Computer Communications 210 (2023) 256–272

Contents lists available at ScienceDirect

Computer Communications

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

Benchmarking methodology for stateful NAT64 gateways ☆

Gábor Lencse [a,*], Keiichi Shima [b], Kenjiro Cho [c]

[a] Department of Telecommunications, Széchenyi István University, 1 Egyetem tér, Győr, H-9026, Hungary
[b] SoftBank Corporation, 1-7-1 Kaigan, Minato-ku, Tokyo 105-7529, Japan
[c] Internet Initiative Japan, Iidabashi Grand Bloom, 2-10-2 Fujimi, Chiyoda-ku, Tokyo 102-0071, Japan

ARTICLE INFO

Index terms:
Benchmarking
Iptables
Jool
OpenBSD PF
Scalability
Stateful NAT64
Tayga

ABSTRACT

The benchmarking of Network Address and Protocol Translation from IPv6 clients to IPv4 servers (stateful NAT64) gateways is challenging from a methodological point of view because the state of the art benchmarking standards have some requirements that are conflicting when applied to stateful NAT64 gateways. In this paper, several methodological gaps are pointed out and a benchmarking methodology is proposed, which is applicable for any stateful NATxy gateways, where x and y are in {4, 6}. It bridges all the gaps by reconciling the conflicting requirements and facilitating the execution of the industry standard benchmarking measurement procedures (throughput, latency, frame loss rate, packet delay variation) with stateful NATxy gateways. New performance metrics specific to stateful testing are also defined: maximum connection establishment rate, connection tear down rate, and connection tracking table capacity. The proposed methodology is suitable for examining the scalability of the stateful NATxy gateways, too. The methodology is validated by applying it to the benchmarking of three radically different stateful NAT64 implementations: Jool, tayga plus iptables, and OpenBSD Packet Filter (PF). The details of the measurements and their results are fully disclosed.

1. Introduction

Stateful *Network Address and Protocol Translation from IPv6 clients to IPv4 servers* (stateful NAT64) [1] has played an important role in the stage of transitioning the Internet from *Internet Protocol version 4* (IPv4) to *Internet Protocol version 6* (IPv6) for about a decade [2]. Together with *Domain Name System (DNS) extensions for network address translation from IPv6 clients to IPv4 servers* (DNS64) [3] they enable IPv6-only clients to communicate with IPv4-only servers. In 2014 a survey on IPv4 address sharing mechanisms [4] stated that "the only actual address sharing mechanism that really pushes forward the transition to IPv6 is stateful NAT64". The *Combination of Stateful and Stateless Translation* (464XLAT) [5] was constructed to address the problem of IPv4-only applications and the usage of IPv4 literals; furthermore, the *Provider-side Translator* (PLAT) of 464XLAT is actually a stateful NAT64 gateway. Unfortunately, the transition to IPv6 happens rather slowly due to various root causes [6]; therefore, the authors expect that stateful NAT64 will be needed for decades.

Several stateful NAT64 implementations have been developed, and their performance and scalability are important decision factors for network operators when selecting the most appropriate one for their purposes. The comparison of their performance has been a research topic over the past ten years (please refer to the overview in Section 3). However, there are methodological gaps (please refer to Sections 2 and 4.2). Therefore, the published performance measurements could not comply with the state of the art benchmarking methodology defined by various *Internet Engineering Task Force* (IETF) *Benchmarking Working Group* (BMWG) *Request for Comments* (RFC) documents requiring testing with bidirectional traffic [7] and using pseudorandom port numbers [8].

It is a common property of any stateful NAT*xy* gateways, where *x* and *y* are in {4, 6}, and also of stateful firewalls that connections may only be initiated from the client side. Packets arriving in the opposite direction are discarded if they do not belong to a *connection*. The connections are registered using a *connection tracking table*, which leads to a *scalability issue* in that the performance of stateful devices degrades with an increase in the number of *network flows*.

The primary aim of this paper was to propose a methodology for benchmarking stateful gateways to measure how their throughput performance degrades with the number of network flows. The proposed method complies with the state of the art benchmarking practices and works with stateful gateways. The methodology was validated by checking if the proposed measurements can be carried out with radically different stateful NAT64 implementations and if they produced meaningful results. The authors also tried to find out if there were any

☆ This work was supported by the International Exchange Program of the National Institute of Information and Communications Technology (NICT), Japan.
* Corresponding author.
E-mail addresses: lencse@sze.hu (G. Lencse), keiichi.shima@g.softbank.co.jp (K. Shima), kjc@iijlab.net (K. Cho).

https://doi.org/10.1016/j.comcom.2023.08.009
Received 24 April 2023; Received in revised form 21 June 2023; Accepted 10 August 2023
Available online 14 August 2023

flaws in the methodology regarding its assumptions and how the tester could interact with the stateful gateway (e.g., the content of its entire connection tracking table could be deleted or a *User Datagram Protocol* (UDP) timeout could be set up).

The remainder of this paper is structured as follows: In Section 2, the problem statement is given. In Section 3, an overview of the research done on the performance analysis of stateful NAT64 implementations is provided and also their weaknesses are highlighted. In Section 4, the necessary background information regarding benchmarking are summarized, the different methodological gaps are pointed out, and a state of the art benchmarking methodology for stateful NATxy gateways is proposed. In Section 5, the available *free software* [9] for stateful NAT64 implementations are surveyed and three radically different ones are selected. In Section 6, the validation of the proposed benchmarking methodology by performing benchmarking measurements of the selected stateful NAT64 implementations is outlined. Sections 7–12 contain details of the benchmarking measurements and their results. In Section 13, the findings are discussed. This paper is summarized in the Conclusion section.

2. Problem statement

There is a long established and continually evolving industry standard methodology for benchmarking network interconnect devices, and its latest version [10] also includes stateful NAT64. (Please refer to Section 4.1 for a short introduction to it.) However, it needs to be mentioned that:

1. This methodology has some requirements that are partially contradictory regarding the benchmarking of stateful NAT64 gateways, and thus they cannot be implemented in their literal meaning: the usage of bidirectional traffic required by RFC 2544 [7] and the usage of pseudorandom port numbers required by RFC 4814 [8] do not work with stateful NAT64 gateways. (Please refer to Section 4.2.1 for details.)
2. This methodology lacks appropriate state-handling related performance metrics that are necessary to give a comprehensive characterization of the performance of stateful NAT64 gateways. (Please refer to Section 4.2.2 for details.)
3. This methodology does not provide sufficient guidelines on how to measure scalability, which is an extremely important property of software-based stateful NAT64 solutions regarding how their performance scales up with the increasing number of active CPU cores and potential degradation of their performance caused by the increasing number of network flows. (Please refer to Section 4.2.3 for details.)

The authors have already written an Internet Draft [11] proposing a methodology that reconciles the contradictory requirements for benchmarking stateful NAT64 gateways, defines the necessary new performance metrics, and is suitable for measuring the scalability of stateful NAT64 implementations. However, to follow up on this:

1. The methodology needs to be validated: it should be checked if the proposed measurements can be carried out with the different stateful NAT64 implementations, and if they produce meaningful and usable results that satisfactorily characterize the performance of the stateful NAT64 gateways.
2. It should be checked whether the requirements of the methodology are both necessary and satisfactory. (If not, then they should be updated.)

Please refer to Section 6 for details on how these goals were achieved.

3. Related work

This is a brief overview of the research results in the field of performance analysis of stateful NAT64 gateways, which also reveals the limitations.

RFC 6146 [1] defined stateful NAT64 in 2011. Several papers were published in the following years regarding the performance analysis of various stateful NAT64 implementations. The common aspect of the first three papers [12,13], and [14] from 2012 was that they measured together the performance of a given NAT64 implementation and a given DNS64 implementation, which was a serious drawback, as explained in [15].

A simple method suitable for separate performance analysis of different DNS64 and stateful NAT64 implementations was invented in 2012 [16], and the performance of tayga plus `iptables` and OpenBSD PF was compared using ICMP in 2013 [17], and using ICMP, TCP, and UDP in 2014 [18].

Barayuga and Yu compared various performance characteristics of three systems: stateful NAT44, stateful NAT64, and IPv6 using UDP in 2014 [19] and using TCP in 2015 [20]. They used Jool for stateful NAT64 implementation and `iperf` as a measurement tool.

Among other IPv4aaS (IPv4-as-a-Service) technologies, the performance of 464XLAT (of which the PLAT was a stateful NAT64 gateway) was measured by means of round-trip-delay, jitter, throughput, and packet loss using the D-ITG traffic generator in 2015 [21].

The authors of [22] compared the performance of various IPv6 transition solutions including the tayga plus `iptables` and the Jool stateful NAT64 solutions by measuring one-way delay and throughput (using `iperf` for the latter) in 2016.

All the above papers predated RFC 8219 [10] (published in 2017) and their measurement methods did not comply with it. Even later researchers could not follow the methodology described in it, due to lack of compliant Testers and the methodological gap regarding stateful NAT64, which is pointed out in Section 4.2.1.

As for the latest results, it is worth mentioning the M.Sc. thesis of Jan Pokorny [23] from 2019, which compared the performance of different stateful NAT64 implementations. Originally, he wanted to use `iperf` but its performance was not high enough, so he used PF_RING [24] as his measurement tool.

It is also worth mentioning two further papers from 2020, including not stateful but rather stateless NAT64 tests with a few RFC 8219 compliant measurements. In the first one [25], a legacy RFC 5180 [26] compliant commercial Tester was used with some trick to perform throughput tests utilizing the fact that the throughput measurement procedure did not change since RFC 2544 [7]. In the second one [27], throughput and frame loss rate measurements of three *Stateless IP/ICMP Translation* (SIIT) implementations were reported.

Paper [28] aimed to compare the scalability of the Jool implementation of 464XLAT and MAP-T [29], and thus it also contained stateful NAT64 measurements. Although the measurement setup only facilitated that the performance of the *Client-side Translator* (CLAT) and PLAT devices were measured together, it was pointed out that the bottleneck was the PLAT. These measurements did not comply with RFC 8219 due to the measurement tool used, but later the measurements were performed in another setup using a different measurement tool [30], and these measurements complied with the Dual *Device Under Test* (DUT) setup of RFC 8219.

4. Introduction of benchmarking methodology for stateful NAT64 gateways

4.1. Summary of the basics

RFC 2544 [7] was published in 1999 and since then it has determined how commercial network performance testers work. It laid down

all important conditions for the benchmarking measurements of network interconnect devices, including test setup, test frame formats and sizes, requirement for bidirectional traffic, and for IP packet forwarding devices, the stipulation of testing with a single IP address pair and also with 256 different destination networks. Furthermore, it specified the usage of UDP as a transport layer protocol. RFC 2544 defined the following benchmarking measurement procedures: *throughput*, *latency*, *frame loss rate*, *back-to-back frames*, *system recovery*, and *reset*. Moreover, it used IPv4 in its examples and the maximum frame rate values for various media presented in the Appendix also show its age.

An IPv6 upgrade was published in RFC 5180 [26] in 2008, which explicitly excluded IPv6 transition technologies from its scope. It also covered some contemporary media types, like *Ten Gigabit Ethernet* (10GbE).

