MoonEm — High-Precision Path Property Emulation Using DPDK

STEFAN LACHNIT, Technical University of Munich, Germany
SEBASTIAN GALLENMÜLLER, Technical University of Munich, Germany
ERIC HAUSER, Technical University of Munich, Germany
FLORIAN WIEDNER, Technical University of Munich, Germany
KILIAN HOLZINGER, Technical University of Munich, Germany
HENNING STUBBE, Technical University of Munich, Germany
THOMAS SENFTL, Technical University of Munich, Germany
GEORG CARLE, Technical University of Munich, Germany

Network path conditions, such as loss, capacity, and delay, have a significant impact on the behavior and performance of networked applications. Path property emulators are essential and widely used tools to perform evaluations under realistic conditions. However, the quality of the emulation and the potential influence on experimental results itself is rarely considered. This work highlights how the performance limitations of existing tools, such as NetEm, a network emulator based on Linux traffic control, can alter network measurements. To address these shortcomings, we introduce MoonEm, a high-performance path property emulator based on the Data Plane Development Kit (DPDK). Moreover, we present a novel approach to precisely control packet transmission times on commodity hardware. MoonEm is focused on the emulation of realistic and reproducible network conditions. Our measurements demonstrate that MoonEm achieves a maximum packet rate of 13.39 Mpps compared to 0.98 Mpps for NetEm. In contrast to NetEm, we improve latency deviation from 71.15 µs to 53 ns for the median and from 769.26 µs to 80 ns for the worst case.

CCS Concepts: • **Networks** → **Network measurement**; **Network experimentation**; *Network performance analysis*.

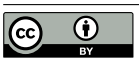
Additional Key Words and Phrases: Network Emulator, Path Emulator, NetEm, DPDK, MoonGen

ACM Reference Format:
Stefan Lachnit, Sebastian Gallenmüller, Eric Hauser, Florian Wiedner, Kilian Holzinger, Henning Stubbe, Thomas Senftl, and Georg Carle. 2025. MoonEm — High-Precision Path Property Emulation Using DPDK. *Proc. ACM Netw.* 3, CoNEXT4, Article 29 (December 2025), 21 pages. https://doi.org/10.1145/3768976

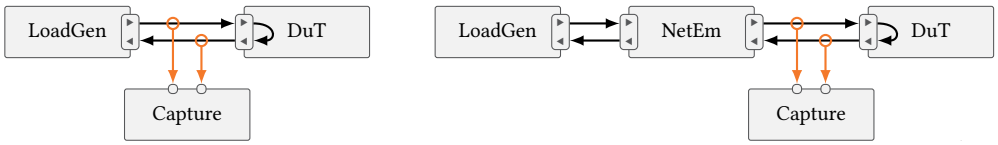
1 Introduction

The behavior of distributed applications is impacted by the properties of the underlying network. Their operating conditions are different, for instance, when working over 5G compared to a data center network. While latency and packet loss are significantly higher over a cellular connection, bandwidth is more limited. These performance limitations naturally impact measurement results.

Authors' Contact Information: Stefan Lachnit, lachnit@net.in.tum.de, Technical University of Munich, Munich, Germany; Sebastian Gallenmüller, gallenmu@net.in.tum.de, Technical University of Munich, Munich, Germany; Eric Hauser, hauser@net.in.tum.de, Technical University of Munich, Munich, Germany; Florian Wiedner, wiedner@net.in.tum.de, Technical University of Munich, Munich, Germany; Kilian Holzinger, holzinger@net.in.tum.de, Technical University of Munich, Munich, Germany; Henning Stubbe, stubbe@net.in.tum.de, Technical University of Munich, Munich, Germany; Thomas Senftl, senftl@net.in.tum.de, Technical University of Munich, Munich, Germany; Georg Carle, carle@net.in.tum.de, Technical University of Munich, Munich, Germany.



ACM 2834-5509/2025/12-ART29
https://doi.org/10.1145/3768976



(a) Direct connection between packet generator (LoadGen) and DuT (OvS)

(b) Passing traffic through an intermediate NetEm host

Fig. 1. Experiment setup of the motivating example

To ensure that realistic behavior is reflected, researchers need to replicate the network conditions as closely as possible. However, access to real-world deployments of such networks is often infeasible. Even if access is possible, constraints, such as, network stability or security and privacy concerns, severely limit the possible investigations. Additionally, many results gained in this way are irreproducible. Thus, to perform such measurements, researchers rely on alternative, reproducible approaches such as network path emulation in controlled testbed environments.

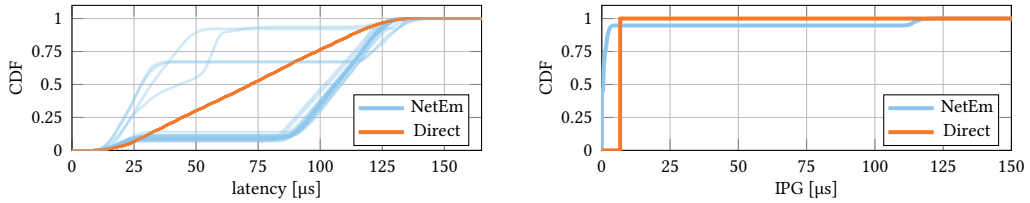
Selected papers at top-tier venues, such as SIGCOMM and CoNEXT, rely on network emulation for their results [5, 20, 36, 38, 39, 41]. However, studies [21, 26] have shown that a widely used network emulator, NetEm [16], has limitations regarding precise latency emulation, thus altering measurement results. Ever growing network bandwidths and packet rates leave less room for packet processing tasks, such as the ones performed by a network path emulator. This development makes path emulation even more challenging, justifying a thorough investigation of network path emulators.

In this paper, we extensively study three exemplary network path emulation tools, NetEm [16], TLEM [33], and DEMU [4, 31, 35], demonstrating their performance and potential shortcomings or inaccuracies. NetEm is a widely used network property emulation tool and part of the Linux traffic control. TLEM uses the netmap framework [32], utilizing patched network drivers to bypass the network stack, consequently achieving high-performance emulation. DEMU is a network path property emulator that relies on the Data Plane Development Kit (DPDK) to improve performance and latency for the emulated properties. We developed our own path emulator called MoonEm that leverages hardware features of the NIC to further increase the precision of the emulation process to provide high-precision delay, rate limiting, and packet loss emulation.

The main contributions of this paper are: (1) demonstrating the potential impact of imprecise path emulation on reported measurement results, (2) evaluating throughput and precision of commonly used path property emulation solutions, (3) proposing a novel approach to mitigate unwanted effects by precisely controlling packet delay with hardware timestamps and exact transmission timing, (4) implementing MoonEm, a path emulator integrating the proposed methodology on an extended version of the widely-used packet generator MoonGen, and (5) concluding, on the basis of extensive measurements, that MoonEm outperforms other tools in terms of timing precision, support for multiple seconds of emulated delay, and throughput. This work does not raise any ethical issues.

2 Motivation: Impact of Path Property Emulation on Measurement Results

In the following, we demonstrate the impact of path emulation on measurement results. We use NetEm to emulate network path properties. As an example of a simple, kernel-networking-based application, we selected Open vSwitch (OvS) to act as our Device under Test (DuT). We configured static rules on OvS to directly return traffic on the port it was received without any additional modifications. We compare two scenarios: (1) directly connecting the packet generator to the DuT



(a) CDF plot of the delay introduced by Open vSwitch

(b) CDF plot of the IPG before the DuT for a single experiment run

Fig. 2. Measurements for a rate of 100 Mbit/s and a packet size of 64 B

without path emulation, and (2) passing traffic through a separate host running NetEm. The two measurement setups are illustrated in Figures 1a and 1b. To determine the potential impact of path emulation, we compare the ingress and egress traffic of the DuT in both scenarios.

