# Checking the Accuracy of Siitperf

Gábor Lencse

*Abstract*— **Siitperf is the world's first free software RFC 8219 compliant SIIT (Stateless IP/ICMP Translation, also called as Stateless NAT64) tester, which implements throughput, frame loss rate, latency and packet delay variation tests. In this paper, we show that the reliability of its results mainly depends on the accuracy of the timing of its frame sender algorithm. We also investigate the effect of Ethernet flow control on the measurement results. Siitperf is calibrated by the comparison of its results with that of a commercial network performance tester, when both of them are used for determining the throughput of the IPv4 routing of the Linux kernel.**

*Index Terms*—**accuracy, network benchmarking tools, calibration, frame loss rate, latency, network performance measurement, siitperf, throughput.**

## I. INTRODUCTION

RFC 8219 [1] has defined a benchmarking methodology for the high number of *IPv6 transition technologies* [2] by classifying them into a small number of categories and defining benchmarking procedures for each category. As far as we know, our **`siitperf`** [3] is the world's first free software RFC 8219 compliant SIIT (Stateless IP/ICMP Translation) [4] (also called stateless NAT64) tester, written in C++ using DPDK (Data Plane Development Kit) [5] available from GitHub [6]. Being a measurement tool, the accuracy of **`siitperf`** is a key issue, which we examine in this paper. To that end, first, we give a short introduction to RFC 8219 and **`siitperf`** only up to the measure necessary to understand the rest of this paper. Then, we define our error model by overviewing the most important factors that could cause unreliable measurement results. Next, we examine the effect of Ethernet flow control on the measurement results. After that, we measure the throughput of the same DUT (Device Under Test) using a commercial network performance tester and **`siitperf`** and compare their results. Finally, we discuss our results and disclose our plans for further research.

## II. A SHORT INTRODUCTION TO RFC 8219 AND SIITPERF

In order to provide the reader with the necessary background information for the understanding of the rest of this paper, we give a short overview of RFC 8219 and **`siitperf`.**

Submitted: March 21, 2021, revised April 15.
G. Lencse is with the Department of Telecommunications, Széchenyi István University, Győr, Hungary.
(e-mail: lencse@sze.hu)

### A. Summary of RFC 8219 in a Nutshell

RFC 8219 has defined a benchmarking methodology for IPv6 transition technologies aiming to facilitate their performance measurement in an objective way producing reasonable and comparable results. To that end, it has defined *measurement setups*, *measurement procedures*, and several parameters such as standard frame sizes, duration of the tests, etc. To be able to deal with the high number of different IPv6 transition technologies, they were classified into the following categories: *dual stack*, *single translation*, *double translation* and *encapsulation* technologies, and the members of each category may be handled together.

RFC 8219 recommends the *Single DUT test setup* shown in Fig. 1 for the performance evaluation of the single translation technologies, where SIIT belongs to. Here, the *Tester* device benchmarks the *DUT* (Device Under Test). Although the arrows would imply unidirectional traffic, testing with bidirectional traffic is required by RFC 8219 and testing with unidirectional traffic is optional. Of course, both X and Y in IPvX and IPvY are from the set of {4, 6}. Naturally, if we are talking about SIIT, then it implies that X≠Y.

From among the measurement procedures, we summarize only those that are implemented by **`siitperf`**.

*Throughput* is defined as the highest (constant) frame rate at which the DUT can forward all frames without frame loss. Although its measurement procedure has special wording, in practice, the throughput is determined by a binary search. There are further conditions, e.g. core measurements of the binary search should last at least for 60 seconds and the tester should continue on receiving for 2 more seconds after finishing frame sending so that all residual (buffered) frames may arrive safely.

The *frame loss rate* measurement procedure measures the frame loss rate at some specific frame rates starting from the maximum frame rate for the media and decreasing the frame rate in steps not higher than the 10% of the maximum frame rate. Measurements may be finished after two consecutive 0% frame loss results.

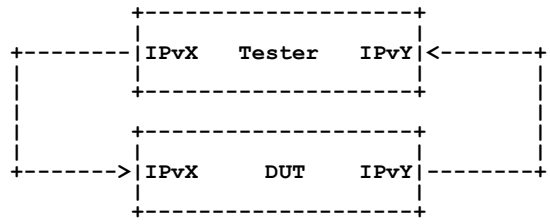


Fig. 1. Single DUT test setup [1].

DOI: 10.36244/ICJ.2021.2.1

Both the *latency* and the *packet delay variation* measurements are to be performed at the frame rate determined by the throughput test. The latency measurement has to tag at least 500 frames, the latencies of which are measured using sending and receiving timestamps, and the final results are the typical latency (the median of the latency values) and the worst case latency (the 99.9th percentile of the latency values). Packet delay variation measurement first determines the one way delay of every single frame, then it calculates the 99.9th percentile and the minimum of the one way delay values, and finally, their difference is the packet delay variation.

For an easy to follow introduction to RFC 8219, please refer to the slides of our IIJ Lab seminar presentation in Tokyo in 2017 [7].