The benchmarking methodology for IPv6 transition technologies was defined in RFC 8219 [10] in 2017. It focused mainly on the differences caused by the various IPv6 transition technologies. In order to handle the high number of IPv6 transition technologies [31] efficiently, it classified them into a small number of categories regarding the method they used for the traversal of the access and core network of the *Internet Service Provider* (ISP): *dual stack*, *single translation*, *double translation*, and *encapsulation*. Additionally, it defined the benchmarking methodology for the last three categories (as dual stack devices can be benchmarked according to RFC 2544 and RFC 5180). As for the benchmarking procedures, it reused the majority of them from RFC 2544, e.g., throughput, frame loss rate, etc., but it redefined the latency measurement procedure, and it defined further ones for *Packet Delay Variation* (PDV) and *Inter Packet Delay Variation* (IPDV). It kept the requirement of testing with bidirectional traffic, but it added the usage of unidirectional traffic as an optional test.

It needs to be mentioned that RFC 4814 [8] recommended the usage of pseudorandom source and destination port numbers. This is very important because RFC 2544 used a fixed *test frame format* including port numbers, which was very convenient for the manufacturers of testing devices; the same test frames could be sent out repeatedly. However, the modern multi-core packet forwarding devices (including routers implemented by Linux servers) usually support *Receive Side Scaling* (RSS), also called *multi-queue receiving*, and they need entropy to be able to distribute the interrupts caused by packet arrivals among all *Central Processing Unit* (CPU) cores. (Otherwise, when testing with a single IP address pair, two CPU cores would process all interrupts; one for each direction, which is far from the typical operation of a router that forwards Internet traffic.)

4.2. Methodological gaps

4.2.1. Requirements of bidirectional traffic and pseudorandom port numbers

Stateful NAT64 belongs to the single translation technologies among the categories of RFC 8219; therefore, it should be tested according to the *Single DUT Setup*. It means that the IPv6 and IPv4 interfaces of a stateful NAT64 gateway should be connected to the IPv6 and IPv4 interfaces of the Tester, respectively, and the Tester should send bidirectional traffic through the DUT. However, RFC 4814 has defined the following ranges for port numbers:

- Source port numbers: 1,024–65,535
- Destination port numbers: 1–49,151

Therefore, the number of possible source port number and destination port number combinations is: 3,170,829,312.

The verbatim application of the requirements of these two RFCs have different negative consequences depending on the direction:

1. In the IPv6 to IPv4 direction, the usage of potentially more than three billion source port number and destination port number combinations would exhaust the capacity of the connection tracking table of the DUT.
2. In the IPv4 to IPv6 direction, the usage of randomly invented port number combinations would result in the drop of the packets that do not belong to a connection stored in the connection tracking table of the DUT, thus, the vast majority of the test frames would be dropped.

It should be noted that the exhaustion of the capacity of the connection tracking table of the DUT may result in various problems depending on its replacement policy and settings. Just to mention a few:

- If the size of the connection tracking table of the DUT is not limited by a specific setting, then the DUT may collapse due to memory exhaustion. (Such a situation has been reported in Section 4.9 of [11].)
- If the size of the connection tracking table of the DUT is limited by a specific setting, then depending on its replacement policy, it either cannot store new connections (and it may also drop the packets) or it can overwrite its connections and due to this extra work the results will not reflect the operation of the DUT when it processes Internet traffic.

Therefore, careful considerations are required to bridge this methodological gap and to facilitate the benchmarking of stateful NAT64 gateways complying with both RFC 8219 and RFC 4814.

4.2.2. Missing state-handling performance metrics

Section 8 of RFC 8219 recommends two additional benchmarking tests for stateful IPv6 transition technologies; they are the *concurrent TCP connection capacity* and the *maximum TCP connection establishment rate measurements*. However, both of them only apply to TCP, whereas testing with UDP is required and a performance metric for connection tear down is completely missing. (For the authors' Internet Draft [11], testing with TCP is out of scope. Please see Table 1 for which document covers what.)

It should be noted that test frame format defined in RFC 2544 exclusively uses UDP (and not TCP) as a transport layer protocol. Testing with UDP was kept in both RFC 5180 and RFC 8219 regarding the standard benchmarking procedures (throughput, latency, frame loss rate, etc.). The proposed methodology of the authors follows this long established benchmarking tradition using UDP as a transport layer protocol, too. The rationale for this is that the standard benchmarking procedures require sending frames at arbitrary constant frame rates, which would violate the flow control and congestion control algorithms of the TCP protocol. TCP connection setup (using the three-way handshake) would further complicate testing.

4.2.3. Missing guidelines for scalability measurements

RFC 8219 and all its predecessors have the view that the performance of a *device* needs to be measured. It is called *Device Under Test*, and it is considered to be a black box. However, stateful NAT64 solutions are often implemented in software and they are not bound to a specific hardware, but they can be executed using a wide range of commodity servers. Thus, the benchmarking results produced by using a given specific hardware configuration do not provide enough information to predict their performance using a different hardware configuration. It would be more useful to know their performance using only a single CPU core of a widely used CPU type, and especially, *how their performance scales up with the number of available CPU cores*. Nevertheless, this topic was out of scope of RFC 8219 and its predecessors.

As for the performance degradation caused by multiple connections, it is partially addressed by Section 10 of RFC 8219. It mentions an *increased number of network flows*, but it does not specify how they are to be achieved. For example, it can be done by using multiple and an increasing number of source or destination IP addresses or source or destination port numbers.

Table 1
The coverage of measurement types and transport protocols.

Type of measurement	TCP	UDP
Maximum connection establishment rate	RFC 8219 [10]	Authors' Internet Draft [11]
Connection tracking table capacity	RFC 8219 [10]	Authors' Internet Draft [11]
Connection tear down rate	–	Authors' Internet Draft [11]

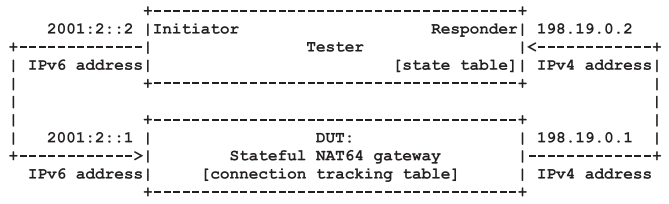

Fig. 1. Test setup for benchmarking stateful NAT64 gateways [11].

4.3. Proposed methodology

The first two authors of this paper proposed a general methodology for benchmarking stateful NAT*xy* gateways using RFC 4814 pseudorandom port numbers, where x and y were in {4, 6}. The first version of their individual Internet Draft was submitted in 2021 and it significantly evolved until its adoption by the BMWG of IETF at the IETF 115 meeting. The first version as a WG draft was published on September 24, 2022 [11]. This version is used to give a short summary of the proposed methodology and also as a reference regarding the further development and refinement of the methodology.

As for the IP version numbers, theoretically all four combinations of 4 and 6 are possible; practically, stateful NAT64 and stateful NAT44, also called *Network Address and Port Translation* (NAPT) have widespread deployment. In this paper, the focus is on stateful NAT64. (Here only a short summary of the methodology described in the 23-page long Internet Draft [11] is given.)

The test setup for benchmarking stateful NAT64 gateways is shown in Fig. 1. The IPv6 port of the Tester is called the *Initiator* and its IPv4 port is called the *Responder*. The Responder maintains an internal data structure, *state table*, of which the one that performs the measurement has full control. The DUT maintains an internal data structure, which is the *connection tracking table*. Its size, content, as well as its policies are unknown to one who performs the measurement, as *black box testing* is done. However, the one that performs the measurement can clear its contents (using some implementation dependent method), adjust its timeout time, and infer to its contents from its observable behavior.

As for the transport layer protocol, RFC 2544 recommends UDP, and it is kept in RFC 8219, which the authors also followed. It is important to note that stateful NAT64 gateways maintain state for both UDP and ICMP in their connection tracking tables, and not only for TCP.

The pseudorandom port numbers required by RFC 4814 are applied to the benchmarking of stateful NAT64 gateways as follows:

1. The Initiator uses *limited port number ranges*. The source port number range is larger (a few times 10,000) and the destination port number range is smaller and it is used as a parameter (e.g., from a few times ten to a few thousands). The rationale for this asymmetry is that the vast majority of the Internet traffic uses a wide range of source port numbers but only a few very popular destination port numbers according to [32].
2. Two *test phases* are used. (For historic reasons, *phase 1* and *phase 2* are called, in [11], *preliminary test phase* and *real test phase*, respectively.)
3. During *phase 1*, only the Initiator sends IPv6 test frames. The DUT performs the stateful NAT64 translation and stores the connections in its connection tracking table. The Responder receives the IPv4 test frames, extracts the *four tuples* (source IPv4 address, source port number, destination IPv4 address destination port number) from the received frames and stores them in its state table.
4. During *phase 2*, the Initiator and the DUT behave as before, and the Responder constructs and sends IPv4 test frames, taking valid four tuples from its state table (swapping source and destination). The Responder also updates its state table according to the information extracted from the newly received test frames.

Although the contents of the connection tracking table of the DUT cannot be directly examined, its contents can be controlled with the careful selection of certain parameters. To that end:

1. It is assumed that the connection tracking table of the DUT is able to store all connections generated by the possible source port number and destination port number combinations determined by their selected ranges. (It can be checked, please refer to the connection tracking table capacity measurement in Section 4.9 of [11].)
2. Only a single source and destination IP address pair is used. (This is a limitation, which is discussed in Section 13.2.4.)
3. Each experiment is started with an empty connection tracking table.
4. In the DUT, a timeout time is set that is larger than the sum of the length of phase 1, the gap between the two phases, and the length of phase 2.
5. During phase 1, all possible source port number and destination port number combinations are enumerated in a pseudorandom order.

By using the above-mentioned simple steps, the following can be achieved:

- During phase 1, all test frames result in creating a new connection in the connection tracking table of the DUT.
- During phase 2, neither new connections are created in the connection tracking table of the DUT, nor connections are deleted from there.

These clear situations make it possible to perform repeatable measurements. Phase 1 is suitable for the measurement of a new metric called *maximum connection establishment rate*. It is the highest (constant) frame rate at which the DUT is able to properly process all test frames in phase 1. In practice, a binary search is used to find the highest such rate. The maximum connection establishment rate measurement has two sub-variants:

1. The first one supposes that if a test frame arrived to the Receiver, then it had been properly processed by the DUT. (This one was the original version and it was retroactively named *non-validated* maximum connection establishment rate measurement.)
2. The second one checks in phase 2, if all connections have been established. To that end, the Responder sends test frames using all stored four tuples at a low enough rate and the Initiator checks if all frames arrived. It is called: *validated* maximum connection establishment rate measurement. Please refer to Section 11 for details.

The maximum connection establishment rate is an important metric for stateful NAT64 gateways, thus it should be reported. Additionally, it is needed for the other tests.

