Computer Networks 257 (2025) 111012

Contents lists available at ScienceDirect

Computer Networks

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

Maptperf: An RFC 8219 compliant tester for benchmarking MAP-T border relay routers

Ahmed Al-hamadani, Gábor Lencse *

Department of Networked Systems and Services, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, Műegyetem rkp. 3., Budapest H-1111, Hungary

ARTICLE INFO

Keywords:
Benchmarking
Border relay
IPv6 transition technologies
MAP-T
Performance analysis

ABSTRACT

The Benchmarking Working Group of IETF has published a comprehensive methodology in its RFC 8219 for benchmarking IPv6 transition technologies. The Mapping of Address and Port using Translation (MAP-T) is one of the most prominent of these technologies, which is also considered a stateless IPv4-as-a-Service (IPv4aaS) technology that belongs to the double translation category in RFC 8219. This paper presents the design and implementation of Maptperf, the World's first MAP-T benchmarking tool that complies with the guidelines of RFC 8219 to test the performance of the Border Relay (BR) router device of the technology since it is considered the focal point of its scalability. As part of the work accomplished in this paper, several design considerations, operational requirements, and configuration settings are discussed. Then, a detailed description of the implementation is disclosed, along with various important design decisions that are considered regarding implementation. Finally, the research findings related to Maptperf for two tests, the performance estimation and the functional tests, are presented. The performance estimation test proves how fast and robust Maptperf is via an initial assessment of its performance, while the functional tests include four types of measurements: Throughput, Frame Loss Rate (FLR), Latency, and Packet Delay Variation (PDV) for MAP-T implementations. For the latter case, the authors chose a popular MAP-T BR implementation, Jool, whose function is also validated via a testbed installed for this purpose.

1. Introduction

The Request For Comments (RFC) 8219 [1] was published in 2017 to standardize a comprehensive methodology for benchmarking the performance of the various IPv6 transition technologies. It classifies them into four main categories, namely, dual stack, single translation, double translation, and encapsulation. It is still a feasible demand, at least for the foreseeable future, to keep providing customers with IPv4 connectivity and services while the network operators maintain IPv6-only core and access networks. The IPv4-as-a-Service (IPv4aaS) [2] refers to those transition technologies that facilitate performing this task. The Combination of Stateful and Stateless Translation (464XLAT) [3], Dual-Stack Lite (DS-Lite) [4], Lightweight 4 over 6 (Lw4o6) [5], Mapping of Address and Port with Encapsulation (MAP-E) [6], and Mapping of Address and Port using Translation (MAP-T) [7] are regarded as the most prominent IPv4aaS technologies. Considering which technology is a better choice for a network operator to deploy is viewed as an active research topic, which can take into consideration various metric factors, such as performance, security, scalability, etc.

Each of the IPv4aaS technologies has its benefits and drawbacks [2], but those using translation have the advantage of avoiding the overhead of encapsulation. Thus, 464XLAT and MAP-T can be considered the dominant IPv4aaS translation technologies. Although 464XLAT is commonly used in mobile operator networks, it suffers from scalability limitations as it deploys stateful NAT64 at its provider-side translator (PLAT). MAP-T may be considered a better choice in terms of scalability as it is an entirely stateless technology in the operator network [8]. However, its performance needs to be tested using different kinds of measurements such as throughput, frame loss rate, latency, and packet delay variation. In general, benchmarking may also help prevent poor network performance. One can avoid poor network performance by using devices the performance of which is undoubtedly sufficient because it was checked by benchmarking tools. Thus, creating a MAP-T BR Tester fills a significant gap in the field of benchmarking.

An earlier paper by the authors [9] initiated the effort of developing the first RFC 8219 [1] compliant MAP-T BR Tester, referred to as

* Corresponding author.
E-mail address: lencse@hit.bme.hu (G. Lencse).

https://doi.org/10.1016/j.comnet.2024.111012
Received 23 January 2024; Received in revised form 14 August 2024; Accepted 21 December 2024
Available online 24 December 2024

"Maptperf", which can measure the performance of this technology, more specifically, its BR device, usually located in the core network of the service provider and considered the focal point of its scalability. However, that paper contained only theoretical considerations, without any efforts for actual implementation.

This paper documents the design, implementation, initial performance estimation, and functional testing of "Maptperf". To that end, it presents, in detail, the design principles and considerations, operational requirements, configuration settings, and implementation constructs and decisions of the Tester. It validates its performance and functionality by:

- performing an initial estimation to check whether its performance is sufficient for benchmarking various MAP-T BR implementations.
- disclosing the results of benchmarking one of the well-known free software MAP-T implementations called Jool [10].

The rest of this paper is structured as follows. Section 2 introduces the RFC 8219 [1] benchmarking methodology. Section 3 summarizes how the MAP-T technology works. Section 4 presents some of the previous related research work. Section 5 highlights the main operational elements of the RFC 8219 [1] compliant MAP-T Tester followed by a description of its scope of measurements. Section 6 discloses the design principles and considerations of the Tester. Section 7 introduces the most important design and implementation decisions of the Tester. Section 8 presents the results of the performance and functional tests that were run to test the Jool [10] MAP-T implementation. Section 9 discusses future plans for testing, performance benchmarking, and developing new research solutions. Finally, Section 10 concludes the paper.

2. The RFC 8219 benchmarking methodology

The Benchmarking Working Group of the Internet Engineering Task Force (IETF) published RFC 8219 [1] to define a comprehensive methodology for benchmarking the IPv6 transition technologies. First, it classified these technologies into four categories based on the technology used to traverse the core network. A short description for each category is as follows:

a) *Dual Stack*: both IPv4 and IPv6 stacks are implemented in the core network nodes, and for each network flow, the relevant stack will be used for communication.
b) *Single translation*: The IPvX packets are translated to IPvY packets and vice versa at the edge between the IPvX domain and the IPvY core domain, where X and Y are part of the set {4,6} and X≠Y.
c) *Double translation*: The IPvY core domain is connected to two IPvX domains. Thus, two translations will occur, the first is from IPvX to IPvY at the edge between the first IPvX domain and the IPvY core domain and the other one is from IPvY to IPvX at the edge between the IPvY core domain and the second IPvX domain. However, the translations will get reversed in the opposite direction of packet traversal.
d) *Encapsulation*: The same architecture of the double translation is followed by the Encapsulation technologies, but instead of translation, an encapsulation will occur at the edge of one IPvX domain and the IPvY core domain, and decapsulation will occur at the edge of the IPvY core domain and the other IPvX domain.

RFC 8219 [1] ignored testing the performance of dual-stack technologies as this could be fully done by the benchmarking methodology of the former RFC 2544 [11] and RFC 5180 [12]. In contrast, it focused on the other three categories of transition technologies.

In practice, the benchmarking methodology of RFC 8219 [1] is based on two types of test setups: single Device Under Test (DUT) and dual DUT. The single DUT test setup is used to evaluate the single translation technologies, where only one DUT is deployed, and it is responsible for the translation of IPvX packets transmitted by one interface of the Tester to IPvY packets to be received by another interface of the Tester. Fig. 1 exhibits the single DUT test setup. On the other hand, the dual DUT test setup is used to evaluate the double translation and encapsulation technologies, where two DUTs are deployed, one to encapsulate or translate the IPvX packets transmitted by one interface of the Tester to IPvY packets and another to decapsulate or translate the IPvY packets back to IPvX packets to be received by another interface of the Tester. Fig. 2 exhibits the dual DUT test setup. However, RFC 8219 [1] recommends additional benchmarking for each DUT separately using the single DUT test setup to avoid any potential asymmetry in behavior between the two DUTs, which could be hidden in case one is forming a bottleneck.

In addition, RFC 8219 [1] recommended following these settings for the transmitted traffic during benchmarking tests:

- Several different frame sizes should be used when the tests are run. RFC 8219 [1] recommends these sizes (in bytes): 64, 128, 256, 512, 768, 1024, 1280, 1518, 1522, 2048, 4096, 8192, and 9216.
- The IPv4 addresses used in the tests should be selected according to the recommendations of section 12 of RFC 2544 [11], whereas the IPv6 addresses should be selected according to the recommendations of Section 5 of RFC 5180 [12].
- Background (i.e., native IPv6) traffic should also be transmitted along with the foreground (translated or encapsulated) traffic, and distinct proportions of the two types should be generated during tests.
- Although the arrows in the test setups (Figs. 1 and 2) are shown as unidirectional, the generated traffic should be bidirectional. However, one can also generate unidirectional traffic to obtain fine-grained test results.
- Due to its simplicity, User Datagram Protocol (UDP) is the recommended transport layer protocol to rely on during tests.

For more details about the benchmarking methodology of RFC 8219, please refer to [1].

3. The MAP-T technology

The MAP-T technology [7] is an IPv4aaS technology that belongs to the double translation category. It is considered stateless as it performs only a stateless NAT64 translation in the network of the service provider.

Two main devices are deployed by this technology to perform its task: the Customer Edge (CE) and the Border Relay (BR) routers. As shown in Fig. 3, which exhibits the MAP-T architecture, the CE connects the subscriber's IPv4 private network to the IPv6 operator network, while the BR connects the IPv6 operator network to the native IPv4 public network. A high number of CEs and several BRs may be connected via an IPv6 network to form a MAP domain, in which all of them will adhere to the same MAP rules. The service provider can manage single or multiple MAP domains. The MAP rules in these domains are of three types, the Basic Mapping Rule (BMR), the Forwarding Mapping Rule

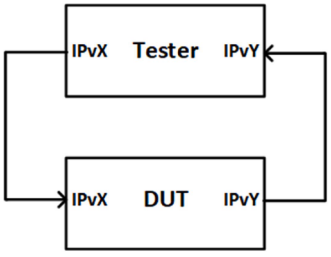

Fig. 1. Single DUT Test Setup [1].

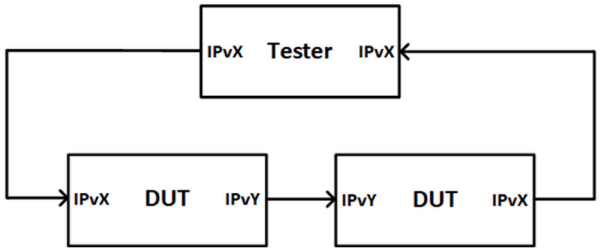

Fig. 2. Dual DUT Test Setup [1].

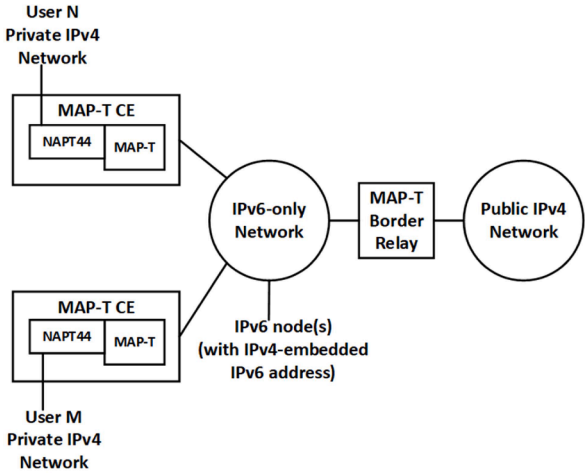

Fig. 3. MAP-T Architecture [7].