On the one hand, we are not aware of any other RFC 8219 compliant benchmarking tools for network interconnect devices than our **`siitperf`**. On the other hand, RFC 8219 has taken several benchmarking procedures from the more than 20 years old RFC 2544 [8]. Several RFC 2544 compliant hardware and software Testers are listed in [9]. Further network benchmarking tools are collected and compared in [10].

### *B. Summary of siitperf in a Nutshell*

We give a short overview of **`siitperf`** on the basis of our open access paper [3], in which all the details can be found. Our aim was to design and implement a high performance and also flexible research tool. To that end, **`siitperf`** is a collection of binaries and shell scripts. The core measurements are performed by one of three binaries, which are executed multiple times by one of four shell scripts. The binaries perform the sending and receiving of certain IPv4 or IPv6 frames[1] at a pre-defined constant frame rate according to the test setup shown in Fig. 1. We note that **`siitperf`** allows X=Y, that is, it can also be used for benchmarking an IPv4 or IPv6 router. The shell scripts call the binaries supplying them with the proper command line parameters for the given core measurement.

The first two of the supported benchmarking procedures (*throughput* and *frame loss rate*) require only the above mentioned sending of test frames at a constant rate and counting of the received test frames, thus the core measurement of both procedures is the same. The difference is that throughput measurement requires to find the highest rate at which the DUT can forward all the frames without loss, whereas the frame loss rate measurement requires to perform the core measurement at various frame rates to determine the frame loss rate at those specific frame rates. The core measurement of both tests is implemented in the **`siitperf-tp`** binary and the two different benchmarking procedures are performed by two different shell scripts.

The *latency* benchmarking procedure requires that timestamps are stored immediately *after* sending and receiving of tagged frames. The latency for each tagged frame is calculated as the difference of the receiving and sending timestamps of the given frame. The latency benchmarking procedure is implemented by **`siitperf-lat`**, which is an extension of **`siitperf-tp`**.

From our point of view, the *packet delay variation* benchmarking procedure is similar to the latency benchmarking procedure, but it requires timestamping of every single frame. The packet delay variation benchmarking procedure is implemented by **`siitperf-pdv`**, which is also an extension of **`siitperf-tp`**.

The binaries are implemented in C++ using DPDK to achieve high enough performance. We used an object oriented design: the **`Throughput`** class served as a base class for the **`Latency`** and **`Pdv`** classes.

Internally, **`siitperf`** uses *TSC* (Time Stamp Counter) for time measurements, which is a very accurate and computationally inexpensive solution (it is a CPU register, which can be read by a single CPU instruction: **`RDTSC`** [11]).

To achieve as high performance as possible, all test frames used by **`siitperf-tp`** and **`siitperf-lat`** are pre-generated (including the tagged frames). The test frames of **`siitperf-pdv`** are prepared right before sending by modifying a set of pre-generated frames: their individual identifiers and checksums are rewritten.

Regarding our error model, it is important that the sending and receiving of the frames are implemented by sender and receiver functions, which are executed as threads by the CPU cores specified by the user in the configuration file.

## III. OUR ERROR MODEL

### *A. Accuracy of the Timing of Frame Sending*

There is an excellent paper that examines the accuracy of the timing of different software packet generators [12]. It points out that the *inter-sending time of the packets is rather imprecise* at demanding frame rates, if pure software methods are used. It also mentions the *buffering of the frames by the NIC* (Network Interface Card) among the root causes of this phenomenon, what we have also experienced and reported: our experience was that when a packet was reported by the DPDK function as "sent", it was still in the buffer of the NIC [3]. Unfortunately, this buffering completely discredits any investigation based on using timestamps stored at the sending of the frames: even if we store timestamps both before and after the sending of a frame, we may not be sure, when the frame was actually sent.

Imprecise timing may come from various root causes. At demanding frame rates, one of them is that our contemporary CPUs use several solutions to increase their performance including caching, branch prediction, etc. and they usually provide their optimum performance only after the first execution (or after the first few executions) of the core of the packet sending cycle, thus the first (few) longer than required inter-sending time(s) is/are followed by shorter ones to compensate the latency. This compensation depends on the

[1] more precisely: Ethernet frames containing IPv4 or IPv6 packets

timing algorithm of the sender function. For example, the original implementation of **dns64perf++** used a sophisticated algorithm that intended to distribute the compensation of such initial latency for the rest of the measurement time [13]. Unfortunately, the compensation algorithm did not work well and thus the sending rate was somewhat lower than required from the beginning of the measurement for a long time, and it was significantly higher than required at the end [14]. Therefore, we have replaced the timing algorithm with a simpler one that promptly compensates the latency [14]. We followed the same approach in **siitperf**, thus it sends the test frame (if it can), when its time has arrived, with no respect to what has happened before [3]. Therefore, **siitperf** very likely produces micro burst(s) at rates close to the upper limit of its sending performance.

