Understanding the Performance of Different Packet Reception and Timestamping Methods in Linux

Alexej Grigorjew[1], Lukas Kilian Schumann[1], Philip Diederich[2], Tobias Hoßfeld[1], Wolfgang Kellerer[2]
[1]University of Würzburg [2]Technical University of Munich
Contact: alexej.grigorjew@uni-wuerzburg.de

***Abstract*—This document briefly presents some renowned packet reception techniques for network packets in Linux systems. Further, it compares their performance when measuring packet timestamps with respect to throughput and accuracy. Both software and hardware timestamps are compared, and various parameters are examined, including frame size, link speed, network interface card, and CPU load. The results indicate that hardware timestamping offers significantly better accuracy with no downsides, and that packet reception techniques that avoid system calls offer superior measurement throughput.**

I. Introduction

Accurate measurements of packet reception times have always been an indispensable tool for the research and verification of networking equipment. In recent years, deterministic performance guarantees have become more important, and with that, the requirements for accurate timing measurements have also increased.

When attempting to verify deterministic upper bounds for delay or traffic volume, it is important the the measurement methodology does not introduce any extra uncertainty. This means the the measurement inaccuracy for timings should be orders of magnitude lower than the intended measured intervals. In addition, the performance of the measurement tool should be sufficient to rule out any packet loss that prevents accurate assessment of traffic volume. However, dedicated measurement equipment can quickly become very expensive compared to standard Linux software and off-the-shelf Network Interface Cards (NICs). Therefore, the accuracy of these affordable methods is investigated in this work.

In WueWoWas'22, a methodology for affordable hardware timestamping with commercially available components and sub-microsecond accuracy was presented [1]. In this work, the presented methodology is investigated with various packet reception techniques, including regular system calls[1], shared ring buffers[2], and relying on libpcap[3] for packet reception. Further, for software timestamps, XDP [2] is used to skip the majority of the Linux networking stack for increased performance. The accuracy and throughput of these techniques is compared, while also altering the frame size, link speed, NIC, and CPU load of the measurement device. While some aspects of the parameter study are still ongoing work, this paper already presents some interesting results.

[1]https://man7.org/linux/man-pages/man2/recvmmsg.2.html
[2]https://www.kernel.org/doc/Documentation/networking/packet_mmap.txt
[3]https://github.com/the-tcpdump-group/libpcap

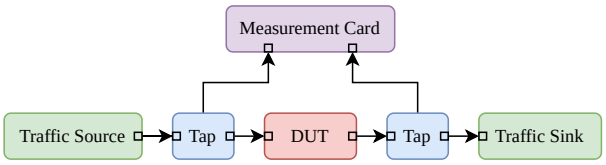

Figure 1. General test setup for the conducted parameter study. The Device Under Test (DUT) is represented by a copper cable for the parameter study, i.e., the delay between source and sink should be constant.

The remainder of the paper is structured as follows. Section II briefly covers related work in the areas of delay measurements and timestamping. In Section III, the measurement setup is presented and all parameters are explained. The preliminary results are explained in Section IV. Finally, Section V discusses our next steps and concludes the paper.

II. Related Work

For accurate measurements, it is important that the variability between timestamps is as small as possible. One factor that creates timestamp variability are the physical properties of the oscillator [3]–[5]. Small variability is also very important for time synchronization. Related Work [4], [6], [7] utilizes hardware timestamping for more precise time synchronization. Furthermore, different works [8], [9] show that it is possible to achieve similar synchronization levels with software timestamping mechanism. Similarly, [10] compares the accuracy of different timestamping methods in a non-synchronization context. The above methods have in common that they use Network Interface Cards (NICs). This is important for synchronization purposes. However, for measuring and timestamping network packets this is not strictly required since the cards must not necessarily interact with the network. For packet capture, special cards like Data Acquisition and Generation Cards (DAG) exist. DAG Cards offer high precision and accuracy since they are purpose-built for this [11], [12]. The drawback of DAG Cards is the high price and they are therefore left out.