NetEm is running on a separate host and configured to delay the received traffic by 10 ms without modifying the traffic in any other way. The delay in the second scenario is merely introduced to enforce path emulation. The forwarding of packets on the OvS host should not be impacted by this additional delay. Therefore, we expect the same behavior of the OvS host in both scenarios, i.e., there should be no measurable difference between the ingress and egress traffic between scenarios one and two, respectively.

The experiment was executed on hosts using Intel Xeon Gold 6421N CPUs with Intel E810-XXVDA2 25G NICs for the traffic generator and the delayer host (running NetEm). The DuT is a dual-socket Intel Xeon Platinum 8568Y+ system with identical NICs. We run Debian bookworm with the 6.1.0-17 Linux kernel and OvS version 3.1.0. On the traffic generator, we used MoonGen to create 100 Mbit/s of constant bitrate (CBR) traffic with 64 B packets. We record the latency of all packets passing through the DuT using a separate capture host. The captured latency only includes delay directly introduced by OvS, excluding the delay and jitter caused by emulation. To check the repeatability and consistency of the results, we repeated the experiment 20 times.

Figure 2a shows cumulative distribution function (CDF) plots of the resulting forwarding latency of the DuT. Each line represents a single experiment run. All experiment runs, using the direct connection, create highly similar latency distributions, resulting in overlapping, almost identical plots. For these cases, latency is uniformly distributed between approx. 20 µs and 130 µs. This is caused by interrupt rate limiting functionality in the NIC [10]. Packets are buffered and only processed at a fixed interval, lower than the packet arrival rate. Therefore, packets are delayed relative to the random position in relation to these processing intervals. When passing traffic through NetEm, the latency behavior of OvS significantly differs. For most runs, packet latency is in the range from 10 µs to 27 µs for approx. 10 % of packets and between 85 µs to 130 µs for the remaining 90 %. We observed significant variations between experiment runs, with a small number of runs resulting in significantly lower latency for most packets.

To explain this effect, we plot the interpacket gap (IPG) before entering the DuT for a single experiment run in Figure 2b. At this point, packets have already passed through NetEm in the respective scenario. In the directly connected scenario, the IPG shows a nearly vertical line at 6.72 µs. This means that the LoadGen host created the requested CBR traffic with a constant IPG. The results for NetEm show that the IPG and the created traffic pattern are significantly impacted. The NetEm scenario shows a high number of packets (94.8 %) with a delay smaller than expected. Some frames were sent back-to-back and the remaining IPG values are at 115 µs—a clear sign of bursty traffic. This demonstrates that NetEm introduced bursts with an average length of 19.23 packets.

Features such as interrupt rate limiting on NICs and packet processing are susceptible to bursty traffic patterns. The processing of these bursts can cause jitter in the processing latency. The bursts introduced by NetEm affect the adaptive interrupt rate limiting functionality, which dynamically adjusts the minimum time between interrupts based on the observed receive rate. With bursts, the DuT switches between different rate limits, while it stays constant for CBR traffic. We confirmed this by repeating the same experiment with a fixed interrupt rate limit on the DuT, which did not result in the same latency artifacts visible in Figure 2a.

Our example shows that traffic patterns can be significantly altered—from CBR to bursty patterns—by software-based path property emulators. Users of path property emulators may not be aware of these changes, as typical performance indicators, such as packet loss or throughput, remain unchanged. However, the burstiness introduced by NetEm significantly changed the behavior of the DuT. This impacts measurements where the inter-packet delay and its potential impact are the actual subject of the investigation. Packet pacing is a technique used in protocols, such as TCP or QUIC, to avoid bursts in favor of a more evenly spread transmission of smaller bursts or individual packets. Using an imprecise network path emulator may severely impact the inter-packet delay or even reintroduce bursts, undermining the goal of the measurement. Time-sensitive Networking (TSN) is a collection of IEEE Ethernet extensions aiming to provide a deterministic, low-latency network service for instance for industrial control or in vehicular networks. Regular, periodic communication is typical for such environments. A network emulator that alters the inter-packet delay or aggregates packets into bursts alters or even destroys these typical patterns. To perform realistic TSN measurements, precise emulation is mandatory. Low-latency network connections, e.g., 5G ultra-reliable low-latency connection (URLLC) or the area of high-frequency trading, may be severely impacted by imprecise network path emulation. In such environments, sub-µs latencies are common. A network path emulator that, by itself, introduces latencies in the µs-range cannot be used to investigate such systems. This impact of traditional emulation on the mentioned examples poses a problem when performing precise performance analysis and motivates us to propose a novel path emulation tool.

3 Related Work

We identified three classes of tools that are relevant for our analysis: path emulators based on the Linux kernel, path emulators based on kernel bypass frameworks, and tools for high-precision packet transmission.

Kernel-based path emulation: Analogously to our motivating example, other researchers have investigated the performance and accuracy of NetEm in the past. Furthermore, as numerous tools, such as Mininet [24] and CORE [3], rely on NetEm for path property emulation, NetEm's properties have been thoroughly analyzed under various conditions. Jurgelionis et al. [21] reveal in their general analysis of NetEm that additional jitter is introduced on top of the configured delay. While NetEm accurately emulates large delays over approx. 50 ms, smaller delays vary significantly. Furthermore, multiple studies have identified limitations in NetEm's accuracy due to its kernel-space implementation [21, 22, 26, 27]. Thereby, interrupts, scheduling delays, and NIC behavior negatively affect the accuracy. For instance, packets are often transmitted in batches rather than at precisely specified times, which impacts the accuracy of emulated delays and increases jitter. Ying et al. [26] analyze NetEm's performance in containerized environments. While the overall performance is comparable to physical hosts, additional delay spikes and an increased deviation are reported. They attribute the worsened results to a different configuration of the interrupt timers. Therefore, the performance of NetEm depends on the emulation target and the used hardware [11].

Kernel-bypass path emulation: Belkhiri et al. [6] demonstrate that DPDK helps increase throughput, reduce latency, and improve accuracy. DPDK replaces the Linux kernel stack and uses polling-based drivers. Aketa et al. [4] present DEMU, a DPDK-based general-purpose path property emulator. DEMU is capable of emulating latencies in the µs domain, handling packet loss [35], packet duplication, and bandwidth limitations [31]. By combining software timestamps with DPDK, DEMU increases the latency accuracy compared to NetEm. Netmap uses an alternative approach to DPDK, by patching existing network drivers to allow bypassing the standard network stack. TLEM is a path property emulator integrated into Netmap, providing bandwidth limiting, packet loss, and latency emulation with µs precision, achieving rates over 10 Gbit/s with 64 B packets. [33]

High-precision packet transmission: As part of our previous work on the DPDK-based packet generator MoonGen [9], we analyze various methods for precisely controlling packet transmission rates. We find that the best results occur when the link is saturated at line rate, using invalid packets to fill the gaps. In this context, invalid refers to packets with an incorrect CRC checksum in the Ethernet header. The receiver automatically discards these invalid packets at the hardware level, ensuring that there is no additional impact on performance. Yu et al. [40] use a similar approach for real-time media transport using invalid packets to allow precise timing for transmission. They apply this approach for time-sensitive video traffic requiring precise transmission times. Stratmann et al. [37] demonstrate the usage of MoonGen and DPDK to emulate path properties of LTE links. They leverage MoonGen's scriptable API to adapt the packet generator into an emulator. However, their focus on LTE networks limits the applicability of their approach to general-purpose network path emulation. Our motivational example already identified significant limitations of kernel-based network path emulators. The DPDK framework, in general, and DEMU specifically claim that significant improvements are possible regarding performance and latency. However, using the invalid packet approach, high-precision transmissions are possible, which are currently not used by general-purpose network path emulators—a severe lack that we intend to fix with MoonEm.