The "classic" measurement procedures (throughput, latency, frame loss rate, PDV, IPDV) can be executed in phase 2. In this case, phase 2 must be preceded by phase 1, during which the frame rate is safely lower than the previously measured maximum connection establishment rate to ensure that all required connections are created in the connection tracking table of the DUT.

The *connection tear down performance* is also an important metric for stateful NAT64 gateways because all established connections have to be terminated. To that end, an aggregate measurement method was recommended. In short, first N number of connections are loaded into the connection tracking table of the DUT and then the entire content of the connection tracking table is deleted using some out-of-band solution (e.g., by deleting the kernel module under Linux). The duration of the deletion (T) is then measured. The *connection tear down rate* is: N/T and needs to be measured for various values of N.

It is complicated to measure the *capacity of the connection tracking table* of stateful NAT64 gateways because the validation of the used validated maximum connection establishment rate measurement may fail for two different reasons that have different meanings:

- It may fail because the capacity of the connection tracking table has been exhausted.
- It may fail because the frame rate was too high.

Please refer to Section 4.9 of [11], for a more detailed analysis of the situation.

Here, only the highlights of the measurement method are given, which consists of two major steps:

1. The order of magnitude of the size of the connection tracking table is determined by an exponential search: Starting from a safe initial number of connections (that can be surely stored in the connection tracking table), the tested number is always doubled and it is checked by the validated maximum connection establishment rate measurement, if the currently tested number of connections can be stored in the connection tracking table. The result is the last safe number (N_s) and the number of connections the test failed at (N_f).
2. In general, a binary search is used to find the exact size of the connection tracking table where the starting interval of the binary search is [N_s, N_f].

The *performance metrics* can be classified into two groups:

1. "Stateful" metrics: connection establishment rate, connection tear down rate, connection tracking table capacity.
2. "Classic" metrics: throughput, latency, frame loss rate, PDV, IPDV.

For *scalability*, no new metrics were defined, but *measurement series* are recommended, through which the value of a parameter is changed. As for scalability against the number of CPU cores, testing with a series of active CPU core numbers is recommended that are powers of two, i.e., 1, 2, 4, 8, etc., in order to be able to scan a wide range while keeping the number of tests reasonable. As for scalability against the number of connections, it is recommended tuning the number of connections by increasing the size of the destination port number range.

4.4. Available measurement tool

As far as the authors know, `siitperf` is the only RFC 8219 and RFC 4814 compliant SIIT and stateful NAT64/NAT44 tester. It is available from GitHub [33] under the GPLv3 license. Its original version (documented in [34]) was only a SIIT tester and it literally followed the fixed test frame format, as defined in Appendix C.2.6.4 of RFC 2544. Later the RFC 4814 pseudorandom port numbers were enabled for use [35]. Eventually, it was extended for stateful benchmarking [36].

`Siitperf` was designed to be a flexible research tool allowing for the usage of several tunable parameters even if the RFCs recommended using constant values. Its core business logic was implemented in C/C++ programs using the Intel *Data Plane Development Kit* (DPDK) [37]. The binaries can be used to perform a single step of the measurements and they are executed by various bash shell scripts. The shell scripts are also provided, but they have to be customized depending on the actual measurement environment and intended parameters.

5. Survey of stateful NAT64 implementations to be benchmarked

For a long time the authors considered only *free software* [9] for benchmarking for the same reasons as presented in [38]:

- The licenses of certain vendors (e.g., [39,40]) do not allow the publication of benchmarking results.
- The free software can be used by anyone for any purpose, thus the results can be helpful for anyone.
- The (free) software is free of charge for the authors, too.

Therefore, the authors surveyed the available free software for stateful NAT64 implementations. They could not test all the available ones due to time limitations. The following criteria were set up for selecting implementations for testing:

1. The set of implementations to be tested should contain *diverse* implementations *to support the validation of the benchmarking methodology*.
2. The selected implementations should not be obsolete but *currently usable* ones in order *to provide* network operators and researchers with *useful information*.
3. The known *high performance* is an advantage, but not a requirement as criterion 1 has a higher priority than criterion 2.

The following implementations were selected for testing for the following reasons:

- Jool [41] has been widely used in the latest papers and it has a good performance.
- Tayga [42] is well-known, and it differs from the other candidates in the sense that it is a user-space stateless NAT64 translator and it is used together with `iptables` to implement stateful NAT64. Although it is old, it is a part of the Debian Linux distribution.
- OpenBSD PF [43] is very diverse from other candidates and it has shown good performance [17].

The authors were aware of the following implementations, but did not choose them for testing:

- FD.io VPP [44] has been mentioned in many research papers over the past five years. As it is DPDK-based, it was expected to have high performance, although the authors did not find any information published about its stateful NAT64 performance. Its version 22.06 was successfully installed. However, it always crashed with segmentation fault when its stateful NAT64 performance was tested. The case was reported on the vpp-dev@lists.fd.io mailing list [45] and the investigation of the problem is in progress. The authors plan to test it later (after a successful bugfix).
- Ecdysis [46] was one of the first stateful NAT64 implementations. Its code has been included into OpenBSD PF and it is still available as a Linux kernel module. In 2012 the authors tested it with different Linux kernels but it did not work [16]. Its latest release was in 2014 when it was updated for 3.13 Linux kernel, but it is highly unlikely to work with 4.x kernels.
- ASAMAP Vyatta [47] was used in [21]. Its latest version is from 2014, and it runs only with Linux 3.x kernels. Moreover, it was built on Debian 6 and did not recognize the RAID controller of the Dell PowerEdge R430 servers used for benchmarking.

Table 2
Throughput of IPv4 packet forwarding as a function of the number of the active CPU cores.

Number of active CPU cores	1	2	4	8	16	32
Error (fps)	400	1,000	2,000	4,000	8,000	8,000
Median (fps)	907,421	1,623,045	3,187,499	6,249,999	11,851,562	12,007,812
Minimum (fps)	901,952	1,592,772	3,091,796	6,121,092	11,617,186	11,992,186
Maximum (fps)	912,890	1,625,976	3,205,076	6,253,906	11,882,812	12,023,436
Average (fps)	907,811	1,619,920	3,173,045	6,230,468	11,831,249	12,007,811
Standard deviation	3,299	10,021	36,169	42,473	76,918	12,758
Median / previous median	–	1.79	1.96	1.96	1.90	1.01

- VyOS [48] can be considered a successor of Vyatta, but it does not actually have a stateful NAT64 implementation, there is only a "feature request" for it where one can state how important this feature would be [49].
- WrapSix [50] is about a 10-year-old user space stateful NAT64 implementation and Jan Pokorny did not manage to get it to work in 2019 [23].

6. Steps of evaluation

To validate the methodology, the authors tested if their proposed benchmarking measurements could be carried out with the three selected stateful NAT64 implementations and documented in detail to show that the measurements work with all of them, despite the fact that they have some radical differences. As there were a high number of possible measurements, some reductions were made and only the relevant ones were carried out. The authors considered the following aspects:

1. To test the feasibility of all measurement types, the following types of measurements were carried out with all three implementations (at least once):
    a. maximum connection establishment rate
    b. throughput (as a representative of the "classic" tests to be performed in phase 2)
    c. connection tear down rate
2. To test scalability, the authors performed:
    a. tests with a different number of active CPU cores where it was relevant
    b. with different number of connections for all implementations
3. To provide network operators with usable performance data, the authors did more measurements with scalable and high performance implementations.

Although RFC 8219 requires testing with bidirectional traffic and makes testing with unidirectional traffic optional, the authors performed a unidirectional test with all implementations because they gave an important insight into the operation of the given stateful NAT64 implementation:

- These results were used in Section 11.
- They were important for network operators because the download traffic of stateful NAT64 gateways is usually significantly higher than their upload traffic.

The authors also investigated if there is a justification for using the validated connection establishment rate as a replacement for the original non-validated one.

The authors verified if the order of port numbers makes a difference, if port number enumeration is used.

The connection tracking table capacity measurements were omitted because they currently do not work with some of the implementations. It is planned that this will be addressed in a separate work.

The results were carefully analyzed from the viewpoint of the methodology and the findings are discussed in Section 13.

7. The Linux test system and baseline measurements

With the exception of OpenBSD PF, the other two implementations were tested using the same Linux test system. Its description is in Appendix A.1.

The throughput of IPv4 and IPv6 packet forwarding of the Linux kernel using 1, 2, 4, 8, 16, and 32 active CPU cores was measured to check the performance of the test system and to give an insight into its scalability.

The tests were executed 10 times with each number of active CPU cores and IP versions.

It may be noted that `siitperf` interpreted the packet rate as "number of packets per second *per direction*" and it also reported the results with this interpretation. However, commercial network performance testers usually report packet rate as the number of all forwarded packets per second. Thus, the reported throughput results of `siitperf` were multiplied by two. This was also done with the *error of the binary search*. The error value expresses the stopping criterion for the binary search. It stops, when:

$$\text{higher\_limit} - \text{lower\_limit} <= \text{error}. \quad (1)$$

The value of error was selected to be less than 0.1% of the results so that the results would have three valuable digits.

As for the summarizing function, the Internet Draft [11] recommended the usage of median because it is less sensitive to the outliers than average.

To express the dispersion of the results, the authors recommended the first percentile and the 99th percentile, which are now the same as minimum and maximum, respectively, as the number of the results is less than 100.

The throughput results of the scalability test of Linux kernel IPv4 packet forwarding are shown in Table 2. In the last row, the proportion of the median and the previous median (measured with half as many cores) was also added.

It might be apparent that the increase of the throughput from a single core to two cores was only 79%, whereas it was 96% in the following two cases. This phenomenon can be explained by two root causes:

1. the multi-core operation has its cost compared to the single-core operation,
2. the measurement system was a *Non-Uniform Memory Access*[1] *(NUMA) inhomogeneous* system from two cores. (The Intel 10G dual-port X540 NIC and the CPU cores 0, 2, 4, ... 30 belonged to NUMA node 0 and the CPU cores 1, 3, 5, ... 31 belonged to NUMA node 1.)

There was an even more salient and strange phenomenon at 32 cores; the throughput showed only a negligible increase (1%). It was caused by an issue that the authors also observed earlier under Debian 9 with 4.9.0-16-amd64 Linux kernel: even if 32 CPU cores were online,

[1] It is a memory system design, where the memory access time depends on the location of the memory. A CPU can access its local memory faster than non-local memory. Please refer to [51] for full depth explanation.

Table 3
Throughput of IPv6 packet forwarding as a function of the number of the active CPU cores.

Number of active CPU cores	1	2	4	8	16	32
Error (fps)	400	1,000	2,000	4,000	4,000	4,000
Median (fps)	771,873	1,250,780	2,464,061	4,707,030	6,703,124	5,874,999
Minimum (fps)	752,732	1,199,218	2,423,436	4,621,092	6,652,342	5,761,718
Maximum (fps)	775,390	1,252,342	2,479,686	4,753,906	6,722,656	5,894,530
Average (fps)	768,670	1,242,968	2,461,874	4,704,686	6,699,999	5,860,155
Standard deviation	7,630	17,508	17,496	38,459	18,941	41,699
Median / previous median	–	1.62	1.97	1.91	1.42	0.88

Table 4
Maximum connection establishment rate of Jool as a function of the number of the active CPU cores, 4M connections.