III. Measurement Setup and Parameters

This section summarizes our measurement setup, including testbed topology, hardware choices, different reception methods, and the investigated parameters. The methodology for delay measurements and hardware timestamping has previously been presented in [1]. For further details on hardware

timestamping with commodity hardware, please refer to that publication.

Figure 1 illustrates the general testbed topology. A traffic source sends data towards a traffic sink, which traverses a Device Under Test (DUT) on its path. The delay that the DUT introduces is assessed by mirroring all packets before and after the DUT towards a separate measurement PC by means of transparent network taps. The delay is then calculated by measuring the difference between the arrival of the first and the second replication of the same packet at the measurement PC. In addition, the inter-arrival times of subsequent packets at any of both measurement interfaces can be used to assess traffic volume before and after the DUT. This can be useful, for example, to measure the effect of networking devices on the burstiness of a flow.

In the following parameter study, the traffic source is equipped with an Intel i7-2600 CPU, 16 GB of RAM and an Intel X550-T1 10G NIC. It is running Ubuntu 18.04 and using Moongen [13] for efficient traffic generation up to 10 Gbit/s. For consistent measurements, the DUT is represented by a simple copper cable that produces constant delays. For the network taps, Dualcomm ETAP-XG 10G are used. In our experiments, they provided extremely low (sub-microsecond) constant delays and full 10 Gbit/s throughput under any traffic mixture. Finally, the measurement PC is equipped with an Intel i7-4790 CPU, 16 Gbit/s of RAM, and different multi-port NICs that support hardware timestamping with different maximum link speeds: Intel I350-T4 (1G), Intel X550-T2 (10G), and Chelsio T520-BT (10G).

A. Packet Reception Methods

In this document, four types of packet reception methods are compared with respect to throughput and accuracy of packet timestamping. The simplest method is packet reception via system calls, namely `recvmsg` and `recvmmsg`[1] for a batch of packets. The disadvantage of system calls for this purpose is the extra work involving the communication with the operating system, most notably copying the packet data from kernel space into user space memory. Therefore, an alternative with a shared RX ring buffer is investigated. The shared memory block is created with a single `mmap`[2] system call. It can be written to by the kernel and can be directly accessed by the user space program without involving another copy operation or context switch. This method is referred to with the keyword *ring* or *rx_ring* in the evaluation. While `mmap` can be used directly, its documentation recommends to use the libpcap[3] library for simplified packet reception instead. Note that, due to inconsistent implementations in the cxgb4 driver[4], the Chelsio T520 NIC cannot receive packets via libpcap with hardware timestamping enabled. However, hardware timestamping does work when sending the flags manually with the `recvmmsg` and `mmap` methods. Finally, in an attempt to skip most of the Linux networking stack, XDP [2] can be used to receive and filter packets by a small program directly in the kernel space. At the time of writing this paper, this method can only produce software timestamps. However, passing meta information such as hardware timestamps is currently being developed. Note that, once again, the cxgb4 driver does not implement XDP at the current time. Therefore, this method is not available for the Chelsio NIC.

[4]https://github.com/Xilinx/linux-xlnx/blob/master/drivers/net/ethernet/chelsio/cxgb4/cxgb4_ethtool.c

Table I
PARAMETERS AND DEFAULT VALUES (UNDERLINED) FOR THE PARAMETER STUDY.

Technique	**Frame size**	**Link speed**	**NIC**	**CPU**
rx_ring, hw/sw	64 bytes	100M	Intel I350	idle
recvmmsg, hw/sw	256 bytes	1G	Intel X550	stress
libpcap, hw/sw	1518 bytes	10G	Chelsio T520	
xdp				

B. Experiment Design, Parameters, and Defaults

For the evaluation, a series of simple measurements has been conducted. The traffic source sends packets with unique identifiers towards the traffic sink at full line rate. Each of these packets is mirrored by both taps and each replication is sent towards the measurement PC. The latter attempts to receive all packets from both interfaces, extracts their unique identifiers, and compares them to an internal list of already seen packets. If a match has been found, the packet is recorded as *successfully measured* and the delay between both receptions is recorded.