4 Requirements

Based on the previously identified shortcomings of available tools, we identified the following requirements for MoonEm: (1) High data rates, as the developed solution should be applicable for traffic with small packet sizes resulting in high packet rates; (2) Simulation of path properties with minimal side effects on the measured system through preserving packet timing to prevent the creation of bursts and to ensure existing bursts are not separated; (3) User-friendly interface that is easily modifiable and extensible.

5 Architecture & Implementation

In this section, the approach to design and implement the high-precision and high-performance path emulation tool MoonEm is described. Initially, in Section 5.1 the MoonGen packet generator is introduced, which is the basis of the MoonEm architecture. It was updated to support modern hardware and we provide an assessment of the achievable packet rate. MoonEm is a MoonGen extension. Its pipeline architecture is detailed in Section 5.2. Then, the emulation of the considered path properties is explained. Section 5.3 describes the emulation of packet loss, Section 5.4 details the highly-precise delay emulation, and Section 5.5 outlines the implementation of rate limiting.

5.1 Upgrading MoonGen

We base our implementation on the packet generator MoonGen [9]. MoonGen is built on DPDK and allows easy and high-performance user-configurable packet generation with a Lua scripting API. Using MoonGen for our path property emulator enables us to reuse existing code for packet

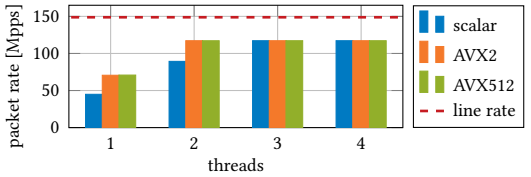


Fig. 3. Generated traffic using MoonGen with varying numbers of threads using a packet size of 64 B

generation and device configuration. Additionally, we implement selected functionality (e.g., precise packet transmission time control) as independent modules for use in other MoonGen Lua scripts.

The repository containing the implementation of MoonGen relies on an outdated version of DPDK 17.11. This version does not support current NICs and does not compile on current Linux distributions. Therefore, before using MoonGen as a base for the implementation of MoonEm, we updated the utilized DPDK library to DPDK 22.11, while also implementing and testing support for new high-performance NICs such as the 100G Intel E810 NIC. The main functionality was additionally implemented and tested for Intel X500 and X700 NICs.

MoonGen provides a feature to control IPG precisely by transmitting packets with invalid CRC checksums between valid packets. However, the Intel E810 NIC does not support sending packets without a valid CRC checksum. To invalidate packets for E810 NICs, we implemented a novel approach that invalidates frames via the EtherType field. EtherTypes smaller than 1535 specify the frame size [17]. We set an EtherType value that does not match the length of the received packet. The E810 NIC discards frames with mismatching lengths at the receiver, similar to packets with invalid CRC checksums. As these packets are sent on the wire, subsequent packets are delayed.

Achievable packet rate: An initial measurement was conducted to assess the achievable packet rate of the updated packet generator. We created a user script generating 64 B UDP traffic with sequence numbers and varying source IP addresses. Figure 3 shows the measured results using an Intel E810-CQDA2 NIC on an Intel Xeon Gold 6421N CPU. For each configuration, a 60 s measurement run was repeated 10 times. The achievable packet rate scales linearly with the number of threads up to a maximum rate of 117.18 Mpps. We are unable to achieve the theoretical maximum line rate of 148.81 Mpps with 64 B packets, even when increasing the number of threads. This is a limitation of the E810 NIC documented in the datasheet [19]. Enabling vector-based transmit functions in DPDK improved performance by 57.24 % when using a single thread, while further increasing the width of the used vector instructions with AVX512 produced negligible performance improvements.

5.2 Architecture

MoonEm is directly integrated into our new version of MoonGen. Parameters for path property emulation, including latency, packet loss, and bandwidth limits, can be configured using command line parameters. MoonEm applies the selected properties to traffic that it forwards between two physical links. It is also possible to return traffic on the same port it was received from.

The main forwarding and emulation functionality is implemented in C, while device initialization and packet crafting are handled by existing functions of MoonGen. Reusable components, such as precise packet transmission time control, or packet loss models, are implemented as independent modules. Due to this design, the emulator can be easily extended with additional functionality or adapted for other applications.

Forwarding pipeline: The high-level architecture of MoonEm is depicted in Figure 4. Packet processing is divided between a receive and a transmit thread. Multiple implementations of the

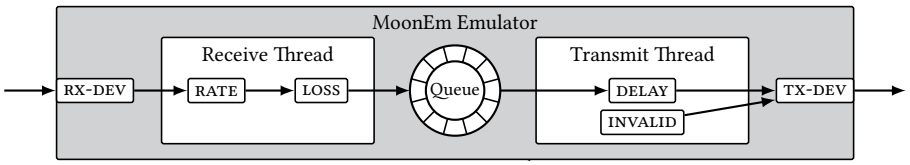


Fig. 4. Software architecture of MoonEm

receive and transmit threads, only implementing the configured properties, are dynamically selected. Therefore, deactivated features do not influence the performance and accuracy of the emulation.

The receive thread fetches bursts of packets from the NIC using the respective DPDK functions. First, it applies the rate limiting functionality described in Section 5.5, potentially dropping excess packets. Then, packets that should be dropped due to the packet loss settings (Section 5.3) are freed by this thread. Afterward, the desired transmission time is calculated based on the latency settings and packet receive timestamp (Section 5.4) and stored with the packet metadata. The received packets are timestamped in hardware, ensuring an accurate calculation of sending times, even if packets are fetched in bursts in software.

Finally, a pointer to the packet buffer is inserted into a FIFO queue. The packet data is kept in the DPDK packet buffer (mempool) until it is either transmitted or dropped. Packet data is never directly accessed by the CPU but is directly stored in memory using Direct Memory Access (DMA) and read from the same buffer when transmitted by the NIC. The number of available packet buffers must be sufficient to store the maximum possible number of delayed packets. MoonEm provides a parameter to specify the amount of storage that should be allocated for buffers associated with the receive queue. For example, allocating 256 GB for the packet buffers results in 125,000,000 packets that can be stored. This would allow us to delay 10 Gbit/s traffic with 64 B packets for up to 8.4 s.

The transmit thread continuously reads the oldest packet pointer from the FIFO queue. Packets are sent as close as possible to the desired transmission time specified in the packet metadata. We provide a hardware-assisted and a software-based implementation of the transmit thread, described in more detail in Section 5.4.

Single thread operation: As an additional optimization, we implemented a simplified architecture if no latency emulation is needed. This was done to reduce the additional latency and jitter introduced by the communication of two threads over the FIFO queue. In this architecture, a single thread reads a batch of packets, performs emulated packet drops or enforces rate limits, and transmits the remaining packets.

5.3 Packet Loss

Packet loss is implemented after considering dropped packets due to rate limits. We implement three packet loss models: (1) random packet loss, (2) the Gilbert Elliot loss model [7, 15], and (3) the 4-state Markov model also implemented in NetEm [34]. When dropping a packet due to loss, packet buffers are discarded and not inserted into the FIFO queue. With random loss, packets are dropped with a configurable constant and independent probability. With the Gilbert Elliot loss model and the 4-state Markov model, more complex and realistic packet loss scenarios, including burst losses, can be emulated. The transition probabilities of these models are configured as command line parameters of MoonEm. For configuring the 4-state Markov model, formulas for deriving parameters from more intuitive settings exist [34]. These settings include properties such as the mean burst length or the loss probability. We utilize the pseudorandom number generator wyhash64 [25] with a

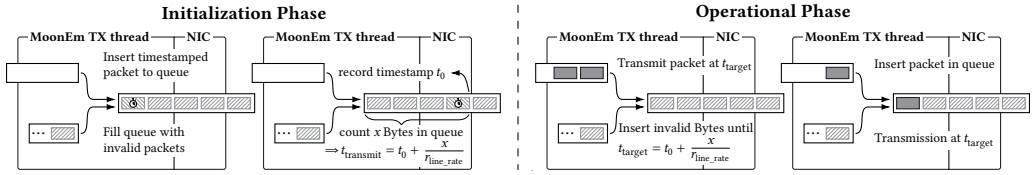


Fig. 5. Usage of invalid packets for transmission control

configurable seed for all loss model implementations. This ensures repeatable loss patterns or bursts between multiple executions of MoonEm.