Number of active CPU cores	1	2	4	8	16
Error (cps)	100	200	400	400	400
Median (cps)	208,984	331,835	420,653	454,345	483,153
Minimum (cps)	208,299	324,804	416,747	452,392	472,411
Maximum (cps)	213,768	337,695	423,095	457,275	486,572
Median / previous median	–	1.59	1.27	1.08	1.06

Table 5
Throughput of Jool as a function of the number of the active CPU cores, 4M connections, bidirectional traffic.

Number of active CPU cores	1	2	4	8	16
Error (fps)	200	200	400	400	400
Median (fps)	236,717	371,286	475,780	491,794	497,654
Minimum (fps)	234,178	368,162	473,044	487,108	495,702
Maximum (fps)	237,694	375,194	480,858	495,702	501,952
Median / previous median	–	1.57	1.28	1.03	1.01

the interrupts were scheduled only to cores 0–15. The authors believe that this was a kernel bug because according to `dmesg`, 32 receive queues are supported by the NICs (`Rx Queue count = 32`). Therefore, the authors decided to use only 1, 2, 4, 8, and 16 CPU cores for testing the scalability of the various stateful NAT64 implementations.

Note: Debian 11.2 with 5.10.0-11-amd64 kernel was also tested and an even worse issue was found: the interrupts were scheduled only to the *even* number CPU cores: 0, 2, 4, ... 30.

The results of the scalability test of Linux kernel IPv6 packet forwarding are shown in Table 3. The same tendencies can also be observed here and the situation is even worse at 32 cores as the throughput decreases by 12%.

Despite all these issues, the Linux test system was found to be scalable up to 16 CPU cores and its performance was more than enough for benchmarking the selected stateful NAT64 implementations.

8. Benchmarking Jool

There is a detailed description of the measurements with Jool in Appendix A.2.

8.1. Scalability against the number of active CPU cores

It was examined how the performance of Jool scaled up with the number of active CPU cores from 1 to 16. For these tests, four million connections were used, based on the suggestion of Vyacheslav Gapon, for a high-loaded NAT server [52]. Following the recommendation of [11], the source port number range was wider (1–40,000) and the destination port number range was narrower (1–100).

The maximum connection establishment rate measurement results are shown in Table 4. They show moderate scalability. The increase of performance was significant up to four CPU cores (59% and 27%), but it started vanishing from 8 CPU cores (8% and 6%).

The throughput measurement results using bidirectional traffic are shown in Table 5. They are very similar to those in the previous table, but here the performance increase is vanishing even more from 8 CPU cores (3% and 1%).

It can be stated that Jool has shown rather poor scalability with the number of active CPU cores. A 16-fold increase in the number of active CPU cores resulted in only 2.3- and 2.1-fold increase in the maximum connection establishment rate and throughput, respectively.

Table 6
Maximum connection establishment rate of Jool as a function of the number of connections, 16 CPU cores.

Number of connections	400,000	4,000,000	40,000,000
Source port numbers	40,000	40,000	40,000
Destination port numbers	10	100	1,000
Error (cps)	500	400	200
Median (cps)	625,976	483,153	356,445
Minimum (cps)	612,792	472,411	349,804
Maximum (cps)	630,370	486,572	358,397
Median / previous median	–	0.77	0.74

Table 7
Throughput of Jool as a function of the number of connections, 16 CPU cores.

Number of connections	400,000	4,000,000	40,000,000
Source port numbers	40,000	40,000	40,000
Destination port numbers	10	100	1,000
Error (fps)	400	400	200
Median (fps)	698,826	497,654	378,513
Minimum (fps)	684,764	495,702	373,240
Maximum (fps)	704,294	501,952	381,834
Median / previous median	–	0.71	0.76

8.2. Scalability against the number of connections

It was examined how the performance of Jool degraded with the number of connections. The number of active CPU cores was always 16 and the number of connections was increased from 400,000 to 40,000,000 in two steps. The maximum connection establishment rate and the throughput measurement results are shown in Table 6 and Table 7, respectively. The 23% to 29% performance decrease due to the tenfold increase of the number of connections is significant, but moderate.

It should be noted that a two order of magnitude difference was covered in the number of connections with only three measurements because the authors wanted to demonstrate the feasibility of the tests

Table 8
Throughput of Jool as a function of traffic direction, 4M connections, 16 CPU cores.

Traffic direction	Upload	Bidirectional	Download
Error (fps)	200	400	200
Median (fps)	490,820	497,654	521,093
Minimum (fps)	484,960	495,702	512,304
Maximum (fps)	497,070	501,952	525,195

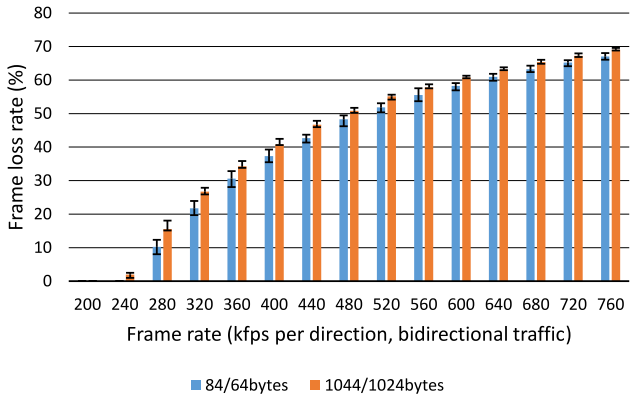

Fig. 2. Frame loss rate of the Jool stateful NAT64 implementation as a function of frame rate and frame size using bidirectional traffic.

with a relatively low number of measurements. For a more refined grain analysis, the authors recommend using smaller steps in the logarithmic scale (e.g., by doubling the number of connections). Those tests will require much more time, if a high number of connections has to be handled at a relatively low connection establishment rate.

8.3. Throughput test with unidirectional traffic

The upload-only, bidirectional, and download-only throughput of Jool was compared using 4,000,000 connections and 16 CPU cores. The results are shown in Table 8. The download throughput of Jool (521,093 fps) is somewhat higher than its upload throughput (490,820 fps), which is favorable from the ISP point of view, as the volume of the download traffic is expected to be higher than the volume of the upload traffic.

8.4. Frame loss rate

The authors did not intend to carry out a comprehensive benchmarking of Jool, so frame loss tests were performed only with 4,000,000 connections, 16 CPU cores and bidirectional traffic. Two different frame sizes were used, 84/64bytes and 1044/1024 bytes, to demonstrate that they do not make a significant difference. The results are shown in Fig. 2. The color bars show the median values whereas the bottom and top ends of the error bars show the minimum and maximum values of the results of the 10 experiments. The results comply with the expectations of the authors, which is the frame size does not make a significant difference because the lion's share of the work of Jool is the header processing.

8.5. Latency and PDV

RFC 8219 requires that *latency* and *PDV* are measured at the frame rate determined by the throughput measurement. It should be noted that IPDV is an optional test and siitperf does not support it.

For latency measurements, RFC 8219 requires the usage of at least 500 timestamps. The authors used 50,000 of them so that the potentially lost ones could be eliminated when the 99.9th percentile is calculated (for details, please refer to [34]).

The latency and PDV measurement results of Jool using 4,000,000 connections and 16 CPU cores at 497,654 fps (frames per second) rate with bidirectional traffic (that is, 248,827 fps per direction) are shown in Table 9 and Table 10, respectively.

Note: these results were included only for completeness, and the latency and PDV tests were not performed with the other implementations, as the authors did not aim to perform full benchmarking of all three stateful NAT64 implementations.

8.6. Connection tear down rate

The connection tracking table of the DUT was filled with 0.4M, 4M, and 40M connections, and their deletion time was measured. To increase the accuracy of the results, the measurements were also performed without loading any connections into the connection tracking table. For simplicity, it is called the *deletion time of an "empty" connection tracking table*, but it also contains all overhead. The second result was subtracted from the first one to get the "net" deletion time used for deleting the connections. Finally, the connection tear down rate was calculated. The results are shown in Table 11. The connection tear down rate measured with 0.4M connections was put into parenthesis as it was considered rather unreliable, since the authors got 0.16s as the difference of the values, of which the first one was measured with high uncertainty. The other two results were fairly similar to each other and they were about an order of magnitude higher than the corresponding maximum connection establishment rate results in Table 6.

9. Benchmarking tayga+iptables

There is a detailed description of the measurements with tayga plus `iptables` in Appendix A.3.

As tayga runs as a single-threaded user process, the authors did not see much value in measuring its scalability against the number of CPU cores, although it should be noted that the measurements benefited somewhat from the 16 CPU cores as they all took part in the processing of the interrupts of packet arrivals and `iptables` could also use them; however, tayga was the bottleneck.

9.1. Scalability against the number of connections

The same number of connections were used as with Jool to test the scalability of the stateful NAT64 gateway implemented by tayga plus `iptables` against the number of connections.

The maximum connection establishment rate results are shown in Table 12. As for scalability, the authors managed to achieve that the performance of the system degraded only very slightly due to the careful selection of the hash table size parameter. (As the number of connections was a power of ten and the hash table size was a power two, their ratio showed minor fluctuations across the three measurements.)

In the third column, there is a rather low minimum value, 49,853 cps, and the second smallest value among the 10 results was 49,951 cps. The measurement log file was checked and it was found that the test failed due to the loss of less than 0.01% of test frames. Moreover, the same occurred occasionally (seemingly randomly) when the test was repeated. This was very likely to be caused by tayga as the authors did not experience such an issue with `iptables`, although earlier it was tested at much higher rates and with a much higher number of connections [53].

The maximum connection establishment rate measurement was repeated by replacing the zero loss criterion with a 0.01% loss tolerance, i.e., a test was considered successful if at least 99.99% of the frames sent by the Initiator arrived at the Responder. (Please see the discussion of this approach and its consequences in Section 13.3.) The results are shown in Table 13. As expected, the median values are quite similar to

Table 9
Latency of Jool at 497,654fps cumulated rate of bidirectional traffic, 4M connections, 16 CPU cores.

	Upload latency		Download latency	
	Typical	Worst case	Typical	Worst case
Median (ms)	0.0316	0.0923	0.0302	0.0957
Minimum (ms)	0.0307	0.0795	0.0292	0.0847
Maximum (ms)	0.0364	0.1749	0.0351	0.1814

Table 10
PDV of Jool at 497,654fps cumulated rate of bidirectional traffic, 4M connections, 16 CPU cores.

Packet delay variation	Upload	Download
Median (ms)	0.0742	0.0774
Minimum (ms)	0.0576	0.0611
Maximum (ms)	0.7788	0.7826

those in Table 12 in all three columns and now the minimum value in the third column is quite close to the median.

The throughput was measured with the normal "zero loss" criterion, both in phase 1 to fill the state table (please refer to Section 13.3 for how this was possible) and in phase 2 to qualify the given step of the binary search successful. The results are shown in Table 14. The throughput results showed practically no degradation. There is a somewhat lower minimum value (93,652 fps) in the third column.