First, two simple experiments have been conducted where the time series of recorded delays is displayed. In the first experiment, hardware and software timestamps are compared to each other. In the second experiment, the accuracy of hardware timestamping is further tested by changing the cable between both taps in the middle of the experiment. The cable length is increased in order to accurately control the latency between both taps – an increased cable length should be recognized as an increased latency with sufficient accuracy.

Finally, a parameter study has been conducted with three key performance indicators: the proportion *successfully measured* packets, the *mean delay* and the *standard deviation* of measured delays. In that study, five parameters have been varied. Their values are summarized in Table I. The *technique* parameter includes both the packet reception method and the timestamping method, i.e., hardware or software timestamps. The *frame size* parameter is indirectly responsible for the *packet rate* at full line rate transmissions. The *link speed* and *NIC* are varied and combined as applicable. Finally, with the *CPU* stress parameter, it is evaluated whether a high CPU utilization has any impact on measurement throughput and timestamp accuracy.

Both main effects and interactions have been investigated for the parameter study. The default values for the fixed parameters of every experiment are underlined in Table I. Every experiment is repeated five times, and mean values with 95% confidence intervals are reported for each applicable parameter combination.

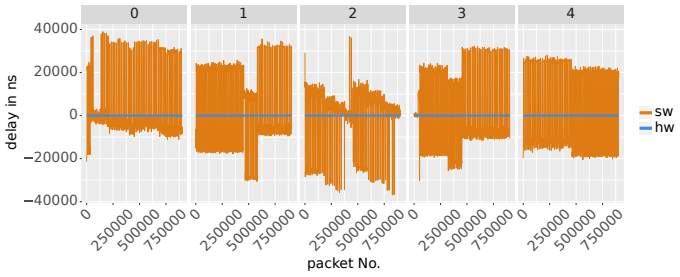

Figure 2. Five time series plots with hardware and software timestamps.

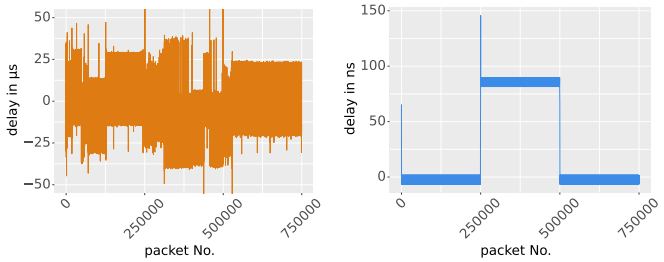

Figure 3. Time series plots with varying cable lengths, software (left) and hardware (right).

IV. Preliminary Results

This section covers the most important results of the parameter study so far. Note that this is still ongoing work, and that presenting all main effects and interactions is omitted due to space constraints.

Figure 2 presents the results of the first time series comparison. It shows five consecutive executions of a measurement with one million packets each. The frame size was set to 256 bytes, the Intel I350-T4 was used for measurements, a link speed of 1 Gbit/s was configured, and the CPU of the measurement PC was idle at the beginning of the test. Both software and hardware timestamps were acquired with the `mmap` (or `rx_ring`) methodology. The results indicate that there is a high fluctuation of ±40 µs in software timestamps, and that the interval, in which the values oscillate randomly, changes after arbitrary time durations. In contrast, the hardware values remain stable within ±16 ns centered around 0 (as calibrated earlier, cf. [1]). This measurement may indicate that hardware timestamps are several orders of magnitude more accurate than software timestamps, but in theory, these values could also be produced by environmental effects such as large scale wbatch processing or extremely low clock tick rates.

For this reason, an experiment with a stable, but controllable, source of delay is required. Figure 3 illustrates the time series results of the second experiment. The left figure (orange) shows an experiment with software timestamps, the right figure (blue) shows hardware timestamps. In both cases, 750 000 packets are transmitted between the two taps. After 250 000 packets, the 2 m cable between the taps is replaced by a longer 20 m cable for the duration of the next 250 000 packets, then it is replaced again by the original 2 m cable. With the anticipated accuracy, the additional propagation delay through 18 m of extra copper wire should become visible.