(FMR), and the Default Mapping Rule (DMR). These rules will be introduced later in this section.

The technology establishes public IPv4 address sharing among many CEs by dividing the source port range of the public IPv4 address into fixed-sized pools called port sets, which can be distinguished by a unique ID, and each of these sets will be assigned to a different CE. Therefore, each CE will be uniquely identified by the provisioned public IPv4 address plus one port set ID (PSID) and will be enforced to use only the source port numbers within its assigned port set. However, the client applications in the IPv4 private network can still use the dynamic port range as they are neither obliged nor aware of the limitation of the port set. Therefore, they can use any port numbers within the wide range of ports when they transmit their packets [8]. When the CE receives these packets, it will first perform a stateful Network Address and Port Translation (NAPT) [13] function to translate the private IPv4 address and port number used by the application to the assigned public IPv4 address and a port number within the restricted port set of the CE. Next, the CE will perform a stateless NAT46 to translate the public IPv4 address and port number into its corresponding IPv6 address and port number depending on its BMR, hence the address is called the MAP address, and the CE will, then, send the IPv6 packets via the IPv6 operator network to their destinations. With this stateless NAT46 translation, MAP-T has an advantage over other transition technologies for eliminating the overhead of "port routing" of the address plus port (*A* + *P*) approach [14] utilized by these technologies and performs only normal IPv6 routing in the operator network [8]. The structure of the MAP address is shown in Fig. 4.

The BMR depends on the following triplet to derive the MAP address:

- Rule IPv6 prefix (including its length): The service provider defines this prefix to identify all the CEs belonging to the same MAP domain. Hence, all these CEs will share the same Rule IPv6 prefix.
- Rule IPv4 prefix (including its length): This prefix is provisioned by the service provider to the CE for usage in communication. Many CEs could share the same IPv4 prefix but will be uniquely identified by their IPv4 suffix and PSID.
- Embedded Address (EA) length: Each CE has its own EA, which represents the concatenated value of both the IPv4 suffix and PSID of that CE. The number of bits in this EA is expressed by this field.

A CE uses the BMR to construct its own MAP IPv6 address. In addition to the BMR, the CE utilizes the other two rules, the DMR and the FMR, to construct the destination IPv6 address of an outgoing packet or to check the source IPv6 address of an incoming packet. If the destination is a public IPv4 site that is located "outside" the MAP network and can be accessed only by way of a BR, the DMR will be used, which embeds the public IPv4 address of that destination into the destination IPv6 address of the packet to be sent and preceding this embedded IPv4 address by a specific IPv6 prefix provisioned by the corresponding BR in the MAP domain. This prefix will help specify the BR that is responsible for routing to such a destination. Alternatively, if the destination is a private IPv4 client of another CE, the FMR will be used. This can occur if and only if the "mesh mode" is activated, i.e., a direct "CE-to-CE" connectivity is possible. The FMR is merely the same as BMR (i.e., it is used to derive and verify the IPv6 MAP address of another CE). This means that there will be a specific FMR for each connected domain. To manage all these cases, the CE will maintain a MAP rule table, where there is a single entry for the DMR and an FMR entry for each connected domain. For any sent or received packet, there will be a look-up for a match of the Rule IPv6 prefix of any of these rule entries, and once it is found, the other MAP parameters of the rule will be used accordingly.

On the other hand, the BR may be connected to one or more MAP domains. Similarly, to manage packet forwarding to/from the CEs of any one of these domains, the BR uses the FMR of the related domain. In contrast, all IPv6 packets intended to be routed to public IPv4 destinations "outside" the MAP network will be translated using a stateless NAT64 with the help of the DMR to verify their IPv6 destination address (for example, by checking if the same prefix of the BR has been included in the IPv6 destination address) and consequently extract the IPv4 destination address. However, the IPv6 source address will also be verified by FMR (i.e., the source address of the CE is valid, and the used source port is within the allowed port range of the CE) and consequently extract the IPv4 source address. Finally, the packets can get forwarded through the IPv4 public network to their destinations. It is also noteworthy that the BR maintains an equivalent MAP rule table with DMR entries (for sources/destinations outside the MAP network) and FMR entries (for the connected MAP domains).

All packets flowing in the reverse direction (i.e., they are received from the public IPv4 network and intended for an application in a private IPv4 network) will experience reverse translations with the help of

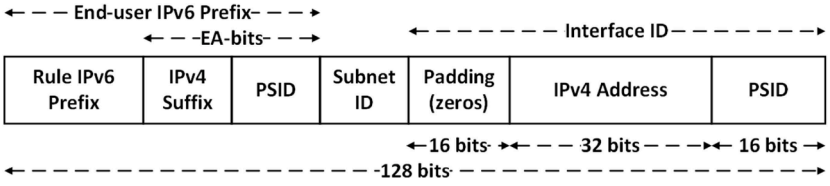

Fig. 4. MAP address format (based on [10]).

Table 1
The occasions of use of each MAP rule during packet translation.

(a) At the CE

NAT46		NAT64	
Source Address & port	Destination Address (& port)	Source Address (& port)	Destination Address & port
BMR To derive the IPv6 MAP address of the CE and manage the source port selection within its port set	DMR To derive the IPv4-embedded IPv6 address of the public IPv4 destination (i.e., outside the MAP network) FMR ("mesh mode" is active) To derive the MAP address of another CE and manage the destination port selection within the port set of the counterpart CE	DMR To verify the IPv4-embedded IPv6 address of the public IPv4 source (i.e., outside the MAP network) via match with the prefix of the corresponding router and then to extract the source address of the IPv4 packet FMR ("mesh mode" is active) To verify the MAP address of the corresponding CE via Rule IPv6 prefix match and whether the source port is within the port set of the counterpart CE and then to extract the source address of the IPv4 packet	BMR or equivalent FMR To verify that the IPv6 destination address matches the IPv6 MAP address of the CE and whether the destination port is within the port set of the CE

(b) At the BR

NAT64		NAT46	
Source Address & port	Destination Address	Source Address	Destination Address & port
FMR To verify the sending MAP address of the CE via Rule IPv6 prefix match and whether the source port is within the port set of the sending CE and then to extract the source address of the IPv4 packet	DMR To verify the IPv4-embedded IPv6 address of the public IPv4 destination (i.e., outside the MAP network) via match with the prefix of the BR and then to extract the destination address of the IPv4 packet	DMR To derive the IPv4-embedded IPv6 address of the public IPv4 source (i.e., outside the MAP network)	FMR To derive the MAP address of the receiving CE and manage the destination port selection within its port set

the same mapping rules for derivation and verification. Table 1 summarizes the occasions of using each of the MAP rules for both the CE and the BR depending on whether the manipulated address of the packet is the source or the destination address during either NAT46 or NAT64 translation.

To have things clearer by an example, suppose a MAP domain with the BMR triplet (2001:db8:8000::/50, 192.0.2.0/24, 14). The rule IPv4 prefix 192.0.2.0/24 allows an IPv4 suffix of 8 bits in length. The EA bits are 14 bits. This means that we have PSIDs of 6 bits long. That is, the number of port sets is $2^6 = 64$, each of which includes 1024 ports. Thus, the range of possible ports of PSID 0 is 0–1023, PSID 1 is 1024–2047, and so on. For example, a CE with the suffix 200 (decimal), which is c8h or 1100 1000b and PSID 10 (decimal), which is 00 1010b, can construct a MAP address of 2001:db8:8000:320a:0:c000:2c8:a and use it as a source address along with a source port number from the range (10,240–11,263) for any IPv6 packet sent into the MAP domain. The destination address of this packet can be, for example, 64:ff9b::203.0.113.56, where 64:ff9b::/64 is the DMR prefix and 203.0.113.56 is the embedded IPv4 address of the intended destination. When the corresponding BR in this domain receives this packet, it will verify the source port used as valid if selected within the allowed range of the port set of the sending CE and extract the public IPv4 address from the MAP address (i.e., 192.0.2.200) to be used as the source address of the IPv4 packet sent towards the intended destination in the public IPv4 network. The destination address of this IPv4 packet is, of course, 203.0.113.56. Later, any reply packet will follow reverse translations in the reverse direction.

For more details about the MAP-T technology, please refer to [7].

4. Related work

Few research studies have been conducted in the literature about benchmarking the performance and scalability of IPv6 transition technologies, especially MAP-T. This section summarizes the most related ones to the scope of this paper.

Georgescu et al. [15] discussed the impact of increasing the traffic load on the overall performance of four IPv4aaS technologies, namely 464XLAT, DS-Lite, MAP-E, and MAP-T, by measuring four performance metrics: round-trip delay, jitter, throughput, and packet loss. Two systems were installed to run the experiments. The first one deployed four servers, each of which ran one of four different functions: the send function of the Distributed Internet Traffic Generator (D-ITG) [16], the Customer Edge (CE) of the technology, the Provider Edge (PE) function of the technology, and the receive function of the D-ITG, while the second system deployed 31 servers: one for the PE function and 10 for each of the other three ones. Although the study presented valuable results in terms of how increasing the load downgrades the performance of the tested technologies, its experiments tested both the CE and the PE devices together at the same time, which hides which one could be the bottleneck. Consequently, the test results could not be accurate enough and reveal the actual performance of the technology devices. RFC 8219 [1] recommends testing each device independently to overcome such a situation.

There already exist testing tools developed for benchmarking some of the IPv6 transition technologies. The second author's team designed and implemented an RFC tool for benchmarking domain name servers at moderate query rates using network address translation between IPv6 clients and IPv4 servers (DNS64) [17], called dns64perf++ [18]. Next, the second author designed and implemented the world's first RFC 8219-compliant free software tool for benchmarking Stateless IP/ICMP Translation (SIIT) technology [19] (also called stateless NAT64), called siitperf [20]. Later, the original version of siitperf was extended with some new features.

In [21], the optional use of pseudorandom port numbers recommended by RFC 4814 [22] was added. Then, siitperf was updated to benchmark stateful NAT64 gateways in [23].

The second author's team proposed an RFC 8219-compliant benchmarking method to measure the performance and scalability of the five most well-known IPv4aaS technologies [24]. This method aims to simplify the problem of building a specific testing tool for each IPv4aaS technology using the dual DUT test setup of RFC 8219 [1] by aggregating the CE and the PE of the tested technology into one stateful system performing a stateful NAT44 (exploiting the fact that, in any of the five IPv4aaS technologies, at least one device, the CE or the BR, usually performs a stateful function), hence, using a stateful NAT44 testing tool for benchmarking the resulted system becomes possible. The proposed method was examined by measuring the performance and scalability of two Jool implementations, one for 464XLAT and another for MAP-T technologies. Although the proposed method could be promising, the study highlighted the limitation of relying only on the

dual DUT test setup to benchmark these technologies because the CE and PE devices have asymmetries in their behavior. Therefore, benchmarking each one individually is crucial. Section 7.6 of the study discussed potential solutions for some IPv4aaS technologies. For instance, siitperf can benchmark the customer-side translation (CLAT) and the provider-side translator (PLAT) devices of 464XLAT separately based on the single DUT test setup of RFC 8219 [1]. The CLAT uses stateless NAT46, and the PLAT uses stateful NAT64. Siitperf is capable of benchmarking both types of translation. However, for MAP-T, siitperf can only benchmark the CE device if its two subfunctions (stateful NAT44 and stateless NAT46) can be implemented using two separate devices. The authors are not aware of any testing tool for benchmarking the PE device of the MAP-T technology (i.e., the BR). This paper discusses the development of such a testing tool.