Unfortunately, we did not have a NetFPGA device used by the authors of [12], therefore, we decided to check, how the imprecise timing of **siitperf** influences its measurement results. Our error model is that traffic with not exact inter-arrival time may have the following influence on the throughput test results:

1. The median decreases, because the imprecise timing causes sometimes overload and thus frame loss at lower rates than the throughput rate with precise timing.
2. The dispersion of the results increases, because some random events (like interrupts) influence each execution of the test differently.

The actual frame loss caused by the imprecise timing may also depend on a further parameter, namely, if Ethernet flow control (IEEE 802.3x) is used or not, because flow control may "iron out" the random peaks of the frame sending rate caused by imprecise timing.

Therefore, first, we test how the presence or absence of flow control influences the results. This phenomenon is interesting by itself, and the results of this comparison proved to be very important due to the limitations of our next examination.

Then, we benchmark the same DUT with both a calibrated tester and **siitperf** so that we can see the difference. The fact that **siitperf** is able to perform pure IPv4 or IPv6 benchmarking tests, allowed us to use an RFC 2544 [8] compliant legacy tester. This solution has also its limitations: although RFC 8219 has taken the throughput and frame loss rate tests verbatim from RFC 2544, the latency test has been redefined (it requires at least 500 tagged frames instead of a single one) and packet delay variation measurement is a completely new one. Thus they cannot be validated by an RFC 2544 tester.

We note that even if we can directly check the accuracy of frame sending of **siitperf-tp** only, we expect that the accuracy of frame sending of the other two programs is not worse, either. As for **siitperf-lat**, the relatively low number of tagged frames, which are distributed evenly, cannot make any significant effect. As for **siitperf-pdv**, the setting of their individual identifier and checksum requires some time, and thus there is non-zero lower bound for their inter-frame time, at least in theory. We note that it guarantees nothing in practice due to the fact of NIC buffering: back-to-back frames (that is frames with minimum inter-frame gap) may still occur.

### B. Consideration of Other Errors

Unlike the sender function that sends frames individually, the receiver function may receive multiple frames together to ensure high performance. This can surely not cause any problem with the throughput and frame loss rate measurements, because the frames are only counted. The receiving timestamps of latency and packed delay variation tests may be influenced, but they are also influenced by buffering even if they are taken out from the receive buffer individually.

The sending and receiving timestamps are subject to further errors due to the fact that interrupts may occur between the sending/receiving of the frames and taking the timestamp by the execution of the RDTSC machine code instruction. This is a kind of error we cannot measure. As for latency measurements, one possible mitigation can be, if the user sets the number of time stamps to be used to a significantly higher value than the required minimum 500 (**siitperf** supports up to 50,000) and thus the calculation of the 99.9th percentile removes the errors, if they are rare enough. This mitigation automatically applies for packet delay variation tests, as all frames are time stamped.

Although it is the responsibility of the user to specify the four cores that execute the sending and receiving threads so that they belong to the same physical CPU as the main core (used for starting the program), **siitperf** does some sanity checks if the TSC-s of the four CPU cores are synchronized with that of the main core. Otherwise the TSC values specified for starting and stopping the experiment as well as the differences of the timestamps of the corresponding senders and receivers would be invalid.

We believe that all other errors including the conversions between (milli)seconds and TSC, the counting of the sent and received frames, the calculations with the timestamps, etc. are subject to general software testing and verification procedures.

## IV. Investigation of the Effect of Ethernet Flow Control

To be able to investigate, how the presence or the absence of the Ethernet flow control influences the results, we needed a test system that is free from any other effects that may make our results noisy. Based on our SIIT benchmarking experience [15], we have chosen to reuse a previously built tests system, which was build up by two identical Dell PowerEdge C6220 servers in the NICT StarBED, Japan. The very same system was also used for benchmarking the extension of **siitperf** with the ability of using random source and destination port numbers [16] as required by RFC 4814 [17].

We have taken the following description of the test system from that paper [16].

"The servers were equipped with two 2GHz Intel Xeon E5-2650 CPUs having 8 cores each, 128GB 1333MHz DDR3 RAM and Intel 10G dual port X520 Ethernet network adapters.

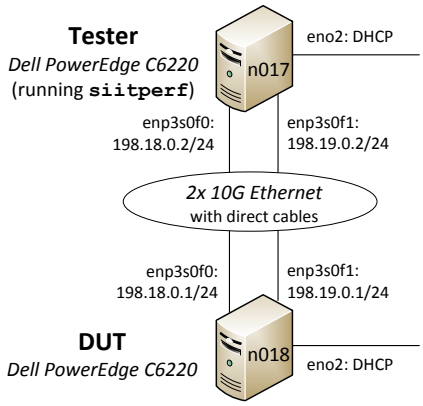


Fig. 2. Measurement setup for IPv4 Linux kernel routing: throughput tests with and without Ethernet flow control.

The Debian Linux operating system was updated to version 9.13 on all computers. The Linux kernel version was: 4.9.0-4-amd64. The DPDK version was 16.11.11-1+deb9u2." [16]

The "varport" branch of **siitperf** was used, its latest commit was bfddb5f on Aug 23, 2020. (Since then, the varport branch was merged into the master branch.)