Software timestamps fluctuate in intervals of roughly 40 µs, as before. The additional propagation delay is orders of magnitude lower and masked within these fluctuations. However, with hardware timestamps, a very distinct increase of roughly 88 ns is measurable. With an extra 18 m of cable length, the propagation delay would be 4.9 ns/m, which corresponds to 0.68 times the speed of light. This is well within the expectations for copper wire. This proves that the hardware timestamps actually represent the behavior of the DUT with a measurement error of roughly ±16 ns, and that the low values are not merely a product of the measurement granularity.

Finally, the most important results of the parameter study are presented in the following. First, the higher accuracy of hardware timestamps was observable throughout all parameter combinations (Fig. 4). Operating the Intel NICs at a lower line rate caused lower accuracies, e.g., the I350 dropped from ±16 ns to ±80 ns at 100 Mbit/s, most likely due to reduced internal clock rates. Aside from the Chelsio NIC, hardware timestamp accuracy was not affected by CPU load, frame size, or packet reception technique (Fig. 5). The Chelsio NIC was slightly affected by frame sizes (still within submicrosecond accuracy), and it seems to fall back to software timestamps at high packet rates. Software timestamps could in general be affected by both CPU load and reception method, especially for the `mmap` and `recvmmsg` methods (Fig. 5). Surprisingly, with XDP packet reception, stressing the CPU did not affect the standard deviation of reported software timestamps significantly. In addition, for both Intel NICs, CPU load had little effect on the libpcap timestamps as well. It can be assumed that libpcap attempts to use XDP for software timestamping whenever applicable, especially since the Chelsio libpcap timestamps (whose driver does not support XDP) are greatly affected by CPU stress. As for successfully measured packets, the `mmap` technique tends to outperform the other techniques, however the difference towards libpcap and XDP is not statistically significant (Fig. 6). System calls with `recvmmsg` showed the worst performance throughout all tests. The timestamping method (`hw`/`sw`) did not have a measurable effect on the observed throughput (Fig. 6). 10 Gbit/s experiments have been conducted, but they remain largely inconclusive due to a reception limit of roughly 2.2 million packets per second, likely caused by the ixgbe driver.

V. Conclusion

In this work, different packet reception techniques and timestamping methods are compared with respect to timestamping accuracy and observable throughput. The results of a preliminary parameter study indicate that hardware timestamping can provide accurate results with ±16 ns measurement inaccuracy, with no measurable negative effect towards throughput. For software timestamps, XDP and libpcap provide a good compromise when applicable. The highest measured throughput was achieved by manually setting up a shared ring buffer via `mmap`. In future experiments, the 10 Gbit/s performance will be investigated in more detail, possibly accompanied by using the DPDK framework for packet reception.