5. Operational elements and scope of measurements

As stated earlier, MAP-T is a double translation technology. This means that its benchmarking should follow the dual DUT test setup of the RFC 8219 [1] methodology. However, the two main devices of MAP-T (i.e., the CE and the BR) are not symmetric in their behavior, thus they should be benchmarked separately via the single DUT test setup according to the recommendations of RFC 8219 [1] to have more accurate test results. Since the BR represents the focal point of scalability in the technology as it, in practice, serves a high number of CEs at the same time, this paper focuses on benchmarking the BR using the single DUT test setup.

5.1. Test setup requirements

The following requirements should be met by the test setup to run according to the guidelines of RFC 8219 [1]:

- The BR is the DUT device, and the Tester is the MAP-T test program (Maptperf) executed by another device.
- Both devices must have at least two interfaces (called by their side as “left” and “right”, for simplicity), where one of them is mainly configured as IPv4 and the other as IPv6. Each side interface must be connected to its counterpart of the same type of the other device (i.e., the IPv4 to IPv4 and the IPv6 to IPv6).
- The IPv4-configured interfaces should also be configurable with IPv6 to be able to forward the non-translated native IPv6 traffic (i.e., background traffic).
- The interfaces of the devices must be able to both send and receive traffic, i.e., bidirectional traffic is possible.
- Although RFC 8219 [1] refers to the possibility of working on different media types, Ethernet would be the one to be relied on as it is the dominant media in use. The Ethernet frame sizes that could be used in testing are 64, 128, 256, 512, 768, 1024, 1280, 1518, 1522, 2048, 4096, 8192, and 9216, taking into consideration the frame overhead introduced by the NAT64 translation. For example, the 64-byte frames carrying IPv4 datagrams should be replaced by 84-byte frames carrying IPv6 datagrams to compensate for the difference in length between the IPv6 and IPv4 headers.
- RFC 8219 [1] has conformed to the requirements of its predecessors (RFC 5180 [12] and RFC 2544 [11]) to use UDP as the transport layer protocol.
- The interfaces of the devices should be set with IPv4 addresses from this recommended range 198.18.0.0/15 and/or with IPv6 addresses from this recommended range 2001:2::/48, as mentioned in section 12 of RFC 2544 [11] and Section 5 of RFC 5180 [12], respectively, and referenced by RFC 8219 [1].

5.2. Scope of benchmarking measurements

Four types of benchmarking measurements are to be executed by Maptperf: throughput, FLR, latency, and PDV. Please, refer to section 3. B of the authors’ earlier paper [9] for a detailed description of each of them.

6. Design principles and considerations

As stated earlier, benchmarking the BR adheres to the single DUT test setup of RFC 8219 [1], where the BR acts as the DUT and Maptperf will be the Tester.

As RFC 8219 [1] requires the possibility to benchmark using bidirectional traffic, the Tester can generate and receive traffic in two directions, called “forward” (the direction of the traffic is the same as that of the arrows in Fig. 1.) and “reverse”.

In the forward direction, the Tester will simulate one or more CE device(s) when sending the frames and a public IPv4 destination host or server when receiving them. The Tester will send IPv6 test frames from its IPv6 interface and should receive them as translated IPv4 frames through its IPv4 interface.

In the reverse direction, the Tester will simulate a public IPv4 source host or server when sending the frames and one or more CE devices when receiving them. The Tester will send IPv4 test frames from its IPv4 interface and should receive them as translated IPv6 frames through its IPv6 interface.

The following things should be considered in the design of the Tester:

- The Tester is not to be designed as a commodity Tester that can run many routine tests at once. Instead, the goal is to build a flexible measurement tool that is primarily used to get useful research results. The design of this tool should focus on performing certain subtasks that give important feedback about the performance of some network entities.
- The Tester should be resilient enough to work with variable testing conditions. For instance, it should handle different input parameters (e.g., frame size, frame rate, percentage of foreground and background traffic, etc.) and produce results of different types of measurements (e.g., throughput, FLR, latency, PDV).
- The Tester should simulate a high number of CEs because this is the usual scenario in the production networks. Thus, implementing such a capability requires setting some configurations and handling a suitable approach.
- Since the Tester can flow packets in two directions, “forward’’ and “reverse”, and in each direction, packets are sent and received, it needs at least four threads. To make things smoother and faster, each thread should be run on a separate CPU core. Consequently, Maptperf requires at least four CPU cores to measure traffic in both directions plus another core for the main program thread.
- The Tester will not deal with some measurements referred to by RFC 8219 [1] as they are either optional or rarely used, such as Inter Packet Delay Variation (IPDV), back-to-back frames, system recovery, and reset. The IPDV is significantly essential for getting fine-grained analysis of delay variation, especially for real-time applications, but it is marked as “optional” in the RFC 8219 [1], and alternatively, the PDV will be sufficient for this matter. The back-to-back frames and system recovery tests demand the Tester to transmit frames at the maximum rate of the connected media, which is not possible in practice when commodity servers are used to execute the test program. The reset test requires the Tester to cause or sense a DUT reset, but in this case, supplementary hardware would be needed.

7. Design and implementation decisions

RFC 8219 [1] requires running the benchmarking tests under various operational conditions to emulate, as much as possible, the environment of production networks [1]. The primary functions of the Tester will be implemented as high-performance object-oriented programs that can be

run by certain shell scripts accepting argument values that can be modified to reflect different benchmarking conditions.

7.1. Maptperf in a nutshell

The Maptperf Tester consists of three main test binaries, namely, **Maptperf-tp**, which is responsible for measuring the throughput as well as the FLR, **Maptperf-lat**, which is responsible for measuring the latency, and **Maptperf-pdv**, which is responsible for measuring the PDV. Each of these binaries accepts several varying parameter values, which change during testing and are provided as command line arguments, and other fixed parameter values, which stay static without change during testing and are provided in a specific configuration file. The design and implementation of Maptperf are inspired by that of siitperf [20]. A brief description of each configuration file parameter can be found in Appendix A.1 and a brief description of each command line argument can be found in Appendix A.2, which also specifies which of the test binaries should be supplied. The results of each of the four designated measurements (i.e., throughput, FLR, latency, and PDV) are to be obtained by running a relevant shell script on the Tester device. Each shell script will execute its related test program, passing the appropriate command-line argument values to reflect different testing conditions. The benchmarking scheme of Maptperf is shown in Fig. 5.

The following sections give an overview of the main Maptperf test binaries to allow for an easy understanding of the following sections.

7.1.1. Maptperf-tp

In this test, a stream of frames is sent from one interface of the Tester at a frame rate equal to the "*frame rate*" parameter value and should be received through the other interface of the Tester. Depending on the active direction of testing, the frames should also be of a size equal to the "*IPv6 frame size*" parameter value or "*IPv6 frame size* minus 20", which represents the IPv4 frame size. Furthermore, "*m*" number of the frames should be foreground frames, and "*n* minus *m*" (of them) should be background frames, where "*n*" and "*m*" are two relatively prime numbers. The frames should be translated by and passed through the DUT. The Tester should keep sending for a time equal to the "*test duration*" parameter value and keep receiving them concurrently from the other interface as long as the "*stream timeout*" parameter value has not expired. Then, the test will be passed if and only if the number of received frames is equal to the number of sent frames for all active directions.

This test must be repeated at least 20 times with different frame rates, frame sizes, and proportions of foreground and background frames (i.e., according to *n* and *m* parameter values). The throughput will then represent the highest frame rate at which the test is "passed".

When measuring the FLR, it can be calculated as in (1).

$$FLR(\%) = \frac{sent\ frames - received\ frames}{sent\ frames} \times 100\% \quad (1)$$

7.1.2. Maptperf-lat

In this test, a stream of frames is also sent for both directions by the Tester with the aforementioned parameter settings at the frame rate previously determined by the throughput test and within at least 120 seconds duration (see Section 7.2 of RFC 8219 [1]), and they should be received by the Tester after being translated by and passed through the DUT. Some of the frames, whose number is specified by the "*Tagged*" parameter value, should be tagged, and their indices in the stream are specified according to the uniform time distribution. The Tester can start sending the tagged frames once the "*first tagged delay*" parameter time value (usually 60 s) expires and for a duration equal to "*test duration*" minus "*first tagged delay*" and can continue receiving the frames as long as the "*stream timeout*" parameter time value has not expired.

Now, the Tester should record the timestamp of complete sending and the timestamp of complete receiving of each tagged frame before calculating two main quantities, the Typical Latency (TL) and the Worst-Case Latency (WCL). The TL represents the median of the latencies of all tagged frames, while the WCL represents the 99.9th percentile of them, where the frame latency is the difference between the receiving timestamp and the sending timestamp.

This test must also be repeated at least 20 times, and consequently, the median values of all TLs and the median value of all WCLs should be reported.

7.1.3. Maptperf-pdv

This test program can play two roles: either as a PDV Tester in the case when the value of its "*frame timeout*" parameter is 0, or as a precise throughput Tester, as recommended by [25], in the case when the "*frame timeout*" parameter value is greater than 0. In the former case, the test should be used at the previously determined throughput frame rate, while the latter one can use any desired "*frame rate*" parameter value.

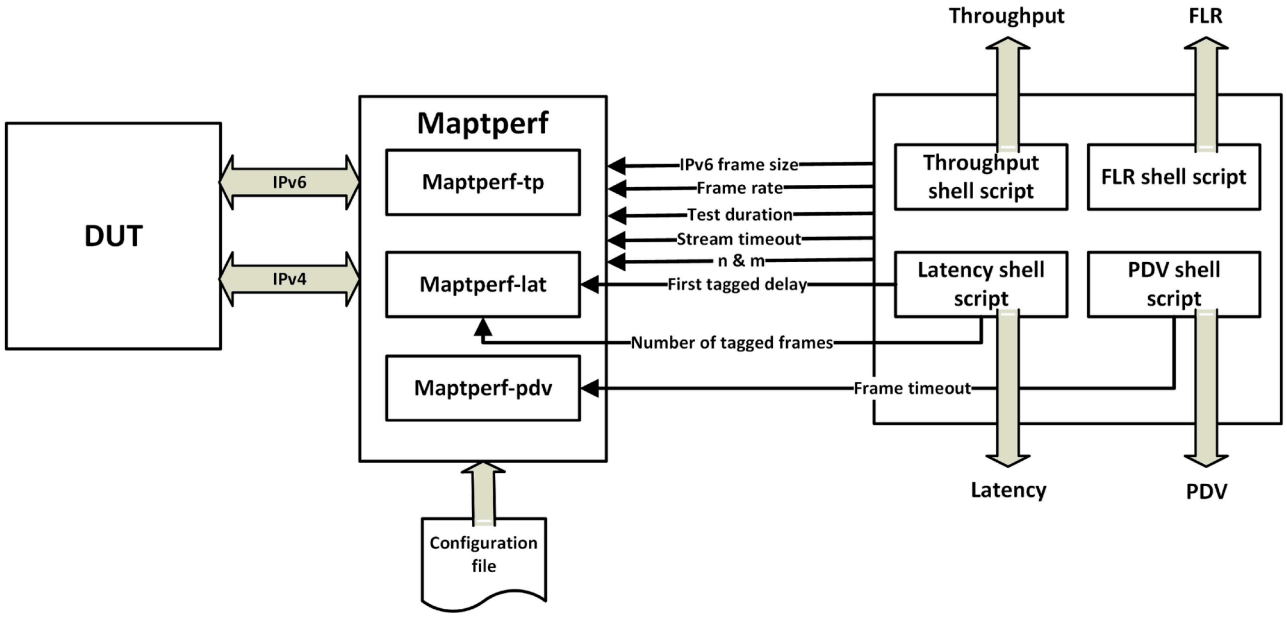

Fig. 5. The benchmarking scheme of Maptperf.