9.2. Throughput test with unidirectional traffic

The upload-only, bidirectional, and download-only throughput of tayga plus `iptables` using 4,000,000 connections and 16 CPU cores are shown in Table 15. As with Jool, the download throughput (151,757 fps) of this solution is also higher than its upload throughput (127,147 fps).

9.3. Connection tear down rate

As the parameters of the connection tracking table of `iptables` were tuned, the deletion time of the empty connection tracking table was measured with each parameter set separately, and it made significant difference, as shown in Table 16. It should be noted that the connection tear down performance characterizes `iptables` alone as tayga did not take part in the deletion of the connections.

10. Benchmarking OpenBSD PF

There is a detailed description of the measurements with OpenBSD PF in Appendix A.4. It is also explained there that OpenBSD did not support setting RSS.

10.1. Baseline measurements

The IPv4 and IPv6 packet forwarding throughput tests were performed using the version of *Multi Processor* (MP) kernel specified in the Appendix and also with its *Single Processor* (SP) variant for comparison. All results are shown in Table 17. The MP kernel achieved a rather low performance increase compared to the SP kernel, it was 29% and 20% for IPv4 and IPv6, respectively. As it was below the expectations of the authors, they checked again the output of the `top` command and found the root cause of the issue: the packet forwarding was done by a single process, the execution of which caused a significantly higher load than the processing of the interrupts, thus processing the interrupts with a separate core for each direction did not help much. An extract from the output of the `top` command is shown in Fig. 3. Therefore, the authors saw no point in measuring the scalability of the stateful NAT64 performance of PF against the number of CPU cores.

10.2. Scalability against the number of connections

The maximum connection establishment rate and throughput results are shown in Table 18 and Table 19, respectively. They show a moderate performance degradation, to a similar extent as Jool.

10.3. Throughput test with unidirectional traffic

The upload-only, bidirectional and download-only throughput of OpenBSD PF using 4,000,000 connections and MP kernel are shown in Table 20. Unlike in the previous two cases, the download throughput of PF was lower than the upload throughput. This can be disadvantageous for ISPs.

10.4. Connection tear down rate

The connection tear down rate results of OpenBSD PF are shown in Table 21.

Comparing the connection tear down rates of the three solutions, it can be stated that Jool significantly outperformed the other two solutions and OpenBSD PF produced the lowest results.

11. Investigation of the validated connection establishment rate

The validated connection establishment rate measurement has been developed to facilitate the connection tracking table capacity measurement. However, without validation, it cannot be proved that the connections are really present in the connection tracking table. Moreover, this can be a problem because of the following experience: there is a paper that aimed to demonstrate a *Denial of Service* (DoS) attack against a stateful NAT64 gateway implemented by tayga plus `iptables` in a virtual machine environment [54]. It documented that a packet with private source IP address appeared at the IPv4 interface of the gateway as `iptables` did not replace it with the public IPv4 address of the gateway (as it did with the other packets), likely due to serious overload. During benchmarking, people are looking for the limits of the tested stateful NAT64 gateway, thus they willfully cause an overload situation. Hence, an improper behavior caused by overload might happen any time. This is why the "00" version of the Internet Draft [11] recommends the usage of the validated connection establishment rate measurement.

However, validation has its cost. It uses a "safety" factor $alpha$, which may have a value less than 1. If R rate is used during the current step of the binary search for the maximum connection establishment rate, then the Responder should use $r = R * alpha$ rate during validation to avoid that the validation fails because of using a too high rate. In the current case, the unidirectional throughput has already been measured in the download direction, thus the authors already knew that it was higher than the maximum connection establishment rate for all tested NAT64 implementations. However, in the case of an untested implementation, there is no such guarantee, thus in the general case, testing with several values of alpha (e.g., 0.5 or 0.8, etc.) may be needed, if validation fails. Therefore, it can be time consuming to use validation. This is why the authors consider it important to check if validation is needed or not.

The validated maximum connection establishment rate measurements were performed with all tested NAT64 implementations and their results are displayed together with the results of the non-validated measurements in Table 22. It is clearly visible that validation makes no difference in the results, and therefore, the authors did not recommend validation in the "02" version of the Internet Draft [11].

Table 11
Connection tear down rate of Jool as a function of the number of connections, 16 CPU cores.

Number of connections	400,000	4,000,000	40,000,000
Source port numbers	40,000	40,000	40,000
Destination port numbers	10	100	1,000
Filled table deletion time median (s)	0.46	1.37	11.79
Filled table deletion time minimum (s)	0.43	1.35	11.76
Filled table deletion time maximum (s)	0.48	1.39	11.95
Empty table deletion time median (s)	0.30	0.30	0.30
Empty table deletion time minimum (s)	0.29	0.29	0.29
Empty table deletion time maximum (s)	0.31	0.31	0.31
Connections deletion time (s)	0.16	1.07	11.49
Connection tear down rate (cps)	(2,758,621)	3,755,869	3,481,288

Table 12
Maximum connection establishment rate of tayga+iptables as a function of the number of connections, 16 CPU cores.

Number of connections	400,000	4,000,000	40,000,000
Source port numbers	40,000	40,000	40,000
Destination port numbers	10	100	1,000
Hash table size	2^{20}	2^{23}	2^{27}
Number of connections / hash table size	0.3815	0.4768	0.2980
UDP timeout of iptables (s)	100	200	2000
Error (cps)	100	50	50
Median (cps)	102,929	98,680	99,218
Minimum (cps)	102,049	97,997	49,853
Maximum (cps)	103,222	99,071	99,951
Median / previous median	–	0.96	1.01

Table 13
Maximum connection establishment rate of tayga+iptables as a function of the number of connections, 16 CPU cores, Beware: Loss Tolerance: 0.01%.

Number of connections	400,000	4,000,000	40,000,000
Source port numbers	40,000	40,000	40,000
Destination port numbers	10	100	1,000
Hash table size	2^{20}	2^{23}	2^{27}
Number of connections / hash table size	0.3815	0.4768	0.2980
UDP timeout of iptables (s)	100	200	2000
Error (cps)	50	50	50
Median (cps)	103,076	98,046	99,755
Minimum (cps)	102,098	97,607	99,560
Maximum (cps)	103,954	98,583	100,146
Median / previous median	–	0.95	1.02

Table 14
Throughput of tayga+iptables as a function of the number of connections, bidirectional traffic, 16 CPU cores.

Number of connections	400,000	4,000,000	40,000,000
Source port numbers	40,000	40,000	40,000
Destination port numbers	10	100	1,000
Hash table size	2^{20}	2^{23}	2^{27}
Number of connections / hash table size	0.3815	0.4768	0.2980
UDP timeout of iptables (s)	100	200	2000
Error (fps)	100	100	100
Median (fps)	129,978	128,416	128,221
Minimum (fps)	129,588	112,402	93,652
Maximum (fps)	130,564	128,610	128,610
Median / previous median	–	0.99	1.00

12. Investigation of the effect of the order of enumeration of the port numbers

Both the benchmarking methodology and the `siitperf` benchmarking program evolved gradually. At the time, when `siitperf` did not support the pseudorandom enumeration of port numbers, the authors experienced that if the connection tracking table of `iptables` was filled using port number enumeration in increasing order, then its maximum connection establishment rate was significantly lower than in the case of pseudorandom port numbers. Therefore, the authors investigated this issue and to that end they used only `iptables` without tayga to be able to repeat the original experiments and compare their results with those in the case when pseudorandom enumeration of port numbers was used. The results of those experiments when simple

```
                                        [...]
CPU02 states:  0.0% user,  0.0% nice, 93.8% sys,  6.2% spin,  0.0% intr,  0.0% idle
                                        [...]
CPU09 states:  0.0% user,  0.0% nice,  0.0% sys,  0.0% spin, 25.0% intr, 75.0% idle
                                        [...]
CPU25 states:  0.0% user,  0.0% nice,  0.0% sys,  0.0% spin, 26.7% intr, 73.3% idle
                                        [...]
```

Fig. 3. An extract from the output of the top command. The deleted lines showed all 0 values. The load was 200 fps per direction IPv6 traffic. The test failed.

Table 15
Throughput of tayga+iptables as a function of traffic direction, 4M connections, 16 CPU cores, hash table size: 2^{23}, UDP timeout: 200 s.

Traffic direction	upload	bidirectional	download
Error (fps)	100	100	100
Median (fps)	127,147	128,416	151,757
Minimum (fps)	125,683	112,402	151,268
Maximum (fps)	128,222	128,610	152,049

pseudorandom port numbers were used are also included because the authors believe that they convey some important lessons.

A very important difference compared to the tests in Section 9 was that for the tests in this section, a much lower hash table size was used compared to the number of connections. The value of the `hashsize` parameter was 2^{19}, and 40,000 source port numbers and 100 destination port numbers were used for the experiments, thus, the number of connections was 4,000,000 when linear or pseudorandom port number enumerations were used. However, when pseudorandom port numbers were used, then several combinations were repeated, and thus only less than 2.53M connections were established. (The exact number was measured; please refer to Table 23.) For this reason, the experiment was repeated also with 40,000,000 test frames in phase 1, which resulted in more than 3,999,000 connections. Besides the results and the usual additional parameters (including the number of connections per hash table size), a line was also added to Table 23 in which the *ending new connection ratio* was displayed, i.e., the ratio of those test frames that resulted in a new connection at the end of the test (expressed in percentage). The 2,371,094 cps maximum connection establishment rate of the experiment when 4M frames were sent with pseudorandom port numbers in phase 1 can be explained by two things:

1. The average length of the linked lists starting from each hash table entry was only 4.82.
2. At the end of the experiment, only 36.79% of the test frames resulted in a new connection.

As for the experiment with 40M phase 1 frames, it can be regarded as a continuation of the previous one by sending another 36M phase 1 frames. While the average length of the linked lists was increasing, up to 7.6290, the proportion of the test frames that resulted in a new connection was decreasing to 0.0055%, which was nearly negligible; the test finished like an upload-only test.

These results highly justify the earlier decision of the authors to introduce pseudorandom port number enumeration as the maximum connection establishment rate could not be clearly measured without it.

Another important lesson is that linear port number enumeration and pseudorandom port number enumeration do not result in significantly different maximum connection establishment rates with `iptables`. However, these results do not tell anything about other implementations. Therefore, the authors still recommend the *pseudorandom enumeration* of port numbers in compliance with RFC 4814.

13. Discussion of findings

13.1. Feasibility and significance of the proposed tests

13.1.1. New performance metrics

The benchmarking measurements have shown that the proposed measurement procedures for *maximum connection establishment rate* and *connection tear down rate* can be carried out for all three implementations and they produce meaningful results. The new performance metrics are meaningful, and characterize well the operation of the examined stateful NAT64 implementations. The knowledge of the maximum connection establishment rate is a precondition to fill both the connection tracking table of the DUT and the state table of the Tester, which are needed in order to execute phase 2. The connection tear down rate also reveals information about how much work it costs for the given stateful NAT64 implementation to delete a connection.

