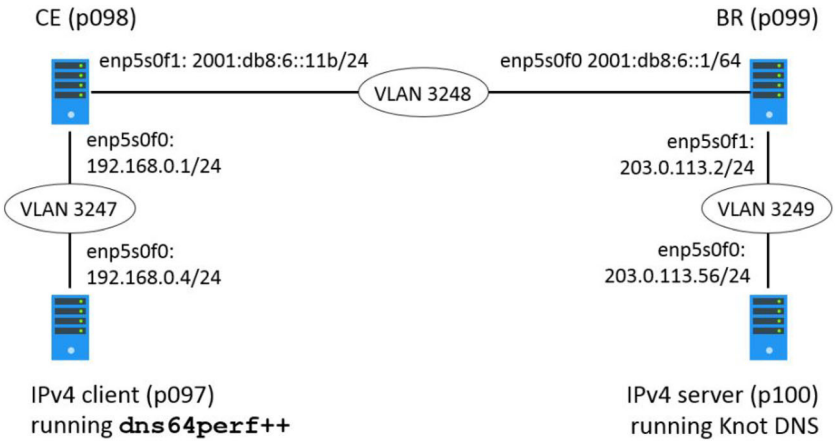

FIGURE 12 Test setup for MAP-T

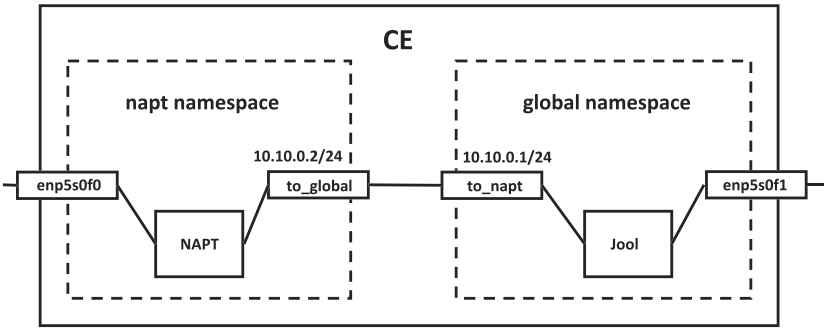

FIGURE 13 Implementation of the CE device using namespaces

The new namespace was created, and the two namespaces were connected by the next two commands:

```
ip netns add napt
ip link add to_napt type veth peer name to_global netns napt
```

The following commands were used to set up everything in the left side block of the CE:

```
ip link set enp5s0f0 netns napt
ip netns exec napt ip link set enp5s0f0 up
ip netns exec napt ip address add 192.168.0.1/24 dev enp5s0f0
```

```
ip netns exec napt ip link set tp_global up
ip netns exec napt ip address add 10.0.0.2/24 dev to_global
ip netns exec napt ip route add default via 10.0.0.1
ip netns exec napt sysctl -w net.ipv4.conf.all.forwarding=1
ip netns exec napt iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -o to_global -p udp -j SNAT \
 --to-source 192.0.2.8:55296-57343
```

And the following commands were used to set up everything except Jool in the right side block of the CE:

```
ip link set to_napt up
ip address add 10.0.0.1/24 dev to_napt

ip link set enp5s0f1 up
ip address add 2001:db8:6::11b/64 dev enp5s0f1
ip route add 64:ff9b::/96 via 2001:db8:6::1
ip route add 192.0.2.8/32 via 10.0.0.2

sysctl -w net.ipv4.conf.all.forwarding=1
sysctl -w net.ipv6.conf.all.forwarding=1
```

As for the configuration of Jool, first, its kernel module was loaded; second, the DMR containing the 64:ff9b::/96 prefix was set; third, the end-user IPv6 address was given; and fourth, the BMR was specified. (It contains the IPv6 prefix, the IPv4 prefix, the number of PSID bits, and finally a 0 showing that the well-known port numbers are also included in the very first port set.) Finally, the fact that this is a CE (and not a BR) was disclosed to Jool. The following commands were used:

```
modprobe jool_mapt
jool_mapt instance add "CE 11b" --netfilter --dmr 64:ff9b::/96
jool_mapt -i "CE 11b" global update end-user-ipv6-prefix 2001:db8:ce:11b::/64
jool_mapt -i "CE 11b" global update bmr 2001:db8:ce::/51 192.0.2.0/24 13 0
jool_mapt -i "CE 11b" global update map-t-type CE
```