For both directions, a stream of frames is sent by the Tester and with the other aforementioned parameter settings for a time equal to the "*test duration*" parameter value and received by the Tester after being translated by and passed through the DUT. Here, the test program will not tag any frame, but rather it will set a unique identifier for every sent frame and record the timestamp of sending and the timestamp of receiving for each one of them. It keeps receiving the frames as long as the time does not exceed the "*stream timeout*" parameter value.

Then, the behavior of the test program will be identified based on the "*frame timeout*" parameter value. If it is 0, the test program will calculate PDV after reporting all the one-way latencies of all the received frames (e.g., from the recorded sending and receiving timestamps) and taking the difference between the 99.9th percentile latency value and the minimum latency value.

If the value of the "*frame timeout*" parameter is greater than 0, the test program will use it as a frame loss specifier, if the one-way latency of any single frame has exceeded it, the frame will be marked as "lost". It should be noted that this approach of calculating the throughput has a greater performance penalty as it could consume more memory resources and CPU cycles since it records and handles the sending and receiving timestamps for every single frame.

This test must also be repeated at least 20 times, and the median value of all calculated values should be considered.

7.2. Most important implementation decisions

The implementation details of the test program are described in Appendix A.5. This section provides an overview of the most important decisions taken during the implementation of the Tester to get correct and efficient operation.

7.2.1. Pregeneration of template frames

Generating the foreground and background test frames in the sending cycle could have a significant negative impact on the performance of the Tester. Therefore, Maptperf pre-generates buffers of template frames (i.e., foreground IPv6 frames for the forward direction, foreground IPv4 frames for the reverse direction, and background IPv6 frames for both directions). Later in the sending cycle, they are manipulated before sending to make them carry the correct field values (i.e., the varying or randomly generated ones that adhere to the MAP rules) such as the source and destination IP addresses, port numbers, and, consequently, the checksums (UDP checksum in the case of both IPv4 and IPv6 frames plus IPv4 header checksum in the case of IPv4 frames). This procedure considerably enhanced the performance of the Tester by saving significant time during the sending cycle.

7.2.2. The CE MAP array

The construction of the MAP addresses of the simulated CEs in the sending cycle could significantly affect the performance of the Tester. Therefore, an individual CE MAP array is built for the sender of each direction in advance to be ready for use when sending packets in the given direction.

The size of the CE MAP array is specified by the "*NUM-OF-CEs*" parameter in the `Maptperf.conf` configuration file. Each element of this array represents a simulated CE and is a struct that binds the following components:

- The IPv6 MAP address. This address will be used as a source address for the foreground packets or the background packets to be sent in the forward direction or as a destination address of the background packets to be sent in the reverse direction.
- The checksum value of the IPv6 MAP address.
- The public IPv4 address of the simulated CE. This address will be used as a destination address of the foreground packets to be sent in the reverse direction.
- The checksum value of the public IPv4 address of the simulated CE.
- The PSID of the port set assigned to the simulated CE. This element will be used for specifying the range of allowed source port numbers in testing in the forward direction or the destination port numbers in testing in the reverse direction for that CE.

The CE elements are pseudorandomly enumerated in the array and the Tester, then, cycles sequentially over all of them during the sending loop.

7.2.3. Construction of MAP addresses

The IPv6 MAP address of each simulated CE element in the CE MAP array will be constructed according to the structure described in Section 3 with the help of the bitwise operations of C++.

As it is correct to have only the BMR rule triplet (Rule IPv6 prefix with its length, Rule IPv4 prefix with its length, and the EA length) defined in the `Maptperf.conf` configuration file as Tester parameters (i.e., the user will specify only these values concerning the BMR settings according to its definition), it is, then, the responsibility of the Tester to calculate all the other values such as the number of IPv4 suffixes, the number of port sets, the number of ports in each port set, etc. from the BMR rule triplet, which helps in deriving the IPv6 MAP address.

To set a uniquely pseudorandom MAP address for each simulated CE element in the CE MAP array, the Tester creates a vector array of the pseudorandom enumerations of all possible IPv4 suffix and PSID combinations based on the Dustenfeld's random shuffle algorithm [26] in the NUMA local memory of the sender of each direction. The function responsible for building the MAP array, then, loops throughout all the enumerations and sets the MAP address of its elements accordingly. Naturally, the number of the simulated CEs must not exceed the number of all possible IPv4 suffix and PSID combinations. It is checked, and a "Configuration Error" message is given, if the input parameters are contradicting (i.e., $Num-of-CEs > 2^{BMR-EA-Length}$).

The selection of the next item in each iteration of Dustenfeld's random shuffle algorithm itself is done using the 64-bit Mersenne Twister pseudorandom number generator (std::mt19937_64), which was chosen based on the results of Oscar David Arbeláez [27].

7.2.4. Verification of the received frames

The receive function could be implemented in a way that verifies the translation functionality of the DUT for each received frame, but the authors did not see this worthwhile as they used other sufficient ways of verification for the DUT translation such as the MAP-T testbed described in Section 5 of their earlier paper [9] and the tshark captures of the traffic of the interfaces of the DUT as explained in Section 8.2. Instead, it will only validate receiving the 64-bit integer value of the ASCII-encoded "IDENTIFY" string (in the case of a normal test frame) or that of the ASCII-encoded "Identify" string (in the case of a latency test frame) at the first 8 octets of the payload data field of each received frame. This is sufficient to distinguish the test frames from other frames in the test network. The advantage gained in this way is to relieve the receive function from the burden of verifying the correctness of the MAP address for every single frame received and to obtain a more resilient and faster function.

7.2.5. Port selection

The source and destination port numbers of the UDP datagrams to be sent will be selected from certain ranges. These ranges are defined through "min" and "max" variables in the Tester program, but the values of only some of them are pre-specified in the configuration file, such as those described in Appendix A.1, and are similar to those of the extension of `Siitperf` [21] without any certain restriction about which values should be selected. Hence, the entire UDP port space (0–65,535) can be utilized. In addition, RFC 4814 [22] recommends some other settings when selecting these values. However, for the foreground packets, the values of the source port range in the forward direction and the destination port range in the reverse direction cannot be specified in

advance because they will be limited to the range of the port set of the pseudorandomly enumerated CEs, which will be known only during the sending cycle. Hence, there are no configuration parameters for these port ranges in the configuration file. On the other hand, as background traffic is not part of the MAP-T operation (i.e., not considered as translated traffic), no such limitation exists for its source and destination ports. Therefore, both forward and reverse sender threads could use the same port ranges for the background traffic, which can be set by their related configuration parameters, which are specified in Appendix A.1.

As for setting the port number value within the port range, the authors follow the approach of `Siitperf` in its extension [21], which defines three ways of changing the value in each sending cycle: Increasing, decreasing, or pseudorandom port number generation. The former two are not RFC 4814 compliant, but they are computationally less expensive and could be useful in some cases [23]. The latter uses the 64-bit Mersenne Twister random number generator (std::mt19937_64) based on the results of Oscar David Arbeláez [27].

7.2.6. Time handling

All timing activities in the Tester binaries are handled using the Time Stamp Counter (TSC) due to its high precision level yet yielding low computation cost. This counter is a 64-bit register whose value will be monotonically incremented based on the CPU clock and can be read by a single instruction, the `RDTSC` instruction [28]. However, the input and output time values in the Tester program will be handled in "seconds" or "milliseconds" and converted to/from the TSC unit [20].

All logical cores ("lcore"-s in DPDK terminology) belonging to the same CPU will share the same TSC readings, but synchronization may be needed for those belonging to different CPUs. Therefore, the user of the Tester must make sure that the four cores allocated for the four different threads of the Tester program as well as the main core running the main program of the Tester are of the same CPU to satisfy synchronized local timing among them, and thus, proper execution of the Tester program [20].

7.2.7. Proportional traffic generation

As stated earlier, RFC 8219 [1] requires sending different proportions of foreground and background traffic during testing; therefore, the authors follow the same approach of dns64perf++ (when it specified the proportions of the repetitions of domain names in the test) [29] to accomplish this in a cost-effective way that is appropriate for a fast sending cycle and ensures suitable interleaving among the two kinds of frames. It simply works as follows:

The Tester will send a foreground packet if and only if $N\%\ n < m$; otherwise, the Tester will send a background packet. N is the ordinal number of the packet to be sent and n and m are relatively prime numbers.

7.2.8. The sending start delay

As the starting of the sender and receiver functions require non-zero time, originally the "`START_DELAY`" parameter was defined as a C preprocessor constant to help in a synchronized start of the sender and receiver functions [20]. When the Tester is ready to start the sender and receiver functions, it reads the actual system time, adds the value of this parameter, and then considers the result as the time when the senders should start sending frames. (The receivers start receiving as soon as they are ready for receiving.)

Later this parameter became useful for another purpose. It was discovered that there was some frame loss at the beginning of the tests because some parts of the test system, at most, the interfaces of the DUT, were not ready to receive the sent frames directly after initializing the interfaces of the Tester. Therefore, the value of the "`START_DELAY`" parameter was increased to 2 seconds, and it fixed the issue. It should be noted that on specific hardware, a further increase of the value of this parameter (e.g., to 4 seconds) may be necessary [23].

7.2.9. Negative delay reset

As stated in Section 7.1.2, when the Tester measures the delay of the sent frame, it records the timestamp right after sending the entire frame. An interrupt might occur at this sensitive time (after sending the frame and before recording the timestamp via the `rte_rdtsc()` DPDK API), which may result in a negative delay value if servicing the interrupt takes a longer processing time than the one-way delay of the sent frame. In practice, it has no impact on the latency test of Maptperf-lat as it only measures the typical value (i.e., TL) and the worst-case value (i.e., WCL). Adversely, the PDV test could be affected as it relies on the one-way delay to obtain the measurement result. This phenomenon was first discovered in [20] and it has been relatively alleviated by resetting the negative values to "0". This solution is not perfect as it cannot be applied to the decreased delay values that are still positive. It could even happen that an interrupt might cause an incorrect receiving timestamp recording for a received frame, which may also negatively affect measuring the PDV correctly because of a wrong 99.9th percentile value. Whatever the case may be, this seldom-happening phenomenon always results in an increased PDV value, which is acceptable in the sense that the real PDV value is certainly less than the one measured by Maptperf-pdv.