However, the authors could not validate their proposed *connection tracking table capacity* measurement procedure with OpenBSD because one billion states were used (please refer to Appendix A.4), and so they intend to deal with it separately.

13.1.2. The “classic” performance metrics

It has been demonstrated that the execution of phase 2 is possible with all three stateful NAT64 implementations. Thus the “classic” measurement procedures (throughput, frame loss rate, latency, etc.), which were not feasible due to the contradicting requirements of RFC 2544 and RFC 4814 in the case of stateful NAT64 gateways, can be carried out.

13.2. Refinements of the tests

13.2.1. Validation of connection establishment

According to the results, the alpha “safety” factor with a value of less than 1 was not necessary for any of the tested implementations. Moreover, the validated maximum connection establishment rate measurement gave the same results as the one without validation, so the authors do not recommend using the validated one.

13.2.2. Scalability against the number of CPU cores

It was pointed out that scalability regarding the number of CPU cores was an important characteristic feature of the software-based stateful NAT64 implementations. Using the powers of 2 as the number of active CPU cores proved to be an efficient solution regarding the ratio of the tested range of the number of CPU cores and the number of tests necessary.

13.2.3. Scalability against the number of connections

RFC 8219 requires testing with a different number of network flows, but it does not specify the implementation details. The authors recommended using the size of the destination port number range as a parameter and it proved to work. A tenfold increase proved to be efficient regarding the ratio of the tested range and the number of tests. Furthermore, there is enough “reserve” in the method, so testing with 40,000 source port numbers and 10,000 destination port numbers can achieve 400,000,000 connections.

Table 16
Connection tear down rate of tayga+iptables as a function of the number of connections, 16 CPU cores.

Number of connections	400,000	4,000,000	40,000,000
Source port numbers	40,000	40,000	40,000
Destination port numbers	10	100	1,000
Hash table size	2^{20}	2^{23}	2^{27}
Number of connections / hash table size	0.3815	0.4768	0.2980
UDP timeout of iptables (s)	100	200	2000
Filled table deleletion time median (s)	0.69	6.32	71.01
Filled table deletion time minimum (s)	0.67	6.24	70.42
Filled table deletion time maximum (s)	0.74	6.41	72.23
Empty table deletion time median (s)	0.21	0.36	2.96
Empty table deletion time minimum (s)	0.20	0.34	2.94
Empty table deletion time maximum (s)	0.22	0.38	2.98
Connections deltion time (s)	0.48	5.96	68.05
Connection tear down rate (cps)	833,333	671,141	587,803

Table 17
Throughput of OpenBSD packet forwarding, bidirectional traffic.

	IPv4 traffic		IPv6 traffic	
	SP	MP	SP	MP
Error (fps)	400	400	200	200
Median (fps)	499,608	642,185	275,194	331,054
Minimum (fps)	487,890	641,014	274,804	330,272
Maximum (fps)	499,608	643,358	275,194	333,006
Scale up	–	1.29	–	1.20

Table 18
Maximum connection establishment rate of OpenBSD PF as a function of the number of connections, MP.

Number of connections	400,000	4,000,000	40,000,000
Source port numbers	40,000	40,000	40,000
Destination port numbers	10	100	1,000
Error (cps)	50	40	50
Median (cps)	120,214	85,039	74,022
Minimum (cps)	118,701	84,882	73,680
Maximum (cps)	122,411	85,351	74,266
Median / previous median	–	0.71	0.87

Table 19
Throughput of OpenBSD PF as a function of the number of connections, bidirectional traffic, MP.

Number of connections	400,000	4,000,000	40,000,000
Source port numbers	40,000	40,000	40,000
Destination port numbers	10	100	1,000
Error (fps)	200	80	100
Median (fps)	237,304	198,828	173,338
Minimum (fps)	236,912	198,046	172,946
Maximum (fps)	250,584	199,452	174,120
Median / previous median	–	0.84	0.87

Table 20
Throughput of OpenBSD PF as a function of traffic direction, 4M connections, MP.

Traffic direction	Upload	Bidirectional	Download
Error (fps)	100	80	100
Median (fps)	237,987	198,828	181,445
Minimum (fps)	234,081	198,046	180,761
Maximum (fps)	239,549	199,452	181,737

13.2.4. Potential additional way of expressing network flows

OpenBSD does not support setting RSS, so that the port numbers are also used by the hash function. In addition to using pseudorandom port numbers, multiple IP addresses could be used. It can only be introduced by the Initiator because the Responder can use only the four tuples it has received. If the Initiator uses multiple source IP addresses, it can provide entropy only for the IPv6 interface of the DUT because the source IP addresses are replaced by the stateful NAT64 gateway. Using multiple destination addresses may help in both directions.

Note: the authors are aware that RFC 2544 also recommends testing with 256 destination networks, but it is for router testing and network operators usually separate the stateful NAT64 function and the routing function. Even if the two functions are implemented by the same device, the proposed methodology deals with the benchmarking of the stateful NAT64 function.

13.3. Considering non-zero loss acceptance criterion

In Section 9.1, the maximum connection establishment rate measurements were performed with 99.99% acceptance criterion. Commercial network performance testers usually have a parameter called *loss tolerance* and allow its setting to be higher than zero value. The authors also used 0.01% loss tolerance in some cases in [55] and demonstrated its effects in [56]. However, if non-zero acceptance criterion is used for measuring the maximum connection establishment rate and then the received result is used during phase 1 of a test that has phase 2, subsequently some possible four tuples may be missing from both the connection tracking table of the DUT and from the state table of the

Table 21
Connection tear down rate of OpenBSD PF as a function of the number of connections, MP.

Number of connections	400,000	4,000,000	40,000,000
Source port numbers	40,000	40,000	40,000
Destination port numbers	10	100	1,000
Filled table deletion time median (s)	1.45	11.56	94.20
Filled table deletion time minimum (s)	1.36	11.03	91.73
Filled table deletion time maximum (s)	1.78	13.81	118.52
Empty table deletion time median (s)	0.37	0.37	0.37
Empty table deletion time minimum (s)	0.36	0.36	0.36
Empty table deletion time maximum (s)	0.37	0.37	0.37
Connections deletion time (s)	1.08	11.19	93.83
Connection tear down rate (cps)	370,370	357,622	426,303

Table 22
Comparison of the validated and non-validated maximum connection establishment rate of each implementation, 4M connections, alpha=1.

Name of implementation	Jool		tayga + iptables		OpenBSD PF	
Type of test	Validated	Non-val.	Validated	Non-val.	Validated	Non-val.
Error (cps)	400	400	50	50	40	40
Median (cps)	484,863	483,153	98,437	98,680	85,000	85,039
Minimum (cps)	475,829	472,411	93,701	97,997	84,648	84,882
Maximum (cps)	487,548	486,572	98,875	99,071	85,351	85,351

Table 23
Maximum connection establishment rate of iptables, 16 CPU cores, various types of port number generation algorithms.

Type	Linear enumeration	Pseudorandom	Pseudorandom	Pseudorandom enumeration
Source port numbers	40,000	40,000	40,000	40,000
Destination port numbers	100	100	1,000	1,00
Number of phase 1 frames	4M	4M	40M	4M
Hash table size	2^{19}	2^{19}	2^{19}	2^{19}
Number of connections median	4,000,000	2,528,332	3,999,782	4,000,000
Number of connections minimum	–	2,527,212	3,999,760	–
Number of connections maximum	–	2,529,555	3,999,817	–
Number of connections / hash table size	7.6294	4.8224	7.6290	7.6294
Ending new connection ratio (%)	100	36.79	0.0055	100
UDP timeout (s)	100	100	100	100
Error (cps)	1,000	2,000	2,000	1,000
Median (cps)	1,307,616	2,371,094	2,337,890	1,305,663
Minimum (cps)	1,297,851	2,248,046	2,208,984	1,295,898
Maximum (cps)	1,315,429	2,396,484	2,404,296	1,313,476

Tester. If this happens, then the `siitperf` binary reports the problem and does not start the phase 2 measurement. The situation can be handled by the shell script in various ways:

- It may also report an error and stop testing.
- It may re-execute the given elementary test until its phase 1 test is successful. (It was done in Section 9.1 for the throughput test.)
- It may use, e.g., a 0.01% lower state table size than the number of frames sent in phase 1. (The authors followed this approach in [30].)

13.4. Further considerations

13.4.1. Potential "breaking" of the requirements of RFC 4814

The proposed methodology reduces the source port number and destination port number ranges recommended by RFC 4814. The authors would like to emphasize that this does not mean breaking the requirements. The statement is supported by two arguments:

1. RFC 4814 says that if the service is identified by the destination port number, then it may have a fixed value. This is a much more significant reduction in the potential port number combinations than what was used by the authors.
2. Section 4.1 of RFC 4814 explains the aim of using pseudo-random identifiers including port numbers. It is to distribute traffic among processing elements of the benchmarked network interconnect device. In an earlier paper [28], a fixed destination port number and only 1,600 different source port numbers were used; this was still enough for hashing, as the CPU idle times were practically the same for all CPU cores.

It may also be noted that the vast majority of the Internet traffic use only a few very popular destination port numbers according to [32]. Therefore, if Internet traffic is considered, RFC 4814 uses too high number of source port number and destination port number combinations, and the reduced ranges recommended by the authors can even be considered a better approximation of the port number combinations of the Internet traffic than that of RFC 4814.

13.4.2. Legal consideration

Data retention legislation may require detailed logging so that the network operators can provide information about the user of a given IP address and port number combination at a given time. Thus, it may be worth measuring the performance of the various stateful NAT64 implementations with the logging enabled, too.

14. Conclusion

The proposed benchmarking methodology has been successfully validated with three radically different stateful NAT64 implementations. The proposed methodology made it possible to perform the "classic" tests like throughput, frame loss rate, latency, etc. with stateful NAT64 gateways in phase 2. The new maximum connection establishment rate benchmarking procedure proved to be good and satisfactory; there is no need for the validation version. The new connection tear down rate measurement also worked perfectly. The authors intend to work on the

connection tracking table capacity measurement. Recommendations for scalability measurements also proved to be effective.

As for the performance of the benchmarked stateful NAT64 implementations, Jool significantly outperformed tayga plus iptables and OpenBSD PF. However, Jool also showed poor scalability with the number of active CPU cores; an increase in the number of active CPU cores over 4 resulted in no significant performance improvement.

CRediT authorship contribution statement

Gábor Lencse: Conceptualization, Methodology, Software, Measurements, Validation, Investigation, Writing – original draft, Writing – review & editing, Visualization. **Keiichi Shima:** Methodology, Writing – review & editing. **Kenjiro Cho:** Writing – review & editing, Project administration.

Declaration of competing interest

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

Data availability

No data was used for the research described in the article.

Acknowledgments

The experiments were carried out remotely using the resources of NICT StarBED, 2-12 Asahidai, Nomi-City, Ishikawa 923-1211, Japan.

The authors would like to thank Shuuhei Takimoto for the possibility to use StarBED, as well as to Tsukasa Nishita and Makoto Yoshida for their help and advice on StarBED usage related issues.