5.4 Delay Emulation

The main technique of MoonEm to achieve high-precision control of packet transmission times is shown in Figure 5. The implemented approach utilizes invalid packets, which are discarded by the NIC hardware at the receiving side without affecting the host or reaching the host memory. This approach is also documented in our previous work [8]. The mode of operation consists of two phases: initialization and operation.

During initialization, the transmit queue of the NIC is filled with invalid packets. After an initial warmup period, a single invalid packet is marked to be timestamped when leaving the physical port of the NIC. Many modern NICs, such as the Intel X550 [18] and E810 [19], or Nvidia ConnectX-6 [28], ConnectX-7 [29], and ConnectX-8 [30], support the timestamping of a small number of selected transmitted packets. In the second step, invalid packets are continuously inserted into the queue following this timestamped packet. The TX thread keeps track of the cumulative length of all inserted packets. Because packets are continuously transmitted without gaps (other than a constant minimal IPG) and the line rate of the link is known, we can calculate the transmission time of the next packet that is inserted into the transmit queue. This transmission time is determined relative to the shared internal timer used for timestamping received and transmitted packets. With an initial transmit timestamp t_0 and x transmitted bytes, the transmission time of the next inserted packet is calculated according to the following equation:

$$t_{\text{transmit}} = t_0 + \frac{x}{r_{\text{line\_rate}}}$$

During regular operation, the transmit queue is always filled with a combination of normal and invalid packets, resulting in continuous transmission of line rate traffic. As receive and transmit timestamps use a shared timer, an accurate transmission time can be calculated based on the receive timestamp of packets and the configured delay. Additional invalid packets can be prepended to the NIC's transmit queue if the desired send time has not yet been reached. This allows the precise control of the transmission time for this next normal packet. When there are no normal packets to forward, only invalid packets are transmitted.

To compensate for interrupts or other temporary delays in packet transmission, MoonEm keeps the hardware transmit buffers full. Thus, even when an interrupt prevents the CPU from inserting new packets, the NIC hardware will continue sending the queued packets. To increase the robustness of the delay implementation, the size of the transmit queue should be set to a large value. However, large queue sizes delay the earliest time a new packet can be forwarded by MoonEm by the duration of sending a full transmit queue. In our measurements, we empirically determined a queue size and invalid packet size, that bounds the minimal delay time to approx. 1 ms. For specialized measurements, one can trade robustness for minimal achievable delay by adjusting this value.

Currently, we implemented this functionality only for Intel E810 NICs. The approach can be adapted for any NIC that supports timestamping of all received packets and sampling of transmit timestamps for selected packets using a common clock.

Software-based latency emulation: To support other NICs without these features, we also added an implementation using software-based timestamping. Timestamps are determined by reading the timestamp counter (TSC) register of the CPU after receiving a packet burst. For every received packet burst, the software timestamp is equivalent to the receive time of the last packet in this burst. The timestamps of all previous packets in that burst are assumed to be equivalent to receiving packets without a gap at line rate. Therefore, packets can be processed as bursts, increasing the achievable performance. Based on packet timestamps, the correct send time is calculated and stored in the metadata field of each packet. The transmitter thread delays the insertion of packets into the NIC transmit queue using busy waiting until the desired send time is reached. This causes additional jitter in the generated delay because packets are not sent immediately by the NIC after being added to the transmit queue. This additional unknown delay depends on factors such as the number of packets already in the transmit queue or additional buffering inside the NIC. Overall, the approach produces less precise results than the previously described hardware-assisted approach but does not require advanced hardware support for timestamping and sending invalid packets. Additionally, using this implementation, it is possible to emulate latency values smaller than 1 ms, as we do not rely on a full transmit queue.

The invalid packet approach differentiates MoonEm from any other path emulator such as NetEm, TLEM, or DEMU. Our previous study [8] has demonstrated that the invalid packet approach lowered the deviation from the configured packet timing to a few tens of nanoseconds. For approaches based on the kernel stack (NetEm), netmap (TLEM), or DPDK (DEMU) we observed deviations of multiple 100s of nanoseconds.

5.5 Rate Limiting

One property of realistic networks is that the achievable data rate is limited by some bottleneck on the transmission path. To emulate the resulting effects, causing dropped packets and increasing latency, we provide two rate limiting implementations with MoonEm: (1) token bucket and (2) leaky bucket, two widely used traffic shaping algorithms.

The resulting behavior resembles the behavior of typical network devices, such as switches sending traffic through a bottleneck link. Temporary bursts of traffic or exceeding the available bandwidth causes queuing of packets in internal buffers and, therefore, increases the latency of packets. The IPG is precisely controlled by relying on hardware-assisted control of transmission times. This ensures that the output traffic will never exceed the rate limit. In purely software-based systems, bursty transmission could lead to temporary violations of rate limits.

Token bucket: With token-bucket rate limiting, tokens are created and accumulated at a fixed interval. The rate at which tokens are created corresponds to the selected rate limit. Each token represents a fixed amount of data that can be transmitted when consuming this token. Our implementation performs rate limiting with the granularity of single bytes. If a packet arrives and there is not a sufficient number of tokens accumulated to transmit the complete packet, it is dropped. Instead of adding tokens based on a software-based timer, the accumulated tokens are only calculated at the arrival of packets. Using the hardware-based receive timestamp captured for every packet and the stored accumulated tokens at the receive time of the previous packet, the current tokens are calculated. This effectively results in a granularity of 1 ns for token generation, corresponding to the resolution of the receive timestamps. By using timestamps captured by the NIC hardware, the rate limiting implementation is independent of jitter and inaccuracies introduced by software-based

implementations. Token-bucket rate limiting can be combined with the latency emulation described in Section 5.4 to keep a constant delay while limiting the available rate.

Leaky bucket: The second implemented algorithm represents leaky-bucket based shaping. Based on the desired rate and the size of each transmitted packet, we calculate the earliest timestamp for the transmission of the next packet. Packet transmission times are controlled with a similar approach as described in Section 5.4. Consequently, the distance between packets is equivalent to sending the same traffic on a link with a rate corresponding to the configured rate limit. If the received rate exceeds the desired rate limit or if there are bursts in the received traffic, packets are queued, increasing their latency. If the amount of queued traffic (in bytes) exceeds a specified limit, newly received packets are dropped.

5.6 Hardware dependency

The packet loss implementation does not depend on specific hardware features. The rate limiting and delay emulation functionality requires hardware support. To determine the delay and rate limit, NICs require precise line-rate hardware timestamping. This feature is supported by Intel X550 and E810 NICs and Nvidia ConnectX-6, ConnectX-7, ConnectX-8 series NICs. For packet transmission, MoonEm provides two implementations: (1) a software-based implementation that is supported on all NICs and (2) a hardware-assisted implementation (invalid packet approach) for more precise replay that requires hardware support. Previously, we have shown that the invalid packet approach works for Intel NICs (X500-based, X700-based, and E810-based NICs [8, 9]). To check the compatibility of Nvidia NICs with the invalid packet approach, we have performed initial tests with Nvidia ConnectX-6 NICs. It was possible to send packets with an invalid EtherType, but these packets were not discarded when received by a ConnectX-6 NIC. To remove these packets, hardware filters can identify and drop them. The dropped packets did not cause any PCI transfers or CPU usage, which means that packet processing applications will not be impacted. However, the NIC hardware may behave differently, due to the increased load for parsing and filtering the invalid packets.