We expected that the difference between the results with and without Ethernet flow control depends on the frame rate and we also wanted to test this hypothesis.

To achieve high enough frame rates, first, we benchmarked IPv4 kernel routing using random source and destination port numbers as we did in [16]. The topology of the test system is shown in Fig. 2. The CPU clock rate was set to fixed 2GHz on both computers and hyperthreading was switched off (using the same BIOS settings as specified in the appendix of [18]). All cores of the second CPU of the DUT were switched off using the **maxcpus=8** kernel parameter to avoid NUMA issues (please refer to [15] for a detailed explanation).

As for frames sizes, we used 64, 128 and 256 bytes from among the standard frame sizes recommended by RFC 8219, because the throughput of the DUT was limited by the performance of the DUT with these frame sizes, whereas throughput was limited by the maximum frame rate of the 10G Ethernet for all higher standard frame sizes.

Technical note: **siitperf** interprets the specified frame size values for IPv6 frames and uses 20 bytes less for IPv4 frames, therefore we always set 20 bytes higher values. This is important, if someone would like to repeat our experiments.

As required by RFC 8219, we used bidirectional traffic and 60s long tests at each step of the binary search. The *Error*[2] parameter of the binary search was set to 1. The throughput tests were performed 20 times and the median, minimum and maximum values of the 20 results were calculated. In addition to that, we have also calculated another value to express the consistent or scattered nature of the results, which we named *dispersion* and defined as follows:

$$Dispersion = \frac{max - min}{median} \cdot 100\% \quad (1)$$

The results are shown it Table I. We note that commercial Testers like the one we used in the next section, usually report the number of *all* frames per second (including frames in both directions), but **siitperf** reports the number of frames per second *per direction*. Thus the number of *all* frames per second forwarded by the Linux kernel was the double of the numbers shown in Table I.

Let us compare the results with and without flow control for each frame size individually. As for 64-byte frames, the median throughput with flow control (3,432,658fps) is about 0.6% higher than the median throughput without flow control (3,411,322fps). The lack of flow control has also significantly increased the dispersion of the results (from 0.6% to 1.28%). As for 128-byte frames, the median throughput with flow control (3,352,378fps) is only about 0.2% higher than the median throughput without flow control (3,444,630fps). Finally, with 256-byte frames, the median throughput with flow control (3,153,894fps) is only about 0.03% higher than the median throughput without flow control (3,152,872fps). We can observe that the difference between the median of the results with and without flow control definitely decreases with the increase of the frame size.

We are satisfied with the results in the sense that the 3.4Mfps is more than the half of the 6.3Mfps maximum frame rate **siitperf** can achieve on the given hardware [16] and our results with and without Ethernet flow control are quite close to each other (even the largest difference is below 1%).

To be able to test the effect of the Ethernet flow control at a significantly lower frame rate using the very same test system, we used fixed port numbers. In this case, the packet processing at the DUT was not hashed to all 8 active CPU cores of the DUT, but only two CPU cores were used[3] (one core per direction). This time we performed the throughput measurements using all standard frame sizes, as throughput was always lower than the theoretical maximum value for the media at the given frame size. Our results with flow control are shown in Table II. As we expected, the dispersion of the results is lower than 1% for all frame rates. The median throughput slightly decreases, when we increase the frame size from 64 bytes (885,643fps) through 128 bytes (878,256fps) to 256 bytes (857,575fps). There is significant decrease at 512 bytes (779,410fps) and median throughput remains constant (within measurement error) at all higher frame sizes. The investigation of the decrease of the median at 512 bytes is beyond the scope of our current paper, we just mention that it can also be observed in Fig. 3 of [15].

[2] Error means that the binary search may stop, when: *upper_limit – lower_limit* ≤ *Error*.

[3] We could observe only the load caused by software interrupts.

TABLE I.
IPV4 LINUX KERNEL ROUTING PERFORMANCE **WITH AND WITHOUT FLOW CONTROL**, DELL POWEREDGE C6220 SERVERS, FIXED 2GHZ CPU CLOCK RATE, 8 ACTIVE CPU CORES, RFC 4814 **RANDOM PORT NUMBERS**

| ***mode*** | ***with flow control*** | | | ***without flow control*** | | |
|---|---|---|---|---|---|---|
| ***frame size*** | ***64 bytes*** | ***128 bytes*** | ***256 bytes*** | ***64 bytes*** | ***128 bytes*** | ***256 bytes*** |
| median (fps) | 3,432,658 | 3,352,378 | 3,153,894 | 3,411,322 | 3,344,630 | 3,152,872 |
| min (fps) | 3,420,774 | 3,347,624 | 3,145,506 | 3,374,999 | 3,312,499 | 3,140,624 |
| max (fps) | 3,441,407 | 3,359,921 | 3,158,448 | 3,418,731 | 3,351,578 | 3,164,064 |
| disp. (%) | 0.60 | 0.37 | 0.41 | 1.28 | 1.17 | 0.74 |