The setup of the BR was signficantly simpler, because the BR performs only a stateless NAT64 specified by the FMR, which is currently the same as the DMR. (And its type is not set to BR, because it is the default value.) The following commands were used:

```
ip link set enp5s0f0 up
ip address add 2001:db8:6::1/64 dev enp5s0f0
ip link set enp5s0f1 up
ip address add 203.0.113.2/24 dev enp5s0f1
ip route add 2001:db8:ce:11b::/64 via 2001:db8:6::11b
sysctl -w net.ipv4.conf.all.forwarding=1
sysctl -w net.ipv6.conf.all.forwarding=1
```

The measurements were executed using 1, 2, 4, 8, and 16 cores at both the CE and the BR. The results are shown in Table 3. The same phenomenon can be observed as we have seen before: Whereas the 2-core performance is only 75% higher than the single core performance, the 4-core performance is 89% higher than the 2-core performance, but now the difference is significantly lower.

The overall scale-up is quite good, the increase of the throughput caused by doubling the number of the cores is at least 75% except for the last case, when it drops to 42%. So, we can definitely say that the Jool MAP-T implementation scales up better than the Jool 464XLAT implementation.

TABLE 3 Throughput of the Jool MAP-T implementation (number of resolved queries per second)

Number of active CPU cores	1	2	4	8	16
Median (qps)	143,498	250,858	475,051	835,180	1,183,615
Minimum (qps)	140,624	245,311	470,311	812,499	1,179,686
Maximum (qps)	145,314	251,953	475,818	837,295	1,184,387
Proportion: current/previous	-	1.75	1.89	1.76	1.42

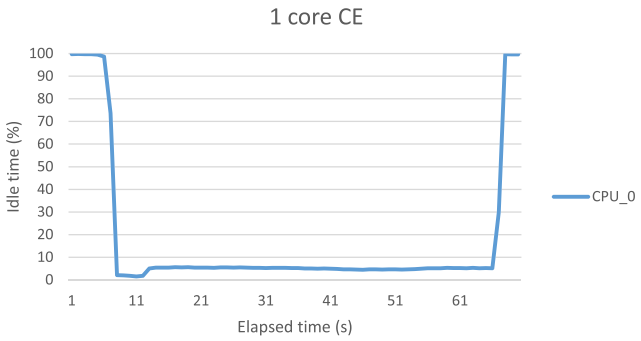

FIGURE 14 CPU idle time, 1 core, CE of MAP-T

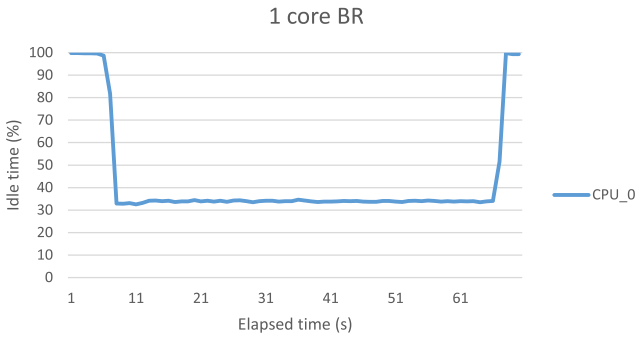

FIGURE 15 CPU idle time, 1 core, BR of MAP-T

Again, we wanted to figure out, which device was the bottleneck, therefore, we have checked the CPU utilization at the median rate for each number of active CPU cores.

As for the single core system, the idle times of the CE and BR are shown in Figure 14 and in Figure 15, respectively. It is well visible that the bottleneck was the CE: It has nearly fully utilized the CPU capacity (1.5%–2% idle time) at the beginning of the 60 s of the test, while the BR had more than 30% idle time.

As for the 16-core system, idle times of the CE and BR are shown in Figure 16 and in Figure 17, respectively. Now, the CE was practically fully utilized, while the BR had about 10% idle time (and it showed some slight difference from core to core).