8. Performance and functional tests

As RFC 8219 [1] methodology can be considered the primary reference for benchmarking IPv6 transition technologies, it includes the most important metrics for measuring the performance of real-world IPv6 transition implementations like throughput, latency, PDV, and FLR. However, one can run different types of experiments to satisfy the RFC 8219 [1] recommendations mentioned in Section 2 for accurate measurement of the performance metrics of the MAP-T BR implementations, such as the following:

- Using different frame sizes.
- Using different proportions of background (i.e., non-translated) and foreground (i.e., translated) traffic.
- Using unidirectional traffic and not only the bidirectional one for fine-grained results.

In addition to that, the authors find it crucial to consider also further parameters regarding testing the scalability of a MAP-T BR device including:

- Using a different number of served CEs.
- Using a different number of managed MAP domains.
- Using a different number of active CPU cores at the tested MAP-T BR (i.e., the DUT).

However, covering all these types of tests is out of the scope of this paper. The main focus of this paper is to validate the design and implementation of Maptperf by testing its functionality and performance through several test experiments that use basic settings (e.g., IPv6 frame size of 84 bytes, bidirectional traffic, 100% background traffic, and so on), which comply with the guidelines of RFC 8219 [1]. Subsequent research work will include such types of tests to show the practical ability of Maptperf to benchmark MAP-T BR implementations under various testing conditions to simulate, to some extent, the environment of the production network and to comprehensively analyze and compare the performance of these implementations (e.g., there could be a discussion on how changing the values of some test parameters could influence their performance and, eventually, how the poor or high performances influence the overall performance).

Two types of test systems were installed: a loopback test system and a Tester-DUT test system. In the loopback test system, the two 10 G network interfaces of a Dell PowerEdge R720 server were interconnected via a direct cable, as shown in Fig. 6, to measure the

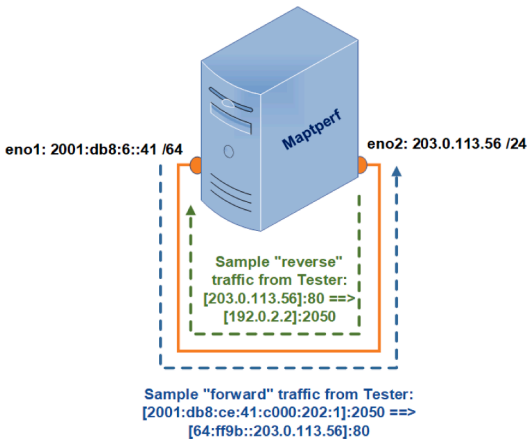

Fig. 6. Loopback test system.

performance of Maptperf through self-tests. On the other hand, the Tester-DUT test system was installed by connecting the two 10 G network interfaces of another Dell PowerEdge R720 server (representing the Tester) to their counterparts of a Dell PowerEdge R730 server (representing the DUT), as shown in Fig. 7, to measure the performance of the DUT.

8.1. Performance test results

As a first step to verify the consistency and stability of Maptperf, in addition to measuring its performance, the loopback test system was used. The experiments were run based on the parameter settings of the `Maptperf.conf` configuration file whose contents are set as in Appendix A.3.

Table 2 shows the results of 20 runs of the Maptperf-tp self-test throughput experiment with one million simulated CEs using bidirectional traffic, different IPv6 frame sizes (IPv4 is 20 bytes less by default), 60 seconds test duration, and 2000 milliseconds stream timeout. While Table 3 shows the results of the special-case throughput test of Maptperf-pdv using the same beforementioned configuration settings and with the frame timeout set to 100 milliseconds. The dispersion in the tables was calculated as in (3).

$$Dispersion(\%) = \frac{99th\ Percentile - 1st\ Percentile}{Median} \times 100\% \quad (3)$$

The results proved that Maptperf is capable of achieving high throughput rates, compared to the theoretical maximum frame rates achieved by the Ethernet interface for the tested frame sizes, which are referred to in Appendix A.1 of RFC 5180 [12]. As a result, the satisfying performance of Maptperf allows for testing powerful MAP-T BR implementations without forming a bottleneck in the test. The bottleneck of the Tester can only happen when the performance of the tested MAP-T BR implementation is so high that Maptperf cannot handle it. That is, the MAP-T BR implementation can yield higher throughput rates than that achieved by Maptperf for the specified settings.

Table 2
Maximum frame rate achieved after 20 runs of Maptperf-tp self-test experiment for different frame sizes using bidirectional traffic.

IPv6 Frame size (bytes)	1st Percentile (fps)	99th Percentile (fps)	Median (fps)	Dispersion (%)
84	6,894,530	6,904,789	6,903,988	0.14860
148	5,767,693	5,769,558	5,768,792	0.03231
276	4,223,186	4,223,193	4,223,188	0.00016
532	2,264,614	2,264,618	2,264,618	0.00015
1044	1,174,883	1,174,886	1,174,885	0.00025

Table 3
Maximum frame rate achieved after 20 runs of Maptperf-pdv self-test experiment for different frame sizes using bidirectional traffic and 100 milliseconds frame timeout.

IPv6 Frame size (bytes)	1st Percentile (fps)	99th Percentile (fps)	Median (fps)	Dispersion (%)
84	4,115,233	4,384,315	4,382,761	6.13
148	3,793,831	4,208,788	4,076,424	10.17
276	2,845,845	3,227,117	2,936,304	12.98
532	1,788,122	2,072,432	1,958,553	14.51
1044	1,124,602	1,174,858	1,174,826	4.27

8.2. Functional test results

After verifying that Maptperf was functional with high enough performance through running the performance test experiments, it was deployed as the Tester program in the Tester-DUT test experiments, where the MAP-T BR implementation of Jool [10] was run at the DUT. These experiments were run using the test system shown in Fig. 7 and based on the parameter settings of the `Maptperf.conf` configuration file whose contents are set as in Appendix A.4.

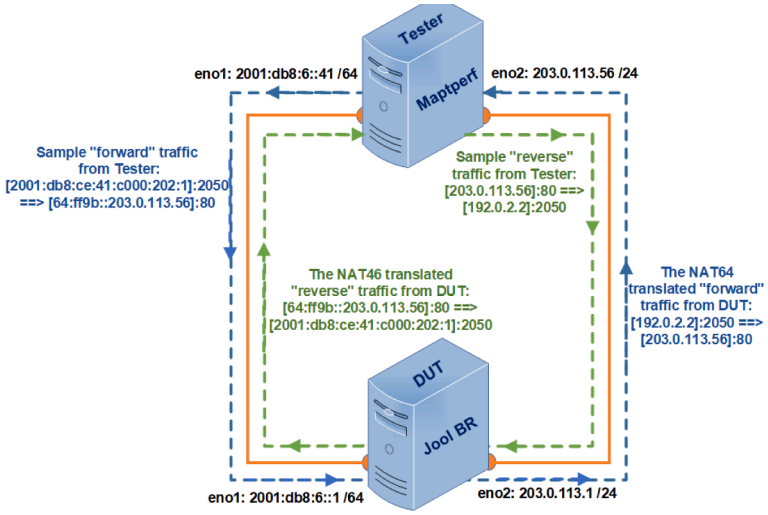

Fig. 7. Tester-DUT test system.

```
5 6.669726595 2001:db8:ce:1815:0:c000:2c0:15 → 64:ff9b::cb00:7138 UDP 80 43195 → 30985 Len=18
6 6.669737665 64:ff9b::cb00:7138 → 2001:db8:ce:1815:0:c000:2c0:15 UDP 80 57735 → 43082 Len=18
7 6.674719300 2001:db8:ce:1596:0:c000:2ac:16 → 64:ff9b::cb00:7138 UDP 80 46822 → 22544 Len=18
8 6.674737547 64:ff9b::cb00:7138 → 2001:db8:ce:1596:0:c000:2ac:16 UDP 80 11170 → 46116 Len=18
9 6.679718756 2001:db8:ce:1e4b:0:c000:2f2:b → 64:ff9b::cb00:7138 UDP 80 23551 → 45566 Len=18
10 6.679724230 64:ff9b::cb00:7138 → 2001:db8:ce:1e4b:0:c000:2f2:b UDP 80 11196 → 23081 Len=18
11 6.684721531 2001:db8:ce:46f:0:c000:223:f → 64:ff9b::cb00:7138 UDP 80 30914 → 15447 Len=18
12 6.684724746 64:ff9b::cb00:7138 → 2001:db8:ce:46f:0:c000:223:f UDP 80 18145 → 32465 Len=18
```

Fig. 8. Sample tshark capture at eno1 interface of the DUT (Jool).

```
5 0.000004858   192.0.2.42 → 203.0.113.56 UDP 60 9798 → 11318 Len=18
6 0.000005113 203.0.113.56 → 192.0.2.35   UDP 60 39255 → 12919 Len=18
7 0.000005402  192.0.2.154 → 203.0.113.56 UDP 60 26329 → 39447 Len=18
8 0.000007485 203.0.113.56 → 192.0.2.42   UDP 60 6778 → 9190 Len=18
9 0.000008330 203.0.113.56 → 192.0.2.113  UDP 60 6798 → 15999 Len=18
10 0.000008895   192.0.2.35 → 203.0.113.56 UDP 60 12855 → 24661 Len=18
11 0.000009213 203.0.113.56 → 192.0.2.223  UDP 60 52013 → 9438 Len=18
12 0.000009267 203.0.113.56 → 192.0.2.71   UDP 60 52125 → 17406 Len=18
13 0.000011297  192.0.2.223 → 203.0.113.56 UDP 60 9249 → 6562 Len=18
14 0.000011900   192.0.2.71 → 203.0.113.56 UDP 60 16840 → 37655 Len=18
```

Fig. 9. Sample tshark capture at eno2 interface of the DUT (Jool).

Figs. 8 and 9 show the "tshark" capture of the traffic at both interfaces of the DUT, eno1 and eno2, respectively. They prove the proper use of the MAP rules by the Tester and the valid translation behavior of the DUT in both directions. For example, packet 5, sent by the Tester and captured by the eno1 interface (which is an IPv6 interface), carried the source port number 43,195, which belongs to the port set (43,008–45,055), identified by the PSID 0 × 15 (21 in decimal), which is part of the MAP address 2001:db8:ce:18**15**:0:c000:2c0:**15**. To compute the port set, it is known from the configuration file that EA length is 13 bits and IPv4 suffix length is 8 bits. These leave 5 bits for the PSID, which results in 32 port sets, each of 2048 ports. Thus, the PSID 21 (in decimal) will include all ports from 21 × 2048 = 43,008 to (22 × 2048)-1 = 45,055. Packet 6 is a translated packet outgoing from the eno1 interface with destination port 43,082, which is in the same previous port set. The port numbers are different because the Tester pseudorandomly generated them.

All the following functional test experiments were run 20 times, and the median value was considered for evaluation. They use bidirectional traffic and stream timeout of 2000 milliseconds. The test duration in the throughput, FLR, and PDV test experiments was 60 seconds, while it was 120 seconds in the latency test experiments.

8.2.1. *Throughput test results*

Table 4 shows the 1st percentile, 99th percentile, and median values for the maximum frame rate achieved by Jool in the throughput test using five different IPv6 frame sizes: 84, 148, 276, 532, and 1044 (the IPv4 frame size is always 20 bytes less). The dispersion value is also included to show how much the results are consistent or scattered.