6 Evaluation

In this section, we present the evaluation of MoonEm, comparing its performance to the emulation tools NetEm, TLEM, and DEMU. We consider the accuracy and precision of latency emulation, the preservation of traffic patterns, and the achievable packet rates for the tested tools.

6.1 Measurement Methodology

The evaluation was performed on a hardware testbed using the pos framework [13] to make the experiments reproducible. Figure 6 shows the setup used for all measurements presented here. We used a host with dual Intel Xeon Platinum 8568Y+ for the load generator and two hosts with Intel Xeon Gold 6421N and 512 GB of RAM for the emulator and capture host. All hosts are running Debian bookworm using the 6.1.0-17 Linux kernel. All connections use 25 Gbit/s links with Intel E810-XXVDA2 NICs and single-mode fiber optic transceivers.

The hosts running the evaluated path property emulation tools were configured with optimizations to reduce latency and jitter. This includes disabling energy-saving features, isolating cores, and reducing the number of interrupts [14].

The tested tools are configured to apply path properties and return the traffic to the same port. For DEMU and TLEM to work in this scenario, we modified the code to allow forwarding to the same port from which traffic was received. Additionally, we update the DPDK version of DEMU to 22.11, to avoid performance differences caused by the DPDK versions and make it usable with our

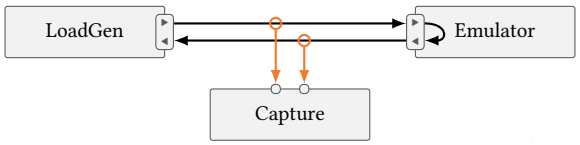


Fig. 6. Measurement hardware setup

experiment setup. TLEM was configured using patched drivers for the Intel E810 NICs and a fixed interrupt moderation setting of 15 µs as suggested in [33]. As a load generator, MoonGen is used with a custom Lua script that creates UDP packets with an incrementing packet ID in its datagram payload. The generated traffic consists either of equidistant packets or of equally spaced packet bursts with a fixed number of packets. For precise IPGs, the invalid-packet-based rate limiting method of MoonGen is used.

We use passive fiber-optical splitters to redirect copies of all packets from both communication directions between load generator and emulator to a capturing host. Because the splitter is a passive component, capturing packets using this setup does not influence the timing of packets or the behavior of the observed devices. Using a MoonGen script, all packets are received and processed to only store packet IDs with the corresponding hardware-based arrival timestamp. The two input ports on the capturing host are located on the same NIC and, therefore, share the same internal timestamping clock. The latency of a packet is calculated based on the difference between timestamps of corresponding packets on both ports. By matching packets captured at both communication directions based on the packet ID, it is possible to determine the latency introduced by the emulator with high precision and accuracy.

In our initial measurements, NetEm showed inconsistent performance results when handling large traffic volumes with multiple flows. Further analysis using profiling showed that in its default configuration, packets received on multiple cores cause contention on the lock of the queue handling packet processing in NetEm. This results in performance degradations when traffic is distributed between multiple CPU cores.

Therefore, we only consider a single flow in all presented measurements. This results in packets being directed to a single CPU core by Receive Side Scaling (RSS). Consequently, the results shown for NetEm represent the achievable single-core performance. This distinction only has a relevant impact on the results for NetEm, as MoonEm, TLEM, and DEMU do not distribute traffic between multiple CPU cores, regardless of the number of received flows. Additionally, this prevents packet reordering and deviations in latency due to packets being processed by different CPU cores.

6.2 Latency

In this section, we evaluate the latency emulation functionality of the compared tools. We focus on emulating a constant delay, as present when, e.g., emulating propagation delays through a cable. For MoonEm, we separately compare the hardware-assisted latency emulation method (called MoonEm) and the approach based on software timestamping (abbreviated as MoonEm SW).

6.2.1 Latency accuracy and precision. First, we look into the accuracy and precision of delay emulation. We consider the deviation of measured latencies from a constant configured delay value. We defined accuracy as the closeness of the median latency from the configured delay [12]. Precision is defined as the deviation of measurement results from the median of all measurements. In particular, precision also encompasses large worst-case delays of a small number of packets. We determine the latency deviation introduced for all tested emulation tools using the measurement setup described in Section 6.1.

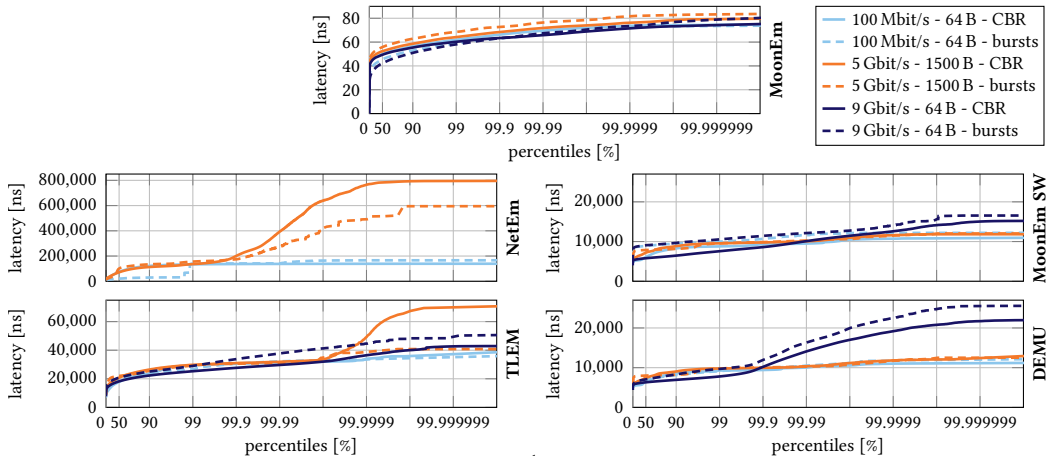


Fig. 7. HDR plot showing the deviation from the configured latency

We configured a constant delay of 10 ms. Three combinations of packet sizes and data rates are tested. 100 Mbit/s traffic with 64 B packets and 5 Gbit/s traffic with 1500 B are selected because all compared tools can process these rates without causing overload conditions and dropping packets. The scenario with 9 Gbit/s of 64 B packets was selected, as it results in the maximum packet rate that can be used with TLEM, DEMU and MoonEm without overload. We do not present results for this scenario with NetEm, as NetEm is not able to process the resulting packet rate. This would result in a large number of dropped packets and a significant increase in latency for the remaining packets due to filling buffers. To consider the effect of bursty traffic, the measurements were executed with CBR traffic and bursty traffic using a burst size of 64.

Figure 7 shows the results of these measurements. First, we consider deviation when applying latency to CBR traffic. For the median generated latencies, MoonEm stays the closest to the configured delay with a deviation of 53 ns in the worst measurement run. DEMU and MoonEm SW deviate by 7.45 µs and 7.21 µs in the median respectively. TLEM achieved a deviation for median latencies of at most 20.78 µs. We observed the largest deviation from the desired median delay of 75.15 µs when using NetEm. When considering the deviation for the worst-case packets we measure 80 ns for MoonEm, 796.26 µs for NetEm, 71.44 µs for TLEM, 15.23 µs for MoonEm SW, and 22.00 µs for DEMU. In our measurements, the hardware-assisted latency implementation in MoonEm achieved the highest precision. For MoonEm, all packet latencies are within a range of only 76 ns. For NetEm, MoonEm SW, and DEMU, there is an increase in latency deviation starting at approx. 99.9 % of packets. For TLEM, an increase can be observed starting at 99.999 %. This increase is largest with 5 Gbit/s of 1500 B packets, when using TLEM. For the other three affected emulators, the impact of these effects increases with higher data rates. DEMU and MoonEm SW show similar latency behavior because they implement latency using a similar approach based on software timestamps. DEMU has 44.47 % higher latency deviation than MoonEm SW for percentiles above 99.9 % at a rate of 9 Gbit/s, while the difference is only 2.15 % at 100 Mbit/s.