3.6 | Discussion of the results

As for the raw numbers, MAP-T has definitely scaled up better than 464XLAT as shown in Figure 18. If we look behind the numbers, we can say that our results can be used only as *a lower bound of the performance of the BR of MAP-T*, as the bottleneck was the CE, and not the BR. However, ISPs are interested in the scalability of the device in their core network, that is, the BR. And as for the BR, its idle time was more than 30% at a single CPU core, whereas it was only

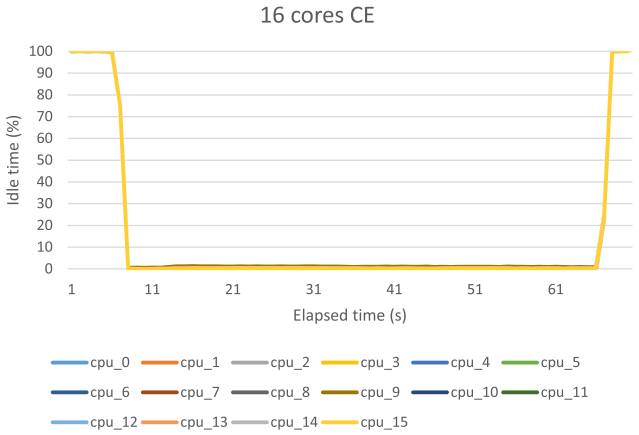

FIGURE 16 CPU idle time, 16 cores, CE of MAP-T

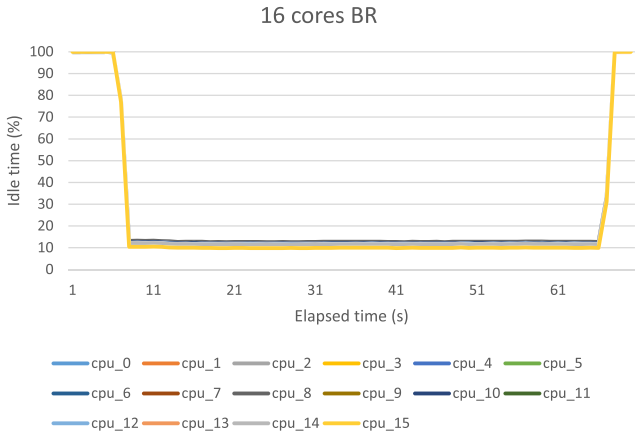

FIGURE 17 CPU idle time, 16 cores, BR of MAP-T

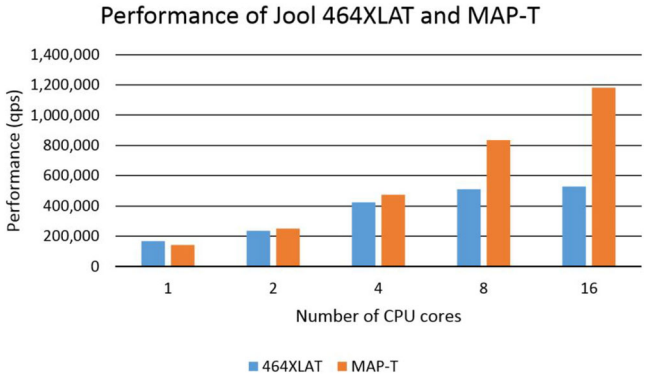

FIGURE 18 Performance of the Jool implementation of 464XLAT and MAP-T as a function of the active CPU cores measured according to the dual DUT setup, where the bottleneck was the PLAT of 464XLAT and the BR of MAP-T

about 10% at 16 cores. Thus, we expect that BR would show *higher performance*, but *somewhat less good scale-up* if a CE with higher performance were used.