TABLE II
IPV4 LINUX KERNEL ROUTING PERFORMANCE **WITH FLOW CONTROL**, DELL POWEREDGE C6220 SERVERS, FIXED 2GHZ CPU CLOCK RATE, 8 ACTIVE CPU CORES, BUT ONLY TWO OF THEM ARE USED DUE TO **FIXED PORT NUMBERS**

| ***frame size*** | ***64 B*** | ***128 B*** | ***256 B*** | ***512 B*** | ***768 B*** | ***1024 B*** | ***1280 B*** | ***1518 B*** |
|---|---|---|---|---|---|---|---|---|
| med (fps) | 885,643 | 878,256 | 857,575 | 779,410 | 779,503 | 779,982 | 779,194 | 779,035 |
| min (fps) | 882,811 | 874,006 | 855,467 | 775,389 | 777,326 | 777,342 | 777,828 | 777,342 |
| max (fps) | 887,696 | 880,860 | 859,631 | 781,746 | 781,861 | 781,251 | 780,274 | 779,663 |
| disp. (%) | 0.55 | 0.78 | 0.49 | 0.82 | 0.58 | 0.50 | 0.31 | 0.30 |

TABLE III
IPV4 LINUX KERNEL ROUTING PERFORMANCE **WITHOUT FLOW CONTROL**, DELL POWEREDGE C6220 SERVERS, FIXED 2GHZ CPU CLOCK RATE, 8 ACTIVE CPU CORES, BUT ONLY TWO OF THEM ARE USED DUE TO **FIXED PORT NUMBERS**

| ***frame size*** | ***64 B*** | ***128 B*** | ***256 B*** | ***512 B*** | ***768 B*** | ***1024 B*** | ***1280 B*** | ***1518 B*** |
|---|---|---|---|---|---|---|---|---|
| med (fps) | 880,381 | 875,126 | 850,740 | 778,295 | 778,861 | 778,535 | 779,078 | 779,069 |
| min (fps) | 826,610 | 742,186 | 749,999 | 757,807 | 765,624 | 734,374 | 749,693 | 749,968 |
| max (fps) | 883,850 | 876,617 | 853,763 | 780,274 | 780,274 | 779,790 | 780,518 | 779,420 |
| disp. (%) | 6.50 | 15.36 | 12.20 | 2.89 | 1.88 | 5.83 | 3.96 | 3.78 |

Our results without flow control are shown in Table III. The difference of the median throughput between the results with flow control (885,643fps) and without flow control (880,381fps) is about 0.6% at 64 bytes frame size. Although this difference decreases to 0.36% at 128 bytes, but it is about 0.8% at 256 bytes frame size. Thus the increase of the frames size was not enough to make the difference diminish. For the following three standard frame sizes, this difference is about: 0.14%, 0.08%, 0.2%, and for the last two frame sizes, the difference is deliberately less than measurement error. Unfortunately, the dispersion of the results of the measurements without flow control is rather high: it exceeds 15% at 128 bytes frame size. At this point, we cannot tell whether this high dispersion is caused by the improper timing of **`siitperf`** or by the nature of the DUT.

## V. CALIBRATION WITH A STANDARD TESTER

We have built two test systems to determine the IPv4 routing performance of the same DUT, which was a Sun Fire X4150 server with two Quad Core 2.83GHz Intel Xeon E5440 CPUs, four 2GB 667MHz DDR2 SDRAM modules and four Gigabit Ethernet ports. Debian 9.11 GNU/Linux operating system with 4.9.0-5-amd64 kernel was installed on it. The clock frequency of all 8 CPU cores was set to fixed 2.833GHz using the **`cpufreq-set`** command of the **`cpufrequtils`** package.

### *A. Reference Measurement*

To provide reference, the throughput of IPv4 Linux kernel routing was measured using a commercial Anritsu MP1590B Network Performance Tester. It had a four port Anritsu MU210212A 10/100/1000M Ethernet Module, and we used Port1 and Port2 of the module. The measurement setup is shown in Fig. 3.

As RFC 8219 has somewhat extended the standard frame sizes to be used for benchmarking originally defined in RFC 2544, we have chosen custom frame sizes and defined the following frame sizes: 64, 128, 256, 512, 768,1024, 1280, 1518.

As required by RFC 8219, bidirectional traffic was used and full 60s length trials were executed and the "Loss Tolerance" parameter was set to 0%.

The Anritsu tester has a parameter called "Resolution", which can be specified as the percentage of maximum frame rate of the media. Its smallest possible value is 0.01. As the theoretical maximum frame rate for Gigabit Ethernet with 64 byte frame size is 1,488,095, this setting means that the

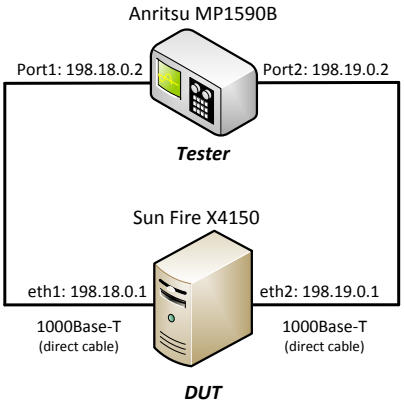


Fig. 3. Measurement setup for IPv4 Linux kernel routing: reference throughput test with a commercial Tester.