Gábor Lencse would like to thank the National Institute of Information and Communications Technology (NICT), Japan for their support of his stay at the Research Laboratory of Internet Initiative Japan, where his research topic was the performance analysis of stateful NAT64 implementations.

The authors would like to thank Vargáné Katalin Kiss and John Kowalchuk, Széchenyi István University, for the English language proofreading of the manuscript.

Funding

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

Appendix

A.1. Linux test system

The topology of the Linux test system is shown in Fig. 4. The Tester and DUT were Dell PowerEdge R430 servers with two Intel Xeon E5-2683v4 CPUs, 384 GB 2400 MHz DDR4 RAM and an Intel 10G dual-port X540 NIC. Direct cable connections were used, Hyper-threading was switched off, and the CPU clock frequency of the servers was set to a fixed 2.1 GHz (using the `tlp` Linux package) to ensure stable measurement results.

As for the Tester, Debian 9.13 with 4.9.0-16-amd64 kernel, 16.11.11-1+deb9u2 DPDK, and `siitperf` latest commit 29643e6 on Sep 24, 2022 were used. As for the DUT, Debian 10.13 with 4.19.0-20-amd64 kernel was used.

The `maxcpus =` n kernel command line parameter was used to set the number of active CPU cores to n, and after changing the value

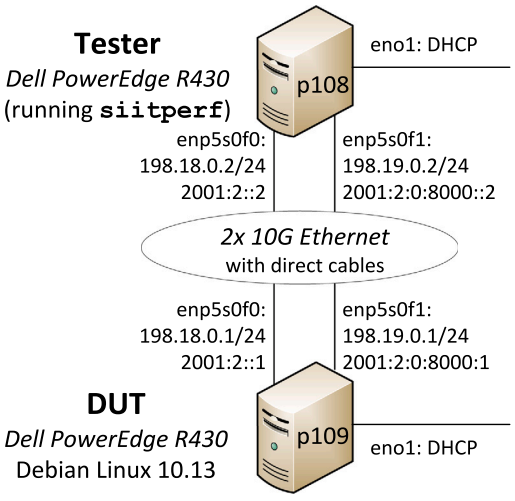

Fig. 4. Topology of the Linux test system.

of n, a "Power Cycle System (cold boot)" power control action of the iDRAC (Integrated Dell Remote Manager Controller) of the server was always performed. It was done in such a way because using only a simple restart after changing the number of active CPU cores caused inconsistent measurement results, and this type of reboot solved the problem. The details of the issue were disclosed in Section 3.E.4 of [55].

RSS was set so that the port numbers were also used by the hash function when the interrupts were distributed among the CPU cores using the appropriate one from following four commands (please refer to the two brace expansions to get them):

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

A.2. Jool measurement setup and execution

The version of Jool used was 4.1.7, the *most mature version* of Jool as of September 2022.

Jool has two modes of operation. One of them is called "iptables Jool" and it requires an `iptables` rule to give over the packets to Jool. The other one is called "netfilter Jool" and Jool tries to "grab" all the packets that it can process. The latter was used as it was simpler to set up.

The proposed methodology required to begin every single elementary test with an empty connection tracking table. It was solved by starting Jool right before each elementary test and by stopping it after finishing the given elementary test. The measurement (bash shell) script that ran on the p108 server executed a short script `set-jool` on p109 using `ssh` with key-based authentication before each test. The relevant content of the `set-jool` script was:

```
modprobe jool
jool instance add --netfilter --pool6 64:ff9b::/96
jool pool4 add 198.19.0.1 --udp 1-65535
```

In addition to this, the script was also able to set the *UDP timeout* value when it was supplied as a command line parameter, but the authors did not need to change the 5-minute default value.

Subsequently, the measurement script performed an elementary test with a single execution of the `siitperf` program, and finally, Jool was stopped by remotely executing the `del-jool` script that contained only a single command:

```
modprobe -r jool
```

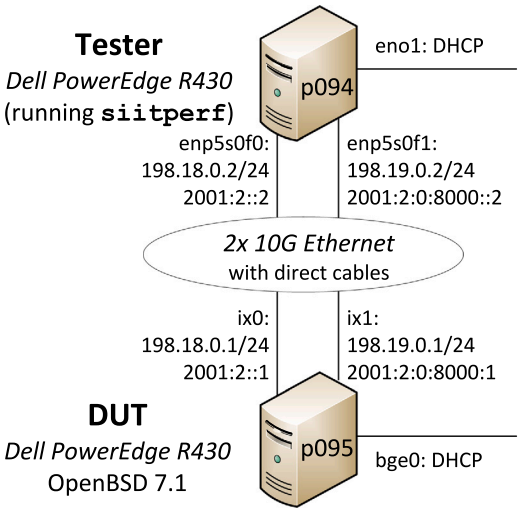

Fig. 5. Topology of the OpenBSD test system.

A.3. Tayga+iptables measurement setup and execution

The version of tayga used was 0.9.2-8 amd64. A *Network Specific Prefix* (NSP) was used for NAT64 translation instead of the NAT64 well-known prefix for reasons specific to tayga. It was set in the `/etc/tayga.conf` file as follows:

```
prefix 2001:2:0:1::/96
```

By default, tayga sets everything to implement stateful NAT64 including the setting of the appropriate `iptables` rule, but it was disabled because the authors wanted to do it manually as a part of the `set-iptables` start script.

As tayga is much slower than Jool and the default timeout value of `iptables` for UDP traffic is 30 s, it had to be changed. Regarding how many connections they can handle, there was an important difference between Jool and `iptables`. For Jool, the only limit is the memory capacity of the server. However, `iptables` has two important parameters that need to be tuned to handle a high number of connections. One of them is the size of the connection tracking table (`nf_conntrack_max`) and the other is the size of the hash table (`hashsize`) [52]. These parameters were tuned to be able to handle the required number of connections and to demonstrate that the careful usage of hashing may make `iptables` quite scalable regarding the number of connections. (Please see the used values in the tables of the results.) To that end, the measurement script passed two command line parameters to the `set-iptables` script: the first one was a number to be used as an exponent of 2, which was set as both hash table size and connection tracking table size and the second one was the timeout time expressed in seconds.

The relevant content of the `set-iptables` file was:

```
/sbin/iptables -t nat -A POSTROUTING -o enp5s0f1 \
 -j MASQUERADE
size=$((2**$1))
echo $size > \
 /sys/module/nf_conntrack/parameters/hashsize
/sbin/sysctl -w net.netfilter.nf_conntrack_max=$size
/sbin/sysctl -w \
 net.netfilter.nf_conntrack_udp_timeout=$2
```

And the `del-iptables` file had the following content:

```
/sbin/iptables -t nat -D POSTROUTING -o enp5s0f1 \
 -j MASQUERADE
/sbin/conntrack -F
```

A.4. Openbsd PF measurement setup and execution

To benchmark OpenBSD PF, another test system was built using the exact same type of Dell PowerEdge 430 servers and they were interconnected in the same way with direct cable links, as shown in Fig. 5. The software configuration of the Tester was also the same, but OpenBSD 7.1 (the latest version of OpenBSD as of September 2022) with GENERIC.MP#465 amd64 kernel was installed on the DUT.

The authors did not find any way to set RSS under OpenBSD, this was supported by the answer received on the misc@openbsd.org mailing list that a few network drivers (including `ix`) have support for multiple queues, “but there’s no interface to adjust what’s fed into the hash function” [57]. This was consistent with the experience of the authors:

- `dmesg` reported 16 queues for the NICs (`ix0` and `ix1`)
- when the load of the CPU cores of the DUT was checked using the `top` command during a throughput test with bidirectional traffic, all interrupts were processed by two CPU cores (one core for each direction).

Therefore, the authors could not use a proper RSS setting with OpenBSD.

The authors encountered another issue with PF. When the number of states was set to a value required by the number of connections (or a few times higher), problems were experienced during the tests. (The results significantly deteriorated with time.) To mitigate the issue, a several order of magnitude higher value for the number of states was set than would have been normally required by the situation. (Normally, the size of the state table was set to a somewhat higher value than necessary to be able to store the connections, e.g., if the authors wanted to store 1M connections, 2M would be the choice, but not 1000M.)

For handling PF, the usual remotely executed scripts were used. The content of `set-pf` was the following single line:

```
pfctl -f /etc/pf-set-nat64
```

The content of the `pf-set-nat64` file was derived from the original `pf.conf` file by appending it with the following lines:

```
set skip on bge0 # to protect ssh
set limit states 1000000000 # 1000M
set timeout interval 3600 # 1 hour
pass in on ix0 inet6 from any to 64:ff9b::/96 \
 af-to inet from 198.19.0.1
```

Similarly, the `del-pf` file contained:

```
pfctl -f /etc/pf-del-nat64
```

Furthermore, the content of the `pf-del-nat64` file was derived from the original `pf.conf` file by appending it only with the following line:

```
set skip on bge0 # to protect ssh
```

This line saved `ssh` connections from being broken when the state table was cleared by `pfctl`.

References

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

[2] P. Wu, Y. Cui, J. Wu, J. Liu, C. Metz, Transition from IPv4 to IPv6: A state-of-the-art survey, IEEE Commun. Surv. Tutorials 15 (3) (2013) 1407–1424, http://dx.doi.org/10.1109/SURV.2012.110112.00200.

[3] M. Bagnulo, A. Sullivan, P. Matthews, I. Beijnum, DNS64: DNS extensions for network address translation from IPv6 clients to IPv4 servers, IETF RFC 6147, 2011, http://dx.doi.org/10.17487/RFC6147.

[4] N. Skoberne, O. Maennel, I. Phillips, R. Bush, J. Zorz, M. Ciglaric, IPv4 address sharing mechanism classification and tradeoff analysis, IEEE/ACM Trans. Netw. 22 (2) (2014) 391–404, http://dx.doi.org/10.1109/TNET.2013.2256147.

[5] M. Mawatari, M. Kawashima, C. Byrne, 464XLAT: Combination of Stateful and Stateless Translation, in: RFC 6877, Internet Request for Comments 6877, RFC Editor, 2013, http://dx.doi.org/10.17487/RFC6877.

[6] M. Nikkhah, R. Guérin, Migrating the internet to IPv6: An exploration of the when and why, IEEE/ACM Trans. Netw. 24 (4) (2016) 2291–2304, http://dx.doi.org/10.1109/TNET.2015.2453338.

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

[8] D. Newman, T. Player, Hash and Stuffing: Overlooked Factors in Network Device Benchmarking, IETF RFC 4814, 2008, http://dx.doi.org/10.17487/RFC4814.

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

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

[11] G. Lencse, K. Shima, Benchmarking Methodology for Stateful NATxy Gateways Using RFC 4814 Pseudorandom Port Numbers, Internet Draft, 2022, [online], available: https://datatracker.ietf.org/doc/html/draft-ietf-bmwg-benchmarking-stateful-00.

[12] K.J.O. Llanto, W.E.S. Yu, Performance of NAT64 versus NAT44 in the context of IPv6 migration, in: Proc. International Multiconference of Engineers and Computer Scientists 2012, IMECS 2012, Hong Kong, Hongkong, 2012, pp. 638–645.