We also measured the latency deviation when applying the tested emulators to bursty traffic. Bursts cause temporary buffering of packets in the NIC receive queues, resulting in inaccurate software receive timestamps. Additionally, full transmit queues further delay packets before they are sent by the NIC. This effect can also be observed in our measurements. For MoonEm SW and DEMU, latency increases for all packet sizes and rates when processing bursty traffic. With TLEM an

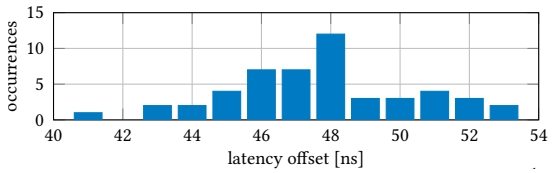


Fig. 8. Histogram of median latency deviations for MoonEm with 1500 B packets and a rate of 5 Gbit/s

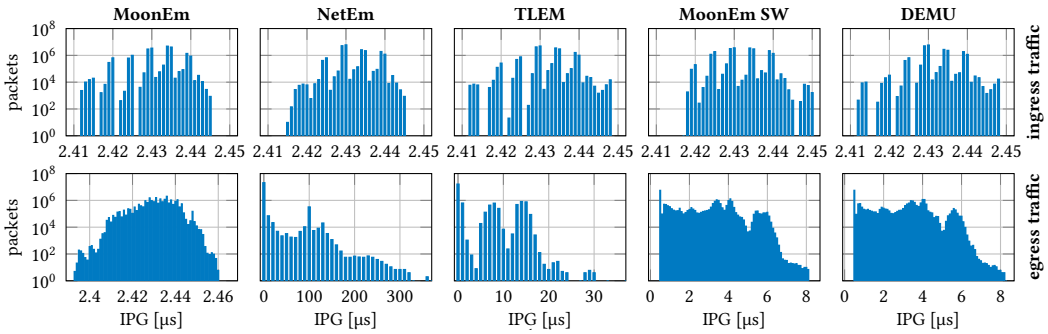


Fig. 9. Histograms showing the distribution of IPG values before and after the latency emulator for 5 Gbit/s CBR traffic using a packet size of 1500 B

increase in latency for high packet rates can be observed. With 1500 B packets and a rate of 5 Gbit/s we measured a decrease of the maximum latency by 42.86 %. For low packet rates, bursty traffic did not introduce significant changes when using TLEM. When using NetEm with 100 Mbit/s of traffic, bursts decreased the latency for most packets and increased latency for the high percentiles. For 5 Gbit/s, most packets were delayed more, while the maximum introduced delay decreased. A possible explanation for this unexpected effect is dynamic interrupt handling or behavior of the Linux network stack (e.g., the NAPI). MoonEm mitigates this issue by using hardware-based receive timestamps and precise transmission time control using the approach described in Section 5.4.

For MoonEm, the median latency deviation between measurement runs was independent of rate and packet size within a range of 13 ns. To further investigate this random deviation, we perform an additional experiment. The measurement scenario using 1500 B packets with 5 Gbit/s traffic was repeated 50 times while capturing the median latency deviation from the constant 10 ms delay. The resulting median values are presented in a histogram in Figure 8. In these runs, all deviation values are within a range of 12 ns, while deviations around the center of 47 ns are most common. This matches the median latency differences between measurement runs visible in the MoonEm results of Figure 7.

6.2.2 Influences on packet timing. As already illustrated in the motivational example in Section 2, packet timing can significantly impact measurement outcomes. It is desirable for a measurement tool not to change packet timing in unintended ways. Delay implementations should only add a constant delay without influencing other timing properties, such as IPG or burstiness. Therefore, we investigated how the tested path property emulation tools influence IPG values when configuring a constant delay.

CBR traffic: First, we examine how IPG values are affected when delaying CBR traffic. We pass 5 Gbit/s of uniformly spaced 1500 B packets through the evaluated tools set to a constant delay of

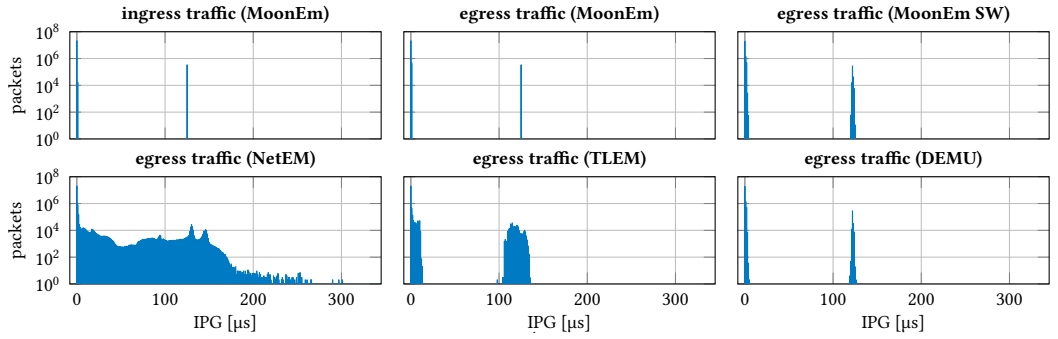


Fig. 10. Histograms showing the distribution of IPG values before and after the latency emulator for 5 Gbit/s bursty traffic using a packet size of 1500 B

10 ms. MoonGen, as a packet generator, causes slight IPG variations when creating CBR traffic. This can be seen in the first row of plots in Figure 9. The y-axis on these graphs uses a logarithmic scale to also make the small number of outliers visible. For all five experiment scenarios, all IPG values for generated traffic are within a small range of 36 ns around the expected value of 2432 ns.

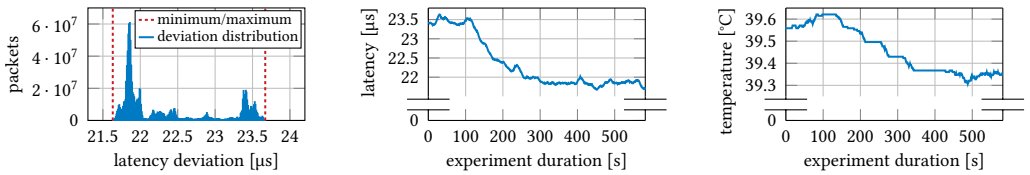
In the bottom row of graphs in Figure 9, histograms for the IPG values after passing through the evaluated path property emulation tools are shown. The histograms use different bin sizes as the resulting IPG values differ by orders of magnitude. MoonEm uses a bin size of 1 ns, MoonEm SW and DEMU use a bin size of 100 ns, TLEM uses 1 µs, while NetEm uses 10 µs.