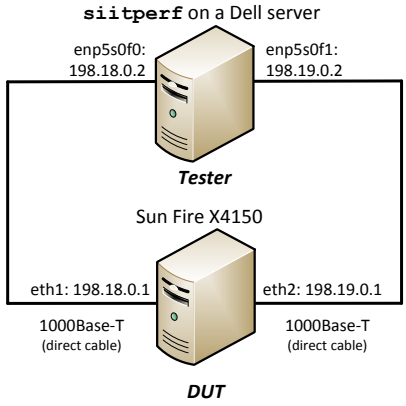


Fig. 4. Measurement setup for IPv4 Linux kernel routing: throughput test with **siitperf**.

resolution (or with other words *error*) of the binary search was about 149fps (calculated as: 1,488,095/10,000).

We note that the Anritsu tester does not perform a binary search, if the test at the "Maximum Frame Rate" is successful. We set the value of this parameter to 100%.

By default, flow control was not enabled on the Anritsu tester. When we enabled flow control, the Anritsu tester became practically unusable for throughput measurements, because it qualified all tests as successful, even if they lasted much longer than 60s. Thus, we could use this tester for meaningful measurements only, when flow control was disabled.

### B. Measurements with Siitperf

The parameters of the DUT were the same as in the previous case, but this time the Tester was a Dell PowerEdge R620 server with two six core 2GHz Intel Xeon E5-2620 CPUs, two 16GB 1600MHz DDR3 SDRAM modules and with an additional Intel I350-T4 Ethernet Server Adapter (needed for DPDK). Debian 9.11 GNU/Linux operating system with 4.9.0-11-amd64 kernel was installed on it. The version of DPDK was 16.11.9-1+deb9u2. As **siitperf** does not have version numbers yet, we can identify its version with its latest commit number 05247a1 on Jul 1, 2020. This time, we used the master branch.

The measurement setup is shown in Fig. 4. We used the same standard frame sizes mentioned before.

As required by RFC 8219, bidirectional traffic was used and full 60s length trials were executed, and the "Error" of the binary search was set to 1.

We note that as the **binary-rate-alg.sh** script distributed with **siitperf** supports only tests for a single pre-set frame size, with a single pre-set upper bound, we have added a **for** cycle to the script with the appropriate frame sizes and the following upper bounds for the consecutive standard frame sizes: 1,500,000 850,000 460,000 240,000 160,000 120,000 100,000 82,000. They are wilfully somewhat higher than the theoretical maximum frame rates for the media with the given frame size, because we wanted to test and demonstrate how **siitperf** behaves, when the maximum frame rate for the media is achieved by the DUT. Our script performed the binary search for all standard fame sizes starting in the interval of 0 (as lower bound) and the above mentioned upper bound values.

Unlike with the Antritsu Tester, the log file of the DUT showed that flow control was enabled on the interfaces used for testing (Flow Control: Rx/Tx). We tried to switch off flow control using the same command as with the 10G Ethernet interfaces (**ethtool -A *interface* rx off tx off**), which has been executed without any error message, however flow control remained enabled.

### C. Results

The Anritsu Tester reported the results in a form of a graph, which we include in Fig. 5 to facilitate an easy overview of the results. Except for the first two frame sizes, the throughput achieved its theoretical maximum value. However, this reporting format covers some very important details by displaying only the average value of the measurement results. Therefore, we processed the detailed result file and calculated the median, minimum, and maximum of the 20 throughput results for each frame size. Please see our results in Table IV. We note that the Anritsu Tester reported the number of *all*

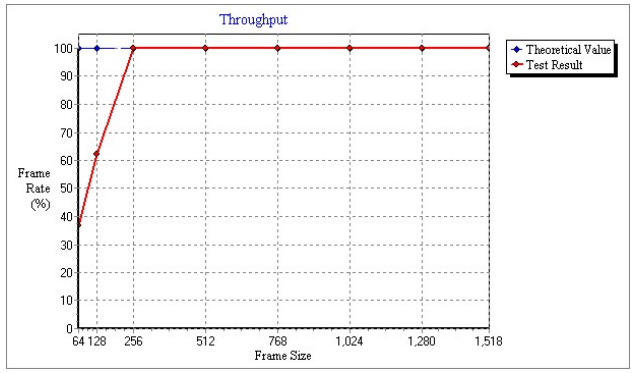


Fig. 5. IPv4 Linux kernel routing performance of the Sun server: reported by the Anritsu Tester with no flow control