The results showed slight performance degradation for Jool as the test frame size increased. However, it should be noted that the frame rates shown in the table were the same for both directions. Thus it can be inferred that the cumulative number of frames being translated and forwarded by Jool during the test duration was double the outcome of the rates in the table.

Table 4
Throughput of Jool achieved after 20 runs of Maptperf-tp Tester-DUT experiment for different frame sizes using bidirectional traffic.

IPv6 Frame size (bytes)	1st Percentile (fps)	99th Percentile (fps)	Median (fps)	Dispersion (%)
84	845,205	861,149	853,175	1.86
148	837,550	852,638	844,578	1.78
276	829,416	847,028	839,222	2.09
532	826,152	836,357	831,469	1.22
1044	808,159	825,651	813,856	2.14

The following two main observations can be concluded from this test:

- With the use of the same test settings, the throughput rates achieved by Maptperf in the self-test experiments (i.e., recorded in Table 2) are much higher than what Jool produced. This highlights the capability of Maptperf to benchmark MAP-T implementations which are more powerful than Jool.
- The performance of Jool was relatively stable despite the slight degradation in the throughput rates against the increase in the test frame size.

8.2.2. *FLR test results*

RFC 8219 [1] reused the definition of FLR mentioned in section 26.3 of RFC 2544 [11], which requires starting testing from the maximum frame rate of the media and then decreasing it (at most) by 10% in each consecutive step. However, in practice, the actual throughput rate is much lower than these required starting rates. So, typical rates were relied on, which could give more meaningful results.

Fig. 10 shows the results of the FLR test when the frame rate starts

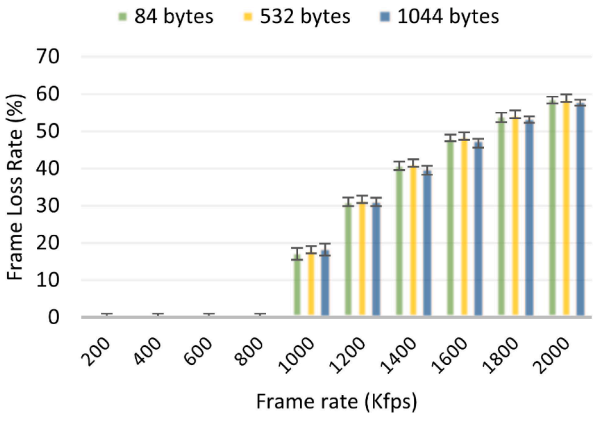

Fig. 10. FLR of Jool MAP-T BR.

Table 5
Latency of Jool after 20 runs of Maptperf-lat Tester-DUT experiment for a frame size of 84 bytes at a frame rate of 853,175 frames per second using bidirectional traffic.

	Forward TL	ForwardWCL	Reverse TL	Reverse WCL
1st Percentile (ms)	0.02266	0.03711	0.02172	0.03604
99th Percentile (ms)	0.02311	0.04443	0.02216	0.04379
Median (ms)	0.02290	0.03831	0.02197	0.03801
Dispersion (%)	1.99	19.10	2.00	20.38

Table 6
PDV of Jool after 20 runs of Maptperf-pdv Tester-DUT experiment for a frame size of 84 bytes at a frame rate of 853,175 frames per second using bidirectional traffic.

	Forward PDV	Reverse PDV
1st Percentile (ms)	0.02386	0.02175
99th Percentile (ms)	0.02772	0.02776
Median (ms)	0.02538	0.02434
Dispersion (%)	15.23	24.69

from 200,000 frames per second until it reaches 2000,000 frames per second with an increase of 200,000 frames per second in each consecutive run. Each bar in the figure represents the median FLR value for the corresponding used IPv6 frame size (84, 532, or 1044 bytes).

The results proved that Jool was losing more frames in a relatively congruent manner when increasing the frame rate regardless of the frame size. This is rational. As more test frames need to be translated by Jool, more frames could not be served within the test time limits due to exceeding the capacity of Jool, and, consequently, more frame loss could happen.

8.2.3. Latency test results

Table 5 shows the 1st percentile, 99th percentile, median, and dispersion values for TL and WCL measurements in each direction of the latency test of Jool using an IPv6 frame size of 84 bytes and a frame rate of 853,175 frames per second, which is the median frame rate achieved by the throughput test for that IPv6 frame size. In each experiment, the number of latency tagged frames was 50,000 and they were sent after passing 60 seconds from the start of the test according to the uniform time distribution.

The results proved a consistent performance of Jool in both directions in terms of the delay incurred during the translation of IPv6 packets to IPv4 packets and vice versa using MAP rules. This can be justified by the fact that the increase of the frame size (whether IPv6 or IPv4 test frame) occurs in the data field and not in the header, whose size is fixed regardless of the entire frame size. And since the translation activity is related to the frame header, its processing time is almost the same for all frame sizes.

8.2.4. PDV test results

Table 6 shows the 1st percentile, 99th percentile, median, and dispersion values of the PDV test of Jool in both directions using an IPv6 frame size of 84 bytes and a frame rate of 853,175 frames per second, which was the median frame rate achieved by the throughput test for that IPv6 frame size. The frame timeout was set to 0 to enable the PDV measurement.

Again, the results reflected a relatively congruous performance of Jool in both directions in terms of the variations of packet delays in each active direction. The same reason related to the latency test results could justify the PDV test results.

9. Future work

For the next step of research work, the authors plan to perform the following main tasks:

- Performing comprehensive benchmarking of Jool and other MAP-T implementations using Maptperf and comparing their performance based on their collected test results. This can be done using more than one testing system to generate a more thorough analysis, (e.g., to distinguish the inherent behavior of an implementation from the effects of a given CPU architecture).
- Using Maptperf as a model for developing benchmarking Testers for other IPv4aaS technologies, especially the MAP-E technology, which deploys a similar concept of using MAP rules but with encapsulation functionality.
- Optimizing the performance of Maptperf even more by adding some other functionalities, such as the capability to reply to neighbor solicitation messages advertised by the DUT to allow dynamic additions of CE information into the neighbor table. Additionally, Maptperf-pdv can be extended to measure the optional IPDV measurement.

10. Conclusions

This paper presented the design and implementation details of a software Tester for benchmarking the MAP-T BR router based on the guidelines and recommendations of the IETF RFC 8219 [1]. It also disclosed several operational requirements and design considerations that had been adhered to, followed by a list of design and implementation decisions that had been taken during the development process.

It can be concluded that the results obtained from the benchmarking tests proved that Maptperf is reliable and efficient enough to benchmark various MAP-T implementations, which also gives a solid basis to use it as a model for developing new Testers for the devices of other IPv6 transition technologies.

CRediT authorship contribution statement

Ahmed Al-hamadani: Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Conceptualization. **Gábor Lencse:** Writing – review & editing, Supervision, Resources, Project administration, Methodology, Conceptualization.

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.

Acknowledgement

The resources of NICT StarBED, 2-12 Asahidai, Nomi-City, Ishikawa 923-1211, Japan were used remotely for the original development of Maptperf.

The authors would like to thank Shuuhei Takimoto for the possibility of using StarBED.

The authors thank Bertalan Kovács for reading and commenting on the manuscript.

Appendix

A.1. Description of configuration file parameters

Configuration File Parameters	Description
Basic Parameters	
Tester-L-IPv6	The IPv6 address of the left-side interface of the Tester. It should be in the same subnet as that of the DUT-L-IPv6. It is used only for the background traffic. For the MAP-T traffic, it is replaced by the MAP address of the simulated CE.
Tester-R-IPv4	The IPv4 address of the right-side interface of the Tester. It should be in the same subnet as that of the DUT-R-IPv4. This will simulate an IPv4 server.
Tester-R-IPv6	The IPv6 address that will be used by the Tester for forwarding the background (i.e., non-translated) traffic via the right-side interface.
Tester-L-MAC	The MAC address of the left-side interface of the Tester.
Tester-R-MAC	The MAC address of the right-side interface of the Tester.
DUT-L-MAC	The MAC address of the left-side interface of the DUT.
DUT-R-MAC	The MAC address of the right-side interface of the DUT.
FW-dport-min	The lower limit value of the destination port range that can be used by the forward test packets of the foreground traffic.
FW-dport-max	The upper limit value of the destination port range that can be used by the forward test packets of the foreground traffic.
RV-sport-min	The lower limit value of the source port range that can be used by the reverse test packets of the foreground traffic.
RV-sport-max	The upper limit value of the source port range that can be used by the reverse test packets of the foreground traffic.
bg-sport-min	The lower limit value of the source port range that can be used by the background test packets.
bg-sport-max	The upper limit value of the source port range that can be used by the background test packets.
bg-dport-min	The lower limit value of the destination port range that can be used by the background test packets.
bg-dport-max	The upper limit value of the destination port range that can be used by the background test packets.
FW-var-sport	How the source port numbers vary within their range when sending forward test frames. Possible values are 1 for increase, 2 for decrease, or 3 for pseudo-randomly change.
FW-var-dport	How the destination port numbers vary within their range when sending forward test frames. Possible values are 1 for increase, 2 for decrease, or 3 for pseudo-randomly change.
RV-var-sport	How the source port numbers vary within their range when sending reverse test frames. Possible values are 1 for increase, 2 for decrease, or 3 for pseudo-randomly change.
RV-var-dport	How the destination port numbers vary within their range when sending reverse test frames. Possible values are 1 for increase, 2 for decrease, or 3 for pseudo-randomly change.
MAP Rules Parameters	
NUM-OF-CEs	The number of CEs to be simulated in the test.
BMR-IPv6-Prefix	The rule IPv6 prefix of the BMR that will be shared among all CEs of the same MAP domain.
BMR-IPv6-Prefx-Length	The number of bits in the MAP address that will be reserved for the BMR-IPv6-Prefix.
BMR-IPv4-Prefix	The public IPv4 prefix of the BMR that will be shared among all CEs of the same MAP domain.
BMR-IPv4-Prefx-Length	The number of bits of the BMR-IPv4-Prefix.
BMR-EA-Length	The number of EA-bits (i.e., IPv4 suffix + PSID) in the MAP address.
DMR-IPv6-Prefix	The IPv6 prefix of the DMR that will be provisioned by the corresponding BR to the simulated CE.
DMR-IPv6-Prefx-Length	The number of bits in the DMR address (i.e., the IPv4-embedded IPv6 address) that will be reserved for the DMR-IPv6-Prefix.
Device Hardware Parameters	
CPU-FW-Send	The ID of the CPU core to be used by the forward sender thread.
CPU-FW-Receive	The ID of the CPU core to be used by the forward receiver thread.
CPU-RV-Send	The ID of the CPU core to be used by the reverse sender thread.
CPU-RV-Receive	The ID of the CPU core to be used by the reverse receiver thread.
Mem-Channels	The number of memory channels to be used.
Network Traffic Parameters	
FW	A switch flag to enable or disable testing in the forward direction. Possible values are 0 for inactive and 1 for active.
RV	A switch flag to enable or disable testing in the reverse direction. Possible values are 0 for inactive and 1 for active.
Promisc	A switch flag to enable or disable testing in the promiscuous mode. Possible values are 0 for inactive and 1 for active. It is only set when needed for the sake of fixing some testing problems (e.g., using incorrect MAC addresses or generating bad checksum).