MoonEm keeps the received packet timing while only introducing minimal jitter. IPG values remain within a range of 67 ns around the original IPG value. MoonEm SW and DEMU produce nearly identical results utilizing a similar latency implementation. 26.9 % of packets are sent with a significantly lower IPG value in the 500 ns bin, while only 0.5 % of packets are in the bin corresponding to the IPG value of the original CBR traffic. The worst-case deviation from the original IPG is 8.13 µs for MoonEm SW and 8.34 µs for DEMU. NetEm created the largest variations from minimal line rate IPG values up to 416 µs, with latencies present in all intermediate histogram bins. Additionally, a significant number of packets (1.53 %) with an IPG of 100 µs was introduced. For TLEM over 99 % of packets showed an IPG variation between 0 µs and 15 µs, while a small number of individual packets deviate by up to 61 µs.

Bursty traffic: In addition to not introducing bursts, latency emulation tools should also preserve bursts in processed traffic. To test this scenario, we pass 5 Gbit/s of traffic with 1500 B packets in bursts of 64 contiguous packets through the evaluated tools. A constant delay of 10 ms is configured. The resulting IPG values are presented in Figure 10. All histograms use a bin width of 1 µs.

The graph on the top left shows the IPG values of traffic before passing through the emulation tools. As the pattern for all five experiment runs is identical, we only show results for the experiment run using MoonEm. The remaining histograms are presented in Appendix A. A single bin at 125 µs corresponds to the gap between bursts, while the remaining packets are sent consecutively, resulting in IPG values at 0 µs and 1 µs.

The five other graphs show the IPG distribution after passing through the emulators. For MoonEm, packet timing is not significantly influenced. IPG values are still mapped to the identical bins. Latency variations introduced by MoonEm SW, TLEM, and DEMU cause variations in the IPG values of packets inside and between bursts. For MoonEm SW previously consecutive packets now occupy bins from 0 µs to 4 µs, while the IPG values between bursts range from 120 µs to 125 µs. DEMU behaved similarly with latency values from 0 µs to 5 µs and 119 µs to 126 µs. TLEM introduced larger



(a) latency deviation histogram (b) latency deviation over time (c) temperature variations

Fig. 11. Latency results of MoonEm with a configured latency of 10 s, a rate of 24.9 Gbit/s, with 1500 B packets

variations, with consecutive packets resulting in IPGs of up to 13 µs, while the distance between bursts now spans a range of 37 µs around the expected value.

For NetEm, all histogram bins between 0 µs and 217 µs are filled, while individual packets showed an IPG of up to 301 µs. A significant number of packets still correspond to the original IPG values at 0 µs. The IPG between bursts at 125 µs is no longer clearly visible. Instead, IPG values cover multiple bins with peaks at 130 µs and 146 µs.

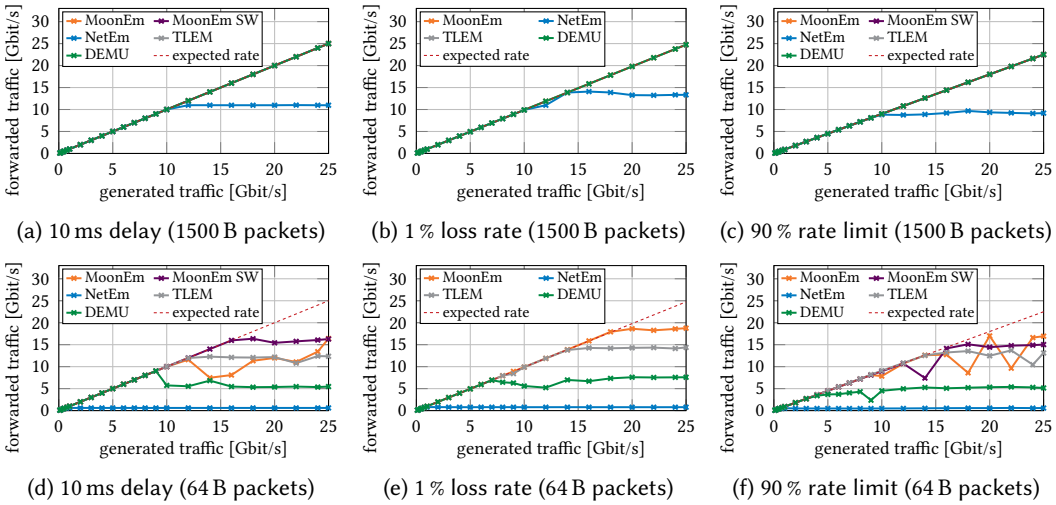
In conclusion, these measurements show that NetEm significantly influences packet timing, both creating bursty traffic from CBR traffic and changing the burst timing of bursty traffic. DEMU, TLEM, and MoonEm SW introduce jitter while keeping the general bursting behavior intact. Our experiment confirms that MoonEm precisely keeps the packet timing and bursts when emulating latency.

Other traffic patterns: CBR and bursty are the two extreme examples for traffic patterns rarely observed in their pure form. The Poisson process is a simple model where the length of the inter-packet delay is exponentially distributed, i.e., shorter inter-packet delays are more probable than longer delays. Though a Poisson process cannot approximate realistic traffic on longer timescales, its approximation is suitable for sub-second timescales [23]. We already demonstrated that the invalid-packet approach of MoonGen supports the generation of inter-packet delays according to a Poisson process [8]. As this process requires the precise generation of *arbitrary* inter-packet delay, the approach is also suitable to create arbitrary distributions. We also showed that the limited precision software-based approaches—kernel, netmap, or DPDK—lead to a lower performance creating specified delays.

6.2.3 Emulation of large delays. In some applications, it is necessary to emulate paths with high latencies and data rates. Examples include satellite or space communication, where latency values in the range of several hundred milliseconds to many seconds are possible. Some emulators, such as DEMU, are limited in the number of packets they can buffer, consequently limiting the maximum achievable packet rate at large delay settings.

To show the applicability of MoonEm for these scenarios, we perform a measurement with a delay of 10 s over a duration of 600 s. We generate 24.9 Gbit/s traffic using 1500 B packets. This results in a bandwidth-delay product of 31.125 GB that needs to be buffered by the emulation tool. A rate lower than the line rate is selected to be able to adjust packet timing with invalid packets. When delaying line rate traffic, only valid packets can be transmitted, reducing the robustness of our implementation.

The distribution of the measured delay deviations from the configured value is presented in the CDF plot in Figure 11a. All measured delay deviations range from 21.63 µs to 23.67 µs. We observe larger absolute errors compared to the measurements with small emulated latencies in Section 6.2.1. This can be explained by inaccuracies of the internal clocks of the emulator NIC or capturing NIC



(a) 10 ms delay (1500 B packets) (b) 1 % loss rate (1500 B packets) (c) 90 % rate limit (1500 B packets)

(d) 10 ms delay (64 B packets) (e) 1 % loss rate (64 B packets) (f) 90 % rate limit (64 B packets)

Fig. 12. Achievable throughput for independently enabled path properties

increasing in impact with the larger delay of 10 s. An absolute latency deviation of 22 µs with a configured delay of 10 s corresponds to 2.2 ppm. Therefore, the relative deviation is lower than the results observed with smaller latencies in Section 6.2.1 equivalent to 5.3 ppm of deviation. This improvement is likely caused by small absolute latency errors independent of the clock speed.

To further investigate the patterns observed in Figure 11a and to confirm the cause of the delay variation, we plot the average latency deviation over the experiment duration of 10 min. The results are presented in Figure 11b. Average latency values fluctuate over time, starting at the maximum observed latency deviation of 23.67 µs. In the remaining time of the experiment, the delay variation decreases while approaching a value of 21.9 µs. One possible cause for these variations over time is varying temperatures due to the cooling behavior of the server. These temperature variations can influence the clock speeds of oscillators on the NIC of the delaying host. As the clockspeed also influences the transmission rate of packets at line rate, the emulated delay is also affected.

To confirm this idea, we recorded the temperature reported by sensors on the transceivers of the delayer host during the experiment. We use these sensors as an indicator for possible temperature fluctuations of the relevant components on the NIC. Figure 11c shows the recorded temperatures over time since the start of the experiment. While the temperature only changes by 0.31 °C, the general shape of the graph closely matches the observed latency variations.