We did not go into a deeper analysis of the CE, but we surmise that the scalability of CE was limited by the scalability of the stateful NAT44 translation and not that of the stateless NAT46 translation. If it is true, then we can say that we have compared the scalability of the Jool stateful NAT64 implementation (as the PLAT of 464XLAT) and the

scalability of the iptables stateful NAT44 implementation (as a part of the CE of MAP-T). As they are both stateful solutions, perhaps the experienced poor scalability belongs to Jool and not to all stateful technologies. Especially, because our measurements showed that the 16-core performance of iptables was more than 10-fold higher than its single core performance, please refer to figure 2 of Lencse.[18]

Finally, we would like to emphasize that even though our aim is to compare the scalability of the 464XLAT and of the MAP-T IPv4aaS technologies, what we did, was the comparison of the scalability of their Jool implementation. And what we can do in the future, will also be only the comparison of the scalability of certain implementations of the technologies. Whereas a good scale-up of a given implementation of a technology proves that the technology can be implemented in a scalable way, a poor scale-up of a given implementation does not prove the opposite about the technology itself.

4 | DIRECTIONS OF FUTURE RESEARCH

In the future, we plan to focus on measuring the scalability of the devices in the core of the ISP network, that is, the BR in the case of MAP-T. To that end, we plan to increase the performance of CE to make BR the bottleneck by

- implementing the two sub-functions of CE by two separate computers thus also eliminating the need for namespaces and virtual Ethernet devices
- using always all the CPU cores of the computers implementing the two sub-functions of CE.

Although currently the performance of `dns64perf++` and Knot DNS did not limit our measurements, our future measurements may require a tester with higher performance. Thus, we plan to use the stateful branch of `siitperf`,[19] which is currently in experimental state. Its latest version is documented by Lencse.[20] Its usage will make it possible to perform RFC 8219 compliant measurements like throughput, frame loss rate, latency, and PDV (packet delay variation), as well as the new, stateful technology specific metrics like maximum connection establishment rate or connection tear down rate defined in our Internet Draft.[21] Our new tool allows the user to specify the number of sessions (that is, the number of source IP address, source port number, and destination IP address destination port number combinations) to be used, thus it opens up the possibility to investigate the performance degradation caused by the increase of the number of concurrent sessions. As for 464XLAT, it can also be used for benchmarking PLAT according to the single DUT setup. As for MAP-T, its benchmarking will still be limited to the dual DUT setup.

Our long term plans include the implementation of a special-purpose tester for the BR of MAP-T, too.

We also plan to examine the scalability of a few other 464XLAT and MAP-T implementations, too.

5 | CONCLUSION

We have measured the throughput of the Jool implementation of both 464XLAT and MAP-T IPv4aaS technologies as a function of the number of CPU cores up to 16 cores. We have found that the Jool implementation of MAP-T scaled up better than the Jool implementation of 464XLAT. We have examined the CPU idle time to check the bottleneck in each case and found that it was the PLAT for 464XLAT and the CE for MAP-T, thus we have pointed out that regarding the performance of MAP-T, our results are a lower bound.

ACKNOWLEDGMENTS

The resources of NICT StarBED, Japan, were used for our measurement. The authors thank Shuuhei Takimoto for the opportunity to use the resources, as well as to Makoto Yoshida and Tsukasa Nishita for their support of StarBED usage-related issues.

The authors thank Keiichi Shima, Sándor Répás, Ádám Bazsó, and Omar D'yab for reading and commenting the manuscript.

DATA AVAILABILITY STATEMENT

Research data are not shared.

ORCID

Gábor Lencse https://orcid.org/0000-0001-5552-3237
Norbert Nagy https://orcid.org/0000-0003-2475-5697