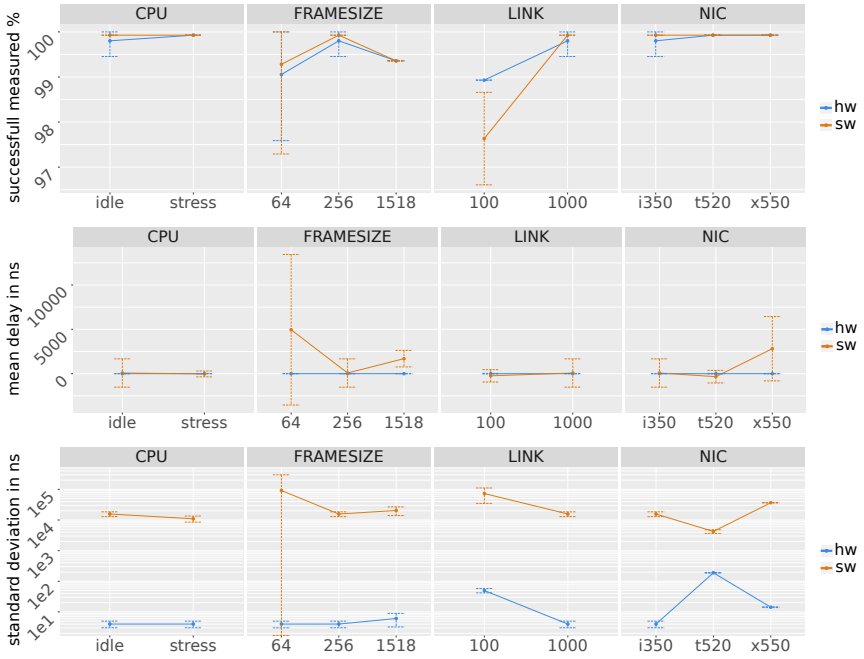

Figure 4. Main effect plots for the parameters in Table I with the `mmap` (or *rx_ring*) technique.

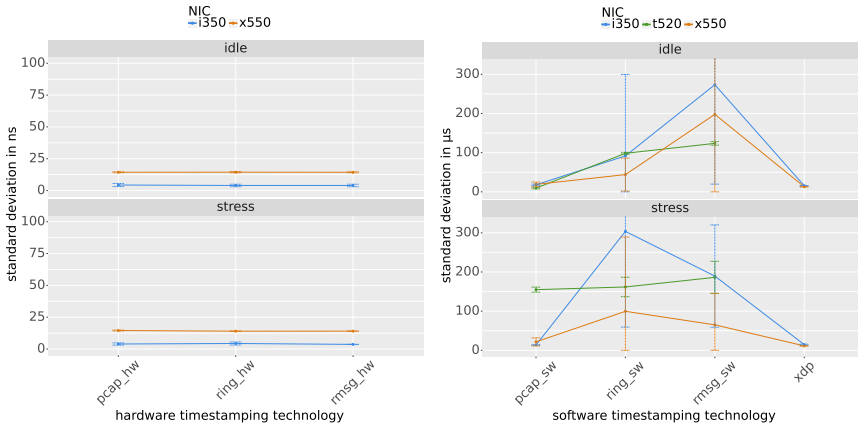

Figure 5. Standard deviation comparison for different timestamping methods, different NICs, 64 bytes frames, and 1 Gbit/s link speed. The Chelsio t520 was omitted in the hardware results due to large deviations skewing the y-axis.

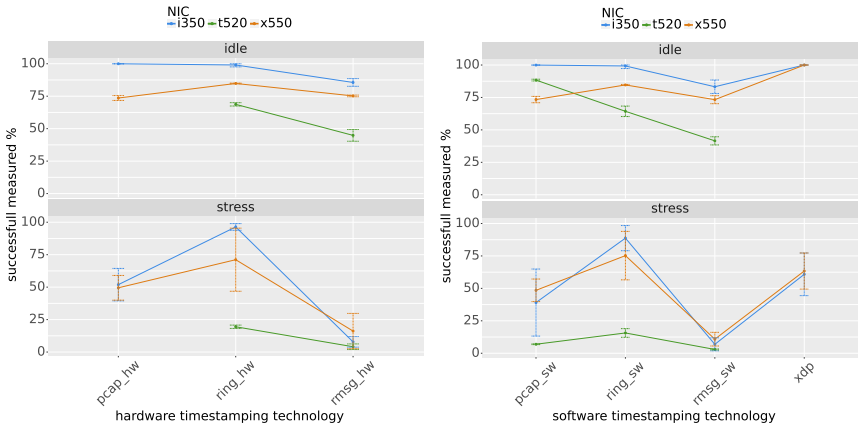

Figure 6. Throughput comparison for different timestamping methods, different NICs, 64 bytes frames, and 1 Gbit/s link speed.

References

[1] A. Grigorjew, P. Diederich, T. Hoßfeld, and W. Kellerer, "Affordable measurement setups for networking device latency with sub-microsecond accuracy," in *Würzburg Workshop on Next-Generation Communication Networks (WueWoWas'22)*, 2022, p. 5. DOI: 10.25972/OPUS-28075.

[2] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, *et al.*, "The express data path: Fast programmable packet processing in the operating system kernel," in *Proceedings of the 14th international conference on emerging networking experiments and technologies*, 2018, pp. 54–66.