A.2. Description and usage of command line arguments

Command Line Argument (unit)	Description	Usage
IPv6 frame size (bytes).	The size of the IPv6 test frames. It should be set according to section 5.1.1 of RFC 8219 [1]. The IPv4 frames will automatically be 20 bytes shorter.	Maptperf-tp, Maptperf-lat, Maptperf-pdv
Frame rate (frames per second).	The rate at which the test frames will be transmitted.	Maptperf-tp, Maptperf-lat, Maptperf-pdv
Test duration (seconds).	The time duration of a single experiment. Section 24 of RFC 2544 [2] specifies the lower limit value to 60, while Maptperf sets the upper limit value to 3600.	Maptperf-tp, Maptperf-lat, Maptperf-pdv
Stream timeout (milliseconds).	How long the Tester should wait before stopping receiving test frames after sending them completely? This parameter could be compared to the 2000 milliseconds “after sending timeout” recommended by RFC 2544 [2] in its section 23 and complied with RFC 8219 [1] recommendations.	Maptperf-tp, Maptperf-lat, Maptperf-pdv
n & m (n/a)	Two relatively prime numbers to specify the proportions of the foreground and background frames.	Maptperf-tp, Maptperf-lat, Maptperf-pdv
First tagged delay (seconds)	The delay from the beginning of the test until the sending of the first tagged frame. Section 7.2 of RFC 8219 [1] specifies the lower limit value to 60, while Maptperf sets the upper limit value to 3600.	Maptperf-lat
Number of tagged frames (n/a)	The number of tagged frames (i.e., frames with timestamps). Section 7.2 of RFC 8219 [1] requires at least 500, while Maptperf sets the upper limit value to 50,000.	Maptperf-lat
Frame timeout (milliseconds)	The frame will be considered “lost” if its delay is greater than the value of this parameter. The value of 0 means that no per-frame timeout is used.	Maptperf-pdv

A.3. The `Maptperf.conf` configuration file to be used in the self-test

```
# Maptperf.conf (to be used in the self-test)
# Basic parameters
Tester-L-IPv6 2001:db8:6::41
Tester-R-IPv4 203.0.113.56
Tester-R-IPv6 2001:db8:42::2 # for background traffic
Tester-L-MAC ec:f4:bb:dd:07:28 # ST eno1
Tester-R-MAC ec:f4:bb:dd:07:2a # ST eno2
DUT-L-MAC ec:f4:bb:dd:07:2a # ST eno2
DUT-R-MAC ec:f4:bb:dd:07:28 # ST eno1
# Port selection parameters
# Some port range boundary values
FW-dport-min 1 # as RFC4814 recommends
FW-dport-max 49,151 # as RFC4814 recommends
RV-sport-min 1024 # as RFC4814 recommends
RV-sport-max 65,535 # as RFC4814 recommends
bg-dport-min 1 # as RFC4814 recommends
bg-dport-max 49,151 # as RFC4814 recommends
bg-sport-min 1024 # as RFC4814 recommends
bg-sport-max 65,535 # as RFC4814 recommends
# How port numbers vary? 1:increase, 2:decrease, 3:random
FW-var-sport 3
FW-var-dport 3
RV-var-sport 3
RV-var-dport 3
# MAP rules parameters
NUM-OF-CEs 1000,000 # Number of simulated CEs in the test
BMR-IPv6-Prefix 2001:db8:ce::
BMR-IPv6-Prefix-Length 51
BMR-IPv4-Prefix 192.0.2.0
BMR-IPv4-Prefix-Length 24
BMR-EA-Len 20
DMR-IPv6-Prefix 64:ff9b::
DMR-IPv6-Prefix-Length 96
# Device hardware parameters
CPU-FW-Send 2 # Forward Sender runs on this core
CPU-FW-Receive 4 # Forward Receiver runs on this core
CPU-RV-Send 6 # Reverse Sender runs on this core
CPU-RV-Receive 8 # Reverse Receiver runs on this core
Mem-Channels 2
# Network traffic parameters
FW 1 #Forward direction (0:inactive ; 1:active)
RV 1 #Reverse direction (0:inactive ; 1:active)
Promisc 0 #Promiscuous mode (0:inactive ; 1:active)
```

A.4. The Maptperf.conf configuration file to be used in the Tester-dut test

```
# Maptperf.conf (to be used at the MAP-T BR Tester)
# Basic parameters
Tester-L-IPv6 2001:db8:6::41
Tester-R-IPv4 203.0.113.56
Tester-R-IPv6 2001:db8:42::2 # for background traffic
Tester-L-MAC ec:f4:bb:ef:98:a0 # Tester eno1
Tester-R-MAC ec:f4:bb:ef:98:a2 # Tester eno2
DUT-L-MAC ec:f4:bb:dc:a6:b8 # DUT eno1
DUT-R-MAC ec:f4:bb:dc:a6:ba # DUT eno2
# Port selection parameters
# Some port range boundary values
FW-dport-min 1 # as RFC4814 recommends
FW-dport-max 49,151 # as RFC4814 recommends
RV-sport-min 1024 # as RFC4814 recommends
RV-sport-max 65,535 # as RFC4814 recommends
bg-dport-min 1 # as RFC4814 recommends
bg-dport-max 49,151 # as RFC4814 recommends
bg-sport-min 1024 # as RFC4814 recommends
```

```
bg-sport-max 65,535 # as RFC4814 recommends
# How port numbers vary? 1:increase, 2:decrease, 3:random
FW-var-sport 3
FW-var-dport 3
RV-var-sport 3
RV-var-dport 3
# MAP rules parameters
NUM-OF-CEs 1000 # Number of simulated CEs in the test
BMR-IPv6-Prefix 2001:db8:ce::
BMR-IPv6-prefix-length 51
BMR-IPv4-Prefix 192.0.2.0
BMR-IPv4-prefix-length 24
BMR-EA-length 13
DMR-IPv6-Prefix 64:ff9b::
DMR-IPv6-prefix-length 96
# Device hardware parameters
CPU-FW-Send 2 # Forward Sender runs on this core
CPU-FW-Receive 4 # Forward Receiver runs on this core
CPU-RV-Send 6 # Reverse Sender runs on this core
CPU-RV-Receive 8 # Reverse Receiver runs on this core
Mem-Channels 2
# Network traffic parameters
FW 1 #Forward direction (0:inactive ; 1:active)
RV 1 #Reverse direction (0:inactive ; 1:active)
Promisc 0 #Promiscuous mode (0:inactive ; 1:active)
```

A.5. Implementation details of Maptperf

The object-oriented design of the Tester is fairly simple and flexible. It is implemented in C++ as three main classes: one base class, the *Throughput* class, and two derived classes from the throughput class, the *Latency* and *PDV* classes. In addition, there exist some other classes that perform certain tasks for the Tester other than the measurement testing itself such as packing parameters for the send and receive functions. The three main classes call some DPDK APIs [30] to perform specific functionalities as this user-space networking framework is popular in offering fast packet processing and efficient buffer management. The source code of Maptperf along with the shell scripts, configuration file, Make file, and other files can be accessed on GitHub [31].

What follows is a walkthrough of the experiment procedure for each one of the test binaries:

A.5.1. Maptperf-tp experiment workflow

As stated earlier, Maptperf-tp measures throughput. Hence, it mainly depends on the "Throughput" class to accomplish its task. First, the `readConfigFile()` member function reads the configuration parameter values stored in the "`Maptperf.conf`" configuration file and sets their corresponding data members in the Throughput class accordingly. Then, the `readCmdLine()` member function reads the command line parameter values passed by the corresponding throughput script and sets their corresponding data members in the Throughput class accordingly. Once the parameter values are set, the Maptperf-tp calls the `init()` member function, which performs the following tasks:

- It initializes the Environment Abstraction Layer (EAL) of DPDK, which provides seamless access to low-level resources such as hardware and memory through a generic interface that hides the environment specifics from the applications and library functions [32].
- It creates the packet pools of both the sender and the receiver threads for both directions (i.e., forward and reverse) after calculating their appropriate sizes.
- It installs the TX/RX queues of the network interfaces.
- It checks, configures, and then starts the network interfaces.
- It checks the states of the links.
- It performs some other sanity checks such as whether there is a match between the Non-Uniform Memory Access (NUMA) node of the CPU cores and of the network interfaces or not and whether the Time Stamp Counters (TSCs) of the deployed CPU cores are synchronized or not.
- It prepares the timing values needed during the test like the number of clock cycles per second, the start time of sending the frames, and the end time of receiving the frames.
- It calculates essential MAP information required to build the MAP CE array and run the test properly.
- Finally, it calls the `buildMapArray()` member function to build the MAP CE array (please refer to Section 7.2.2).

Next, the `measure()` member function packs all the required parameters for the sender threads in all active directions (i.e., forward and reverse) via instantiating the `senderCommonParameters` class and the `senderParameters` class and packs all the required parameters for the receiver threads in both directions via instantiating the `receiverParameters` class. The `senderCommonParameters` class involves those sending parameters whose values are the same for testing in both directions, whereas the `senderParameters` class involves those sending parameters whose values are different. Then, the `measure()` function launches the sender and receiver threads for the active direction(s) by calling the `rte_eal_remote_launch()` DPDK API twice: one for the sender thread and passing the appropriate sender parameter values to its related `send()` function and another for the receiver thread and passing the proper receiver parameter values to its related `receive()` function.

As a next step, the `send()` function starts by creating an N-sized buffer of template foreground test packets and another N-sized buffer of template background test packets, where N is a predefined integer number specifying the number of template test packets in the buffer. The reason behind

creating N copies of template test packets and not a single one is to avoid facing the "rewrite after send" problem that has been experienced by `siitperf` before. The `rte_eth_tx_burst()` DPDK API was reporting that some frames were sent, but actually, they were still in their sending buffer, and it caused the problem that they were rewritten during the preparation of the next frame to send. For more information about this problem, please refer to section 3.6.5 of [20]. The sender thread will cycle through these N copies of packets. Depending on the active direction, the foreground frames could be IPv6 packets (in the case of the forward direction) or IPv4 packets (in the case of the reverse direction), while all test packets in the background buffer are native IPv6. For each of the test packets, any header field that needs to be manipulated before sending the packet will be accessed by a pointer inside the sending cycle. This includes the following fields:

- The IPv6 source address of the foreground packet to be sent in the forward direction, as this will be the MAP address of the simulated sending CE. This address will be constructed later in the sending loop of the `send()` function (i.e., it will be specified dynamically during the sending loop cycle, as described in Section 7.2.3).
- The destination IPv4 address of the foreground packet which is to be sent in the reverse direction as this will represent one of the targeted receiving CEs.
- The IPv4 header checksum of the foreground packet in the reverse direction as its calculation depends on the change occurring to the destination IPv4 address field.
- The UDP source port number, the UDP destination port number, and the UDP checksum fields of all types of test packets as the UDP port numbers will vary, and the UDP checksum should be changed accordingly. Additionally, the UDP source port number of the foreground packet in the forward direction will be selected according to the PSID of the simulated CE, and this will only be identified inside the sending loop of the `send()` function. This would also be applied to the UDP destination port number of the foreground packet in the reverse direction. As a result of all of that, the UDP checksum should be recalculated.