[13] C.P. Monte, M.I. Robles, G. Mercado, C. Taffernaberry, M. Orbiscay, S. Tobar, R. Moralejo, S. Pérez, Implementation and evaluation of protocols translating methods for IPv4 to IPv6 transition, J. Comput. Sci. Technol. 12 (2) (2012) 64–70.

[14] S. Yu, B.E. Carpenter, Measuring IPv4-IPv6 Translation Techniques, Computer Science Technical Reports (2012-001), Dept. of Computer Science, Univ. of Auckland, Auckland, New Zeeland, 2012, [online], available: http://hdl.handle.net/2292/13586.

[15] G. Lencse, S. Répás, Performance analysis and comparison of different DNS64 implementations for Linux, OpenBSD and FreeBSD, in: Proc. IEEE 27th Internat. Conf. on Advanced Information Networking and Applications, AINA 2013, Barcelona, Catalonia, Spain, 2013, pp. 877–884, http://dx.doi.org/10.1109/AINA.2013.80.

[16] G. Lencse, G. Takács, Performance analysis of DNS64 and NAT64 solutions, Infocommun. J. 4 (2) (2012) 29–36.

[17] G. Lencse, S. Répás, Performance analysis and comparison of the TAYGA and of the PF NAT64 implementations, in: Proc. 36th International Conference on Telecommunications and Signal Processing, TSP 2013, Rome, Italy, 2013, pp. 71–76, http://dx.doi.org/10.1109/TSP.2013.6613894.

[18] S.R. Répás, P. Farnadi, G. Lencse, Performance and stability analysis of free NAT64 implementations with different protocols, Acta Technica Jaurinensis 7 (4) (2014) 404–427, http://dx.doi.org/10.14513/actatechjaur.v7.n4.340.

[19] V.J.D. Barayuga, W.E.S. Yu, Study of packet level UDP performance of NAT44, NAT64 and IPv6 using iperf in the context of IPv6 migration, in: Proc. 2014 International Conference on IT Convergence and Security, ICITCS, Beijing, China, 2014, http://dx.doi.org/10.1109/ICITCS.2014.7021814.

[20] V.J.D. Barayuga, W.E.S. Yu, Study of packet level TCP performance of NAT44, NAT64 and IPv6 using iperf in the context of IPv6 migration, in: Proc. 2015 5th International Conference on IT Convergence and Security, ICITCS, Kuala Lumpur, Malaysia, 2015, http://dx.doi.org/10.1109/ICITCS.2015.7293006.

[21] M. Georgescu, H. Hazeyama, Y. Kadobayashi, S. Yamaguchi, Empirical analysis of IPv6 transition technologies using the IPv6 network evaluation testbed, EAI Endorsed Trans. Ind. Netw. Intell. Syst. 2 (2) (2015) http://dx.doi.org/10.4108/inis.2.2.e1.

[22] A. Quintero, F. Sans, E. Gamess, Performance evaluation of IPv4/IPv6 transition mechanisms, Int. J. Comput. Netw. Inf. Secur. 8 (2) (2016) http://dx.doi.org/10.5815/ijcnis.2016.02.01.

[23] J. Pokorny, NAT64 Performance Evaluation (MSc thesis), Brno University of Technology, 2019, [online], available: https://www.fit.vut.cz/study/thesis/21826/.

[24] ntop, PF_RING: High-speed packet capture, filtering and analysis, [online], available: https://www.ntop.org/products/packet-capture/pf_ring/,

[25] G. Lencse, Benchmarking stateless NAT64 implementations with a standard tester, Telecommun. Syst. 75 (3) (2020) 245–257, http://dx.doi.org/10.1007/s11235-020-00681-x.

[26] C. Popoviciu, A. Hamza, G.V. de Velde, D. Dugatkin, IPv6 Benchmarking Methodology for Network Interconnect Devices, IETF RFC 5180, 2008, http://dx.doi.org/10.17487/RFC5180.

[27] G. Lencse, K. Shima, Performance analysis of SIIT implementations: Testing and improving the methodology, Comput. Commun. 156 (1) (2020) 54–67, http://dx.doi.org/10.1016/j.comcom.2020.03.034.

[28] G. Lencse, N. Nagy, Towards the scalability comparison of the Jool implementation of the 464XLAT and of the MAP-T IPv4aaS technologies, Int. J. Commun. Syst. 35 (18) (2022) e5354, http://dx.doi.org/10.1002/dac.5354, [online], available: https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.5354.

[29] X. Li, C. Bao, W. Dec (Ed.), O. Troan, S. Matsushima, T. Murakami, Mapping of Address and Port using Translation (MAP-T), RFC 7599, RFC Editor, 2015, http://dx.doi.org/10.17487/RFC7599, [online], available: https://www.rfc-editor.org/rfc/rfc7599.txt.

[30] G. Lencse, A. Bazsó, Benchmarking methodology for IPv4aaS technologies: Comparison of the scalability of the Jool implementation of 464XLAT and MAP-T, unpublished.

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

[32] T. Kurahashi, Y. Matsuzaki, T. Sasaki, T. Saito, F. Tsutsuji, Periodic observation report: Internet trends as seen from IIJ infrastructure - 2020, Internet Infrastruct. Rev. 49 (2021) [online], available: https://www.iij.ad.jp/en/dev/iir/pdf/iir_vol49_report_EN.pdf.

[33] G. Lencse, Siitperf: an RFC 8219 compliant SIIT and stateful NAT64/NAT44 tester, source code, 2022, https://github.com/lencsegabor/siitperf.

[34] G. Lencse, Design and implementation of a software tester for benchmarking stateless NAT64 gateways, IEICE Trans. Commun. E104-B (2) (2021) 128–140, http://dx.doi.org/10.1587/transcom.2019EBN0010.

[35] G. Lencse, Adding RFC 4814 random port feature to siitperf: Design, implementation and performance estimation, Int. J. Advances in Telecomm., Electrotechn. Signals Syst. 9 (3) (2020) 18–26, http://dx.doi.org/10.11601/ijates.v9i3.291.

[36] G. Lencse, Design and implementation of a software tester for benchmarking stateful NATxy gateways: Theory and practice of extending siitperf for stateful tests, Comput. Commun. 172 (1) (2022) 75–88, http://dx.doi.org/10.1016/j.comcom.2022.05.028.

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

[38] G. Lencse, S. Répás, Performance analysis and comparison of four DNS64 implementations under different free operating systems, Telecommun. Syst. 63 (4) (2016) 557–577, http://dx.doi.org/10.1007/s11235-016-0142-x.

[39] Cisco, End user license agreement, online https://www.cisco.com/c/en/us/about/legal/cloud-and-software/end_user_license_agreement.html.

[40] Juniper, End user license agreement, https://support.juniper.net/support/eula/.

[41] NIC Mexico, Jool: SIIT and NAT64, 2022, [online], available: http://www.jool.mx/en/.

[42] N. Lutchansky, TAYGA: Simple, no-fuss NAT64 for Linux, 2011, [online], available: http://www.litech.org/tayga/.

[43] P. Hansteen, The Book of PF: A No-Nonsense Guide to the OpenBSD Firewall, third ed., No Starch Press, San Francisco, ISBN: 978-1-59327-589-1, 2015.

[44] FD.io, FD.io VPP – Whitepaper, 2017, https://fd.io/docs/whitepapers/FDioVPPwhitepaperJuly2017.pdf.

[45] G. Lencse, VPP Stateful NAT64 crashes with segmentation fault, 2022, the list archive of the vpp-dev mailing list, [online], available: https://lists.fd.io/g/vpp-dev/message/22153.

[46] Ecdysis project, Ecdysis: open-source implementation of a NAT64 gateway, [online], available: https://ecdysis.viagenie.ca/,

[47] M. Asama, MAP supported Vyatta, [online], available: https://www.ginzado.ne.jp/~m-asama/vyatta/map/,

[48] VyO.S. maintainers and contributors, VyOS — The Universal Router, [online], available: https://vyos.net/,

[49] Y. Andamasov, VyOS — The universal router, planned features: nat, 2020, [online], available: https://portal.productboard.com/vyos/1-vyos-roadmap/c/28-nat64-support.

[50] xHire, WrapSix, [online], available: https://www.wrapsix.org/,

[51] B. Jacob, S.W. Ng, D.T. Wang, Memory Systems: Cache, DRAM, Disk, Morgan Kaufmann Publishers, 2008, http://dx.doi.org/10.1016/B978-0-12-379751-3.X5001-2.

[52] V. Gapon, Tuning nf_conntrack, 2019, IT Blog, personal blog, [online], available: https://ixnfo.com/en/tuning-nf_conntrack.html.

[53] G. Lencse, Scalability of IPv6 Transition Technologies for IPv4aaS, Internet Draft, Internet Request for Comments, RFC Editor, 2022, [online], available: https://datatracker.ietf.org/doc/html/draft-lencse-v6ops-transition-scalability.

[54] A. Al-Azzawi, G. Lencse, Identification of the possible security issues of the 464XLAT IPv6 transition technology, Infocommunications J. 13 (4) (2021) 10–18, http://dx.doi.org/10.36244/ICJ.2021.4.2.

[55] G. Lencse, Benchmarking authoritative DNS servers, IEEE Access 8 (1) (2020) 130224–130238, http://dx.doi.org/10.1109/ACCESS.2020.3009141.

[56] G. Lencse, A. Kovács, K. Shima, Gaming with the throughput and the latency benchmarking measurement procedures of RFC 2544, Int. J. Adv. Telecommun. Electrotech. Signals Syst. 9 (2) (2020) 10–17, http://dx.doi.org/10.11601/ijates.v9i2.288.

[57] S. Henderson, Re: Does OpenBSD support Receive Side Scaling (also called: multi-queue receiving), 2022, the list archive of the OpenBSD MISC mailing list, [online], available: https://marc.info/?l=openbsd-misc&m=166581934723445&w=2.

Gábor Lencse received M.Sc. and Ph.D. in computer science from the Budapest University of Technology and Economics, Budapest, Hungary in 1994 and 2001, respectively.

He has been working full time for the Department of Telecommunications, Széchenyi István University, Győr, Hungary since 1997. Now, he is a Professor . He is also a part time Senior Research Fellow at the Department of Networked Systems and Services, Budapest University of Technology and Economics, Budapest, Hungary since 2005. His research interests include the performance and security analysis of IPv6 transition technologies. He is a co-author of RFC 8219.

Keiichi Shima is a deputy director at the Research Institute of Advanced Technology of SoftBank Corp.

His research field is the Internet and mobile network, including designing and implementing communication protocols, operation technologies, network security, and so forth. He also works as a board member of the WIDE project operating a nationwide research network in Japan.

Kenjiro Cho is Research Director at Internet Initiative Japan, Inc. He is also a board member of the WIDE project. He received the B.S. degree in electronic engineering from Kobe University, the M.Eng. degree in computer science from Cornell University, and the Ph.D. degree in media and governance from Keio University. His current research interests include Internet data analysis, networking support in operating systems, and cloud networking.