[3] A. Najafi and M. Wei, "Graham: Synchronizing Clocks by Leveraging Local Clock Properties," en, Apr. 2022.

[4] S. Gallenmüller, F. Wiedner, J. Naab, and G. Carle, "How Low Can You Go? A Limbo Dance for Low-Latency Network Functions," en, *Journal of Network and Systems Management*, vol. 31, no. 1, p. 20, Dec. 2022. DOI: 10.1007/s10922-022-09710-3. [Online]. Available: https://doi.org/10.1007/s10922-022-09710-3 (visited on 05/12/2023).

[5] C. Andrich, M. Engelhardt, A. Ihlow, N. Beuster, and G. del Galdo, "Measurement of Drift and Jitter of Network Synchronized Distributed Clocks," in *2020 Joint Conference of the IEEE International Frequency Control Symposium and International Symposium on Applications of Ferroelectrics (IFCS-ISAF)*, ISSN: 2375-0448, Jul. 2020, pp. 1–8. DOI: 10.1109/IFCS-ISAF41089.2020.9234926.

[6] M. Ulbricht, J. Acevedo, S. Krdoyan, and F. H. Fitzek, "Precise fruits: Hardware supported time-synchronisation on the RaspberryPI," in *2021 International Conference on Smart Applications, Communications and Networking (SmartNets)*, Sep. 2021, pp. 1–6. DOI: 10.1109/SmartNets50376.2021.9555415.

[7] Y. Li, B. Noseworthy, J. Laird, T. Winters, and T. Carlin, "A study of precision of hardware time stamping packet traces," in *2014 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, ISSN: 1949-0313, Sep. 2014, pp. 102–107. DOI: 10.1109/ISPCS.2014.6948700.

[8] A. Mahmood and T. Sauter, "Improving the accuracy of software-based clock synchronization and encountering interrupt coalescence," in *2015 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, ISSN: 1949-0313, Oct. 2015, pp. 82–87. DOI: 10.1109/ISPCS.2015.7324688.

[9] V. Moreno, P. M. S. del Rio, J. Ramos, J. J. Garnica, and J. L. Garcia-Dorado, "Batch to the Future: Analyzing Timestamp Accuracy of High-Performance Packet I/O Engines," *IEEE Communications Letters*, vol. 16, no. 11, pp. 1888–1891, Nov. 2012, Conference Name: IEEE Communications Letters. DOI: 10.1109/LCOMM.2012.092812.121433.

[10] P. Emmerich, S. Gallenmüller, G. Antichi, A. W. Moore, and G. Carle, "Mind the Gap - A Comparison of Software Packet Generators," in *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, May 2017, pp. 191–203. DOI: 10.1109/ANCS.2017.32.

[11] G. Antichi, M. Shahbaz, Y. Geng, *et al.*, "OSNT: Open source network tester," *IEEE Network*, vol. 28, no. 5, pp. 6–12, Sep. 2014, Conference Name: IEEE Network. DOI: 10.1109/MNET.2014.6915433.

[12] F. Girela-Lopez, E. Ros, and J. Diaz, "Precise Network Time Monitoring: Picosecond-Level Packet Timestamping for Fintech Networks," en, *IEEE Access*, vol. 9, pp. 40 274–40 285, 2021. DOI: 10.1109/ACCESS.2021.3064987. [Online]. Available: https://ieeexplore.ieee.org/document/9373573/ (visited on 05/12/2023).

[13] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "MoonGen: A Scriptable High-Speed Packet Generator," in *Proceedings of the 2015 Internet Measurement Conference*, ser. IMC '15, New York, NY, USA: Association for Computing Machinery, Oct. 2015, pp. 275–287. DOI: 10.1145/2815675.2815692. [Online]. Available: http://doi.org/10.1145/2815675.2815692 (visited on 05/06/2022).

Acknowledgment

This work was partly funded by the Deutsche Forschungsgesellschaft DFG (German Reasearch Agency) under Grant No 316878574.