TABLE IV
IPV4 LINUX KERNEL ROUTING PERFORMANCE OF THE SUN SERVER: MEASURED BY THE ANRITSU TESTER WITHOUT FLOW CONTROL

| *frame size* | *64 B* | *128 B* | *256 B* | *512 B* | *768 B* | *1024 B* | *1280 B* | *1518 B* |
|---|---|---|---|---|---|---|---|---|
| med (fps) | 548,958 | 526,351 | 452,899 | 234,962 | 158,629 | 119,732 | 96,154 | 81,274 |
| min (fps) | 511,310 | 522,720 | 452,853 | 234,962 | 158,629 | 119,732 | 96,154 | 81,274 |
| max (fps) | 553,720 | 529,561 | 452,899 | 234,962 | 158,629 | 119,732 | 96,154 | 81,274 |
| disp. (%) | 7.73 | 1.30 | 0.01 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

TABLE V
IPV4 LINUX KERNEL ROUTING PERFORMANCE OF THE SUN SERVER: MEASURED BY **SIITPERF** WITH FLOW CONTROL

| *frame size* | *64 B* | *128 B* | *256 B* | *512 B* | *768 B* | *1024 B* | *1280 B* | *1518 B* |
|---|---|---|---|---|---|---|---|---|
| med (fps) | 549,297 | 522,413 | 452,930 | 234,986 | 158,652 | 119,752 | 96,173 | 81,294 |
| min (fps) | 547,850 | 521,285 | 452,926 | 234,986 | 158,650 | 119,752 | 96,173 | 81,294 |
| max (fps) | 550,782 | 524,610 | 452,960 | 234,990 | 158,667 | 119,767 | 96,178 | 81,301 |
| disp. (%) | 0.53 | 0.64 | 0.01 | 0.00 | 0.01 | 0.01 | 0.01 | 0.01 |

frames per second (including frames in both directions), but we divided the results by two to report the number of frames per second *per direction*. We did so to show values comparable with the theoretical maximum frame rates given in Appendix A.1 of RFC 5180 [19].

One of the most conspicuous things in the table is the high dispersion of the results at 64-byte frame size. It is caused by a single outlier. We have investigated the case in the measurement log file, and we found that 8 frames were missing during that step of the binary search when the target rate was 34.37%. Of course, it meant that the test failed. After that, all tests were successful and thus the final result was 34.36%. This single outlier does not influence the median, but it is reflected by the minimum and, therefore, in the dispersion, too.

As for the results at 128-byte frame size, the minimum and the maximum are nearly symmetrical around the median.

As for the results at 256-byte frame size, a single test failed at 100% due to the loss of a few frames, therefore, binary search was performed, which finished at 99.99%. All other tests passed at 100% and thus no binary search was performed.

No binary search was performed at any higher frame sizes, this is why their minimum and maximum values are equal with their medians.

The results of the throughput measurements with **siitperf** are shown in Table V. The dispersion of the results is always below 1%, and it is practically 0 upwards from 256 bytes frame size, as the maximum frame rate for the media has limited the throughput. As the upper limit was set higher than the theoretical maximum frame rate for the media, **siitperf** executed binary search and it measured slightly higher values. It was possible for at least two reasons:

- As Appendix A.1 of RFC 5180 states: "Ethernet's maximum frame rates are subject to variances due to clock slop. The listed rates are theoretical maximums, and actual tests should account for a +/- 100 ppm tolerance."
- The "TOLERANCE" parameter of **siitperf** was set to 1.00001, which means that 0.001% more time is allowed for sending.

There are two throughput values that were limited by the CPU performance: throughput measured with 64 bytes and 128 bytes frame sizes. The differences of the results of the two test systems are 0.06% and 0.75%, which we consider good and acceptable, respectively.

## VI. DISCUSSION AND PLANS FOR FUTURE RESEARCH

Our conditions for calibrating **siitperf** with a standard tester were far from ideal. We cannot tell the maximum frame rate, at which the CPU of the Dell PowerEdge R620 server would be able to generate frames, but it is very likely several million frames per second, thus the measured throughput around 550,000 fps was not at all close to it. The technical issue that we could use the Anritsu tester only without flow control, whereas we could use **siitperf** only with flow control (in the Gigabit Ethernet environment) makes the comparison of their results more difficult.

We plan to purchase a NetFPGA device like the one used by the authors of [12] and examine the inter-frame time of the traffic generated by **siitperf**.

We also plan to test the accuracy of **siitperf** in a 10GBase-T environment with a Spirent SPT-N4U Tester used out of courtesy for the measurements of [20].

## VII. CONCLUSION

We have carefully examined, what kind of factors may distort the measurement results of **siitperf**, and we set up an error model.

We have compared the results of **siitperf** used with and without Ethernet flow control in a 10GBase-T environment, and we found that the deviation of the results was always below 1%.

We have calibrated **siitperf** with a commercial Tester in

a Gigabit Ethernet environment, and we found that the deviation of the results was below 1%.

We conclude that it is necessary to calibrate **`siitperf`** also in a 10GBase-T environment and we plan to do so.