The sender thread keeps sending the test frames until their number reaches the (*Test duration* * *Frame rate*) value. This number, then, will be reported to be "grep"-ed by the corresponding measurement script for further processing.

For the test packets to be distinguished from other packets in the test network, the sender thread will insert a verifiable string "IDENTIFY" ASCII-encoded as a 64-bit integer in the first 8 octets of the UDP data payload. The `receive()` function will recognize the test frames by checking if this string exists in the payload of the received UDP datagrams. This makes the `receive()` function fast and resilient in processing and verifying the received packets. Similarly, it will report how many test frames are received to be "grep"-ed by the corresponding measurement script for further processing.

A.5.2. Maptperf-lat experiment workflow

The latency measurement is implemented by the *Maptperf-lat* test binary. It mainly depends on the "*Latency*" class, which is derived from the "*Throughput*" base class, and its implementation is fairly similar to that of the throughput test. The main differences exist in the `send()` and `receive()` functions. Here, a certain *T* number of test frames are tagged to be involved in the latency measurement. These latency test frames are also pre-generated and put into a specific buffer ahead of the sending cycle. To differentiate latency test frames from other frames in the network, including normal test frames, the sender thread will insert a verifiable string "Identify" ASCII-encoded as a 64-bit integer in the first 8 octets of the UDP data payload (it is different from the counterpart "IDENTIFY" string of the normal test frames). The first tagged frame will be sent after a delay specified by the *First tagged delay* parameter value. Then, the latency test frames are uniformly distributed in the remaining test duration interval.

Once the sender thread sends a specific latency test frame, it records the sending timestamp using the `rte_rdtsc()` DPDK API. The receiver thread (through the `receive()` function), in contrast, will check the first 8 octets of the payload of every received frame to verify if it contains the "Identify" tag. Whenever the tag is found, the corresponding frame will be validated as a latency test frame, and its receiving timestamp will be recorded. The recorded sending and receiving timestamps of all latency frames will then be used by the `evaluateLatency()` function to calculate the Typical Latency (TL) and the Worst-Case Latency (WCL) for the active direction/s (forward and/or reverse), as described in Section 7.1.2, and prepare them for collection by the latency shell script for summary evaluation.

A.5.3. Maptperf-pdv experiment workflow

The *Maptperf-pdv* test binary is based on the "*Pdv*" class, which is derived from the "*Throughput*" base class. Here, the sent test frames will not be tagged, instead, *each* of them will have a unique 64-bit integer identifier. The sender thread will insert the value of this identifier (typically, the ordinal number of the test frame) into the second eight octets of the payload after the "IDENTIFY" string. This identifier will be used later as an index in the arrays which store the sending and receiving timestamps of the test frames for the evaluation of their delays. Once the sender thread sends a test frame, it records the timestamp of complete sending and stores it in the element of the sending array whose index is equal to the identifier of the sent frame. When the receiver thread receives a PDV test frame, it uses its 64-bit identifier as an index in the receiving array to determine where to store the receiving timestamp.

Next, the `evaluatePdv()` function reads the value of the "*Frame timeout*" command line parameter to decide whether to calculate the PDV (if it is "0") or to calculate a "special case" throughput (if it is higher than "0"). In the latter case, the delay of each test frame (i.e., the difference between the receiving timestamp and the sending timestamp values) will be checked individually against the value of the "*Frame timeout*", as described in Section 7.1.3. Any frame whose delay value is greater than the "*Frame timeout*" value will be counted as a "lost" frame.

To calculate the PDV, the `evaluatePdv()` function will specify two values: the lowest recorded delay (D_{min}) and the 99.9th percentile delay ($D_{99.9th\_perc}$). The PDV will, then, represent their difference as in (2).

$$PDV = D_{99.9th\_perc} - D_{min} \tag{2}$$

Finally, the PDV shell script will "grep" the PDV values in the active direction(s) for summary evaluation.

Data availability

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

References

[1] M. Georgescu, L. Pislaru, and G. Lencse, "Benchmarking methodology for IPv6 transition technologies", IETF RFC 8219, Aug. 2017, doi:10.17487/RFC8219.

[2] G. Lencse, J.P. Martinez, L. Howard, R. Patterson, and I. Farrer, "Pros and cons of IPv6 transition technologies for IPv4-as-a-service (IPv4aaS)", IETF RFC 9313, Oct. 2022, doi:10.17487/RFC9313.

[3] M. Mawatari, M. Kawashima, and C. Byrne, "464XLAT: combination of stateful and stateless translation", IETF RFC 6877, Apr. 2013, doi:10.17487/RFC6877.

[4] A. Durand, R. Droms, J. Woodyatt, and Y. Lee, "Dual-stack lite broadband deployments following IPv4 exhaustion", IETF RFC 6333, 2011, doi:10.17487/RFC6333.

[5] Y. Cui, Q. Sun, M. Boucadair, T. Tsou, Y. Lee et al., "Lightweight 4over6: an extension to the dual-stack lite architecture", IETF RFC 7596, 2015, doi:10.17487/RFC7596.

[6] E.O. Troan, W. Dec, X. Li, C. Bao, S. Matsushima et al., "Mapping of address and port with encapsulation (MAP-E)", IETF RFC 7597, 2015, doi:10.17487/RFC7597.

[7] X. Li, C. Bao, E.W. Dec, O. Troan, S. Matsushima et al., "Mapping of address and port using translation (MAP-T)", IETF RFC 7599, 2015, doi:10.17487/RFC7599.

[8] 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, https://doi.org/10.1002/dac.5354.

[9] A. Al-hamadani, G. Lencse, Towards implementing a software tester for benchmarking MAP-T devices, Infocommun. J. 14 (3) (2022) 45–54, https://doi.org/10.36244/ICJ.2022.3.6.

[10] Jool. "Jool MAP-T Summary"; 2024 https://www.jool.mx/en/map-t.html.

[11] S. Bradner, and J. McQuaid, "Benchmarking methodology for network interconnect devices", IETF RFC 2544, Mar. 1999, doi:10.17487/RFC2544.

[12] C. Popoviciu, A. Hamza, G.V. d. Velde, and D. Dugatkin, "IPv6 benchmarking methodology for network interconnect devices", IETF RFC 5180, May. 2008, doi:10.17487/RFC5180.

[13] P. Srisuresh, and M. Holdrege, "IP network address translator (NAT) terminology and considerations", IETF RFC 2663, 1999, doi:10.17487/RFC2663.

[14] R. B. (ed.), "The Address plus Port (A+P) approach to the IPv4 address shortage", IETF RFC 6346, 2011, doi:10.17487/RFC6346.

[15] M. Georgescu, H. Hazeyama, T. Okuda, Y. Kadobayashi, S. Yamaguchi, Benchmarking the load scalability of IPv6 transition technologies: a black-box analysis, in: Proceedings of the IEEE Symposium on Computers and Communication (ISCC), 2015, pp. 329–334, https://doi.org/10.1109/ISCC.2015.7405536.

[16] S. Avallone, S. Guadagno, D. Emma, A. Pescape, G. Ventre, D-ITG distributed Internet traffic generator, in: Proceedings of the First International Conference on the Quantitative Evaluation of Systems, 2004, pp. 316–317, https://doi.org/10.1109/QEST.2004.1348045.

[17] M. Bagnulo, A. Sullivan, P. Matthews, and I.V. Beijnum, "DNS64: DNS extensions for network address translation from IPv6 clients to IPv4 servers", IETF RFC 6147, Apr. 2011, doi:10.17487/RFC6147.

[18] G. Lencse, D. Bakai, Design and implementation of a test program for benchmarking DNS64 servers, IEICE Trans. Commun. 100 (6) (2017) 948–954, https://doi.org/10.1587/transcom.2016EBN0007.

[19] C. Bao, X. Li, F. Baker, T. Anderson, F. Gont, IP/ICMP Translation Algorithm, IETF RFC (2016) 7915, https://doi.org/10.17487/RFC7915.

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

[21] G. Lencse, Adding RFC 4814 random port feature to siitperf: design, implementation and performance estimation, Int. J. Adv. Telecommun. Electrotech. Signals Syst. 9 (3) (2020), https://doi.org/10.11601/ijates.v9i3.291.

[22] D. Newman, and T. Player, "Hash and stuffing: overlooked factors in network device benchmarking", IETF RFC 4814, Mar. 2007, doi:10.17487/RFC4814.

[23] 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. 192 (2022) 75–88, https://doi.org/10.1016/j.comcom.2022.05.028.

[24] G. Lencse, Á. Bazsó, Benchmarking methodology for IPv4aaS technologies: comparison of the scalability of the Jool implementation of 464XLAT and MAP-T, Comput. Commun. 219 (2024) 243–258, https://doi.org/10.1016/j.comcom.2024.03.007.

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

[26] R. Durstenfeld, Algorithm 235: random permutation, Commun. ACM 7 (7) (1964) 420, https://doi.org/10.1145/364520.364540.

[27] O.D. Arbeláez. "How competitive are C++ standard random number generators"; 2024 https://medium.com/@odarbelaeze/how-competitive-are-c-standard-random-number-generators-f3de98d973f0.

[28] Intel. "Intel 64 and IA-32 architectures software developer's manual," Volume 2B: instruction Set Reference, M-U, Order Number: 253667-060US"; 2024 https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-2b-manual.pdf.

[29] G. Lencse, Enabling dns64perf++ for benchmarking the caching performance of DNS64 servers, J. Comput. Inf. Technol. 26 (1) (2018) 19–28, https://doi.org/10.20532/cit.2018.1004078.

[30] DPDK. "DPDK documentation"; 2024 https://core.dpdk.org/doc/.

[31] A. Al-Hamadani. "Maptperf: an RFC 8219 Compliant MAP-T BR Tester written in C++ using DPDK " 2024; https://github.com/alhamadani-ahmed/Maptperf_v1.3.

[32] DPDK, "Programmer's guide-environment abstraction layer" 2024; https://doc.dpdk.org/guides/prog_guide/env_abstraction_layer.html#environment-abstraction-layer.

Ahmed Al-hamadani graduated from the Florida Institute of Technology (FIT), the USA in 2013 with an MSc in Computer Science. Since then, he has worked as a lecturer at the Department of Computer Engineering, University of Mosul, Iraq. Ahmed has been awarded the Stipendium Hungaricum scholarship to do his Ph.D. in informatics at the Budapest University of Technology and Economics (BME), Hungary. He started his study in the Department of Networked Systems and Services in September 2020. His research field is the benchmarking and performance analysis of IPv6 Transition Technologies.

Gábor Lencse received his M.Sc. and Ph.D. degrees in computer science from the Budapest University of Technology and Economics, Budapest, Hungary in 1994 and 2001, respectively. He has been working for the Department of Telecommunications, Széchenyi István University, Győr, Hungary since 1997. Now, he is a Professor. He has also been a part-time Senior Research Fellow at the Department of Networked Systems and Services, Budapest University of Technology and Economics since 2005. His research interests include the performance and security analysis of IPv6 transition technologies. He is a co-author of RFC 8219 and RFC 9313.