REFERENCES

1. Lencse G, Kadobayashi Y. Comprehensive survey of IPv6 transition technologies: a subjective classification for security analysis. *IEICE Trans Commun*. 2019;E102-B(10):2021-2035. doi:10.1587/transcom.2018EBR0002
2. Palet Martinez J, Liu H. M.-H., Kawashima M. "Requirements for IPv6 customer edge routers to support IPv4-as-a-Service", IETF RFC 8585, May 2019, doi:10.17487/RFC8585
3. Lencse G, Palet Martinez J, Howard L, Patterson R, Farrer I. "Pros and cons of IPv6 transition technologies for IPv4aaS", May 23, 2022, Internet Draft, draft-ietf-v6ops-transition-comparison-04. Accessed July 9, 2022. https://datatracker.ietf.org/doc/html/draft-ietf-v6ops-transition-comparison
4. Mawatri M, Kawashima M, Byrne C. "464XLAT: combination of stateful and stateless translation", IETF RFC 6877, April 2013, doi:10.17487/RFC6877
5. Li X, Bao C, Dec W, Troan O, Matsushima S, Murakami T. "Mapping of address and port using translation (MAP-T)", IETF RFC 7599, July 2015, doi:10.17487/RFC7599
6. Bao C, Li X, Baker F, Anderson T, Gont F. "IP/ICMP translation algorithm" IETF RFC 7915, 2016, doi:10.17487/RFC7915
7. Lencse G. Estimation of the port number consumption of web browsing. *IEICE Trans Commun*. Aug. 2015;E98-B(8):1580-1588. doi:10.1587/transcom.E98.B.1580
8. Cotton M, Eggert L, Touch J, Westerlund M, Cheshire S. "Internet Assigned Numbers Authority (IANA) procedures for the management of the service name and transport protocol port number registry", IETF RFC 6335, Aug. 2011, doi:10.17487/RFC6335
9. R. Bush (Ed.), "The address plus port (A+P) approach to the IPv4 address shortage", IETF RFC 6346, doi:10.17487/RFC6346
10. Jool MAP-T summary, https://www.jool.mx/en/map-t.html
11. Bao C, Huitema C, Bagnulo M, Boucadair M, and Li X. "IPv6 addressing of IPv4/IPv6 translators," IETF RFC 6052, Oct. 2010, doi:10.17487/RFC6052
12. Lencse G, Kadobayashi Y. Benchmarking DNS64 implementations: theory and practice. *Comput Commun*. Sep. 2018;127:61-74. doi:10.1016/j.comcom.2018.05.005
13. Lencse G. Benchmarking authoritative DNS servers. *IEEE Access*. Jul. 2020;8:130224-130238. doi:10.1109/ACCESS.2020.3009141
14. Georgescu M, Pislaru L and Lencse G. "Benchmarking methodology for IPv6 transition technologies", IETF RFC 8219, Aug. 2017, doi:10.17487/RFC8219
15. Lencse G, Bakai D. Design and implementation of a test program for benchmarking DNS64 servers. *IEICE Trans Commun*. June 2017;E100-B(6):948-954. doi:10.1587/transcom.2016EBN0007
16. Lencse G, Shima K. Performance analysis of SIIT implementations: testing and improving the methodology. *Comput Commun*. Apr. 15, 2020;156:54-67. doi:10.1016/j.comcom.2020.03.034
17. Lencse G, Kovács Á, Shima K. Gaming with the throughput and the latency benchmarking measurement procedures of RFC 2544. *Int J Adv Telecommun Electrotech Signals Syst*. 2020;9(2):10-17. doi:10.11601/ijates.v9i2.288
18. Lencse G. "Scalability of IPv6 transition technologies for IPv4aaS", June 30, 2022, active Internet Draft, draft-lencse-v6ops-transition-scalability-03. Accessed July 9, 2022. https://datatracker.ietf.org/doc/html/draft-lencse-v6ops-transition-scalability
19. Lencse G. Design and implementation of a software tester for benchmarking stateless NAT64 gateways. *IEICE Trans Commun*. February 1, 2021;E104-B(2):128-140. doi:10.1587/transcom.2019EBN0010
20. Lencse G. Design and implementation of a software tester for benchmarking stateful NATxy gateways: theory and practice of extending siitperf for stateful tests. *Comput Commun*. August 1, 2022;172(1):75-88. doi:10.1016/j.comcom.2022.05.028
21. Lencse G, Shima K. "Benchmarking methodology for stateful NATxy gateways using RFC 4814 pseudorandom port numbers", June 30, 2022, active Internet Draft, draft-lencse-bmwg-benchmarking-stateful-04. Accessed July 9, 2022. https://datatracker.ietf.org/doc/html/draft-lencse-bmwg-benchmarking-stateful

How to cite this article: Lencse G, Nagy N. Towards the scalability comparison of the Jool implementation of the 464XLAT and of the MAP-T IPv4aaS technologies. *Int J Commun Syst*. 2022;e5354. doi:10.1002/dac.5354