## ACKNOWLEDGEMENTS

The measurements for the investigation of the effect of Ethernet flow control were carried out by remotely using the resources of NICT StarBED, 2-12 Asahidai, Nomi-City, Ishikawa 923-1211, Japan. The author would like to thank Shuuhei Takimoto for the possibility to use StarBED, as well as to Satoru Gonno and Miku Takuma for their help and advice in StarBED usage related issues.

The author thanks the National Media and Infocommunications Authority (NMHH) of Hungary for lending us the Anritsu MP1590B Network Performance Tester. The author thanks István Pilisi, NMHH, for his support how to use the device.

The author thanks István Pilisi and Alexandru Moise for reading and commenting the manuscript.

## REFERENCES

[1] M. Georgescu, L. Pislaru, G. Lencse, "Benchmarking methodology for IPv6 transition technologies", IETF RFC 8219, 2017. DOI: 10.17487/rfc8219

[2] G. Lencse and Y. Kadobayashi, "Comprehensive survey of IPv6 transition technologies: A subjective classification for security analysis", *IEICE Transactions on Communications*, vol. E102-B, no 10, pp. 2021–2035. DOI: 10.1587/transcom.2018ebr0002

[3] G. Lencse, "Design and implementation of a software tester for benchmarking stateless NAT64 gateways", *IEICE Transactions on Communications*, vol. E104-B, no. 2, pp. 128-140. DOI: 10.1587/transcom.2019ebn0010

[4] C. Bao, X. Li, et al., IP/ICMP translation algorithm, IETF RFC 7915, DOI: 10.17487/rfc7915

[5] D. Scholz, "A look at Intel's dataplane development kit", *Proc. Seminars Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM)*, Munich, 2014, pp. 115–122. DOI: 10.2313/NET-2014-08-1_15

[6] G. Lencse, Siitperf: An RFC 8219 compliant SIIT (stateless NAT64) tester, free software under GPLv3 license, [Online] Available: https://github.com/lencsegabor/siitperf

[7] G. Lencse, Benchmarking methodology for IPv6 transition technologies, IIJ Lab seminar, Tokyo, Oct. 10, 2017. [Online] Available: https://seminar-materials.iijlab.net/iijlab-seminar/iijlab-seminar-20171010.pdf

[8] S. Bradner, J. McQuaid, Benchmarking methodology for network interconnect devices, IETF RFC 2544, 1999. DOI: 10.17487/rfc2544

[9] D. Raumer, S. Gallenmüller, et al., "Revisiting Benchmarking Methodology for Interconnect Devices", *Proc. 2016 Applied Networking Research Workshop (ANRW'16)*, Berlin, 2016. DOI: 10.1145/2959424.2959430

[10] K. Velásquez and E. Gamess, "A survey of network benchmark tools", in: *Machine Learning and Systems Engineering*, Springer, Dordrecht, 2010, pp. 465–480. DOI: 10.1007/978-90-481-9419-3_36

[11] Intel, Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 2B: Instruction Set Reference, M-U, Order Number: 253667-060US, 2016. [Online] Available: https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-2b-manual.pdf

[12] P. Emmerich, S. Gallenmüller, et al., Mind the gap - A comparison of software packet generators, *Proc. 2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, Beijing, China, 2017. DOI: 10.1109/ancs.2017.32

[13] G. Lencse, D. Bakai, "Design and implementation of a test program for benchmarking DNS64 servers", *IEICE Transactions on Communications*, vol. E100-B, no. 6, pp. 948–954. DOI: 10.1587/transcom.2016EBN0007

[14] G. Lencse and A. Pivoda, "Checking and increasing the accuracy of the dns64perf++ measurement tool for benchmarking DNS64 servers", *International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems*, vol. 7, no 1, pp. 10–16. DOI: 10.11601/ijates.v7i1.255

[15] G. Lencse, K. Shima, "Performance Analysis of SIIT Implementations: Testing and Improving the Methodology", *Computer Communications*, vol. 156, no. 1, pp. 54–67. DOI: 10.1016/j.comcom.2020.03.034

[16] G. Lencse, "Adding RFC 4814 random port feature to siitperf: Design, implementation and performance estimation", *International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems*, vol. 9, no. 3, pp. 18–26. DOI: 10.11601/ijates.v9i3.291

[17] D. Newman, T. Player, "Hash and stuffing: Overlooked factors in network device benchmarking", IETF RFC 4814, 2008. DOI: 10.17487/RFC4814

[18] G. Lencse and Y. Kadobayashi, "Benchmarking DNS64 Implementations: Theory and Practice", *Computer Communications*, vol. 127, no. 1, pp. 61–74. DOI: 10.1016/j.comcom.2018.05.005

[19] C. Popoviciu, A. Hamza, et al., "IPv6 benchmarking methodology for network interconnect devices", IETF RFC 5180, 2008. DOI: 10.17487/rfc5180

[20] G. Lencse, "Benchmarking Stateless NAT64 Implementations with a Standard Tester", *Telecommunication Systems*, vol. 75, no. 3, pp. 245–257. DOI: 10.1007/s11235-020-00681-x

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

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

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