Overall, the presented measurements confirm that MoonEm can be used for applications requiring long delays and large bandwidth-delay products with small deviations in the range of 2.2 ppm.

6.3 Throughput

One use case of path property emulation tools is evaluating high-performance packet processing systems. Therefore, they need to handle high data and packet rates typically present in those systems. Exceeding the maximum bandwidth of an emulation tool may cause severe deviation from the configured properties or the dropping of excess traffic.

First, we consider the three evaluated properties independently: delay, random loss, and rate limiting. We generate CBR traffic with increasing rates starting from 100 Mbit/s to the line rate of 25 Gbit/s. Two packet sizes were compared: a typical Ethernet MTU size of 1500 B and minimally sized 64 B packets. For each combination of packet rate and size, we perform a 60 s measurement

run. We determine the average received data rate based on hardware packet counters on the NIC of the load generator.

The graphs in Figure 12 show the results of these measurements. As the packet loss implementations of MoonEm and MoonEm SW are identical, we only show the results for MoonEm in the respective graphs. For each property, the expected forwarded traffic during regular operation differs. We configure a constant delay of 10 ms for the evaluation of delay emulation functionality. In this case, no packet should be dropped. For testing packet loss, an independent random loss of 1 % is configured, reducing the expected returned traffic by this percentage. To determine the achievable throughput for the rate limiting functionality, we set the rate limit to 90 % of the generated traffic in the respective experiment run. Consequently, the expected returned traffic is 90 % of the generated traffic. When the received traffic is lower than the expected value, the emulator is overloaded. In this state, configured path properties will no longer hold, for example, causing significant increases in delay due to buffering. The reported bandwidth values in this section represent bandwidth at the input of the emulation tool.

With 1500 B packets, MoonEm, MoonEm SW, TLEM, and DEMU achieve the line rate of 25 Gbit/s for all three evaluated properties. Figures 12a, 12b, and 12c visualize the results. NetEm cannot process line rate traffic of 25 Gbit/s. The maximum rate without additional packet drops is 10 Gbit/s (822.37 kpps) for delay and rate limiting and 14 Gbit/s (1.15 Mpps) for packet loss emulation. Increasing the rate further causes NetEm to drop all packets exceeding this maximum rate. For packet loss emulation, there was an anomaly with 12 Gbit/s of traffic causing packet drops, even though 14 Gbit/s could be handled without additional drops.

Using a packet size of 64 B reduces the achievable rate with all tested tools. As operations for path property emulation do not process the packet content, performance mainly depends on the packet rate. With 64 B packets, none of the compared implementations can reach the line rate of 25 Gbit/s. This allows us to compare the achievable performance of property emulation with the tested emulators.

First, we consider delay, depicted in Figure 12d. NetEm achieves a rate of 600 Mbit/s equivalent to a packet rate of 892.86 kpps slighly higher than the 822.37 kpps for 1500 B packets. The maximum data rate was 9 Gbit/s for DEMU, 10 Gbit/s for MoonEm, and 12 Gbit/s for TLEM, while MoonEm SW achieved the highest rate at 16 Gbit/s.

The performance of packet loss emulation for small packet sizes is evaluated in Figure 12e. NetEm achieves a rate of 750 Mbit/s (1.12 Mpps), a nearly identical packet rate compared to the results with larger packet sizes. DEMU can process 7 Gbit/s without dropping additional packets, while MoonEm can reach 18 Gbit/s. For TLEM, there was an anomaly at 9 Gbit/s causing excess drops, even though increasing the rate further showed that up to 14 Gbit/s could be reached. All tested tools behaved consistently in the overloaded state, dropping excess packets.

The results for rate limiting are shown in Figure 12f. NetEm can handle 500 Mbit/s (744.5 kpps). DEMU achieves 4 Gbit/s, while MoonEm can rate limit traffic up to 9 Gbit/s. MoonEm SWs rate limiting implementation reaches a throughput of 12 Gbit/s. The highest achieved rate was possible with TLEMs with a bandwidth of 14 Gbit/s. Both MoonEm rate implementations behaved inconsistently when increasing the rate over their maximum rate.

In some applications, a combination of multiple path properties should be emulated. To assess the performance when combining the independently considered functions, we performed an experiment with a combination of properties. We use a delay of 10 ms, a random packet loss of 1 %, and a rate limit of 90 % of the generated traffic. We only show results for 64 B packets because previous measurements show that the performance mainly depends on the packet rate. The results are plotted in Figure 13. With TLEM there is no significant decrease in performance, compared to emulating properties independently, still reaching a rate of 12 Gbit/s (17.86 Mpps), identical to

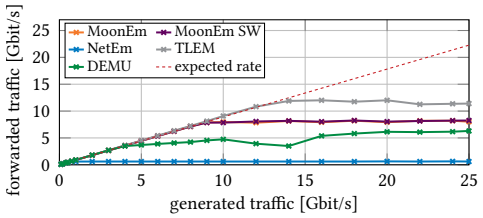


Fig. 13. Throughput when combining delay, packet loss and rate limiting (64 B packets)

emulating only packet loss. The achievable rate for all other tested tools decreases compared to emulating individual properties. NetEm achieved a rate of 660 Mbit/s (982.14 kpps). DEMU could apply the desired properties to traffic up to 4 Gbit/s (5.95 Mpps). In this scenario, MoonEm and MoonEm SW achieved nearly identical results at 9 Gbit/s (13.39 Mpps). The similar performance can be explained by a bottleneck in the receiver thread of MoonEm handling packet loss and rate limiting, which is nearly identical for both implementations.

Overall, we show that the throughput of path property emulation strongly depends on the packet size. Both versions of our implementation outperform NetEm and DEMU for all three considered properties, while achieving similar performance to TLEM.

7 Conclusion

Our initial case study shows that network path emulators may change the outcome of an experiment. These alterations seem insignificant and experimenters may not even be aware of this alteration, which makes them all the more challenging to detect and prevent.

We implemented three features in MoonEm: loss, rate-limit, and delay emulation. We demonstrated severe shortcomings for NetEm with traffic rates as low as 1 Gbit/s in selected scenarios. DEMU, TLEM, and MoonEm offer consistent line-rate emulation for 1500-byte packets up to 25 Gbit/s, and multi-gigabit rates for minimum-sized packets. Kernel-based network path emulators significantly alter the delay by up to 800 µs, with kernel-bypass-based emulators showing a clear advantage keeping the additional worst-case delay below 100 µs. For applications requiring precise latency emulation and packet timing, MoonEm further improves this delay, with worst-case alterations lower than 100 ns. DEMU, TLEM, and MoonEm have shown superior performance compared to kernel-based tools such as NetEm. MoonEm additionally provides high-precision latency emulation, significantly improving network measurements over state-of-the-art tools. We released the source code of the updated version of MoonGen, including MoonEm (cf. Appendix B). Thus, MoonEm can help create real-world performance figures with researcher-friendly pricing.

For future work, we aim to extend MoonEm to support Nvidia NICs and investigate the performance of Nvidia- and Intel-based NICs. SmartNICs offer programmable processing paths directly on the hardware with the potential to extend the capabilities and precision of MoonEm. These programmable NICs present an interesting target for investigation.

Acknowledgments

This work was funded by the EU Horizon Europe programme, projects SLICES-PP (101079774) and GreenDIGIT (101131207), by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), project HyperNIC (CA595/13-1), and by the Bavarian Ministry of Economic Affairs, Regional Development and Energy, project 6G Future Lab Bavaria.

References

[1] 2025. Artifacts for paper "MoonEm - High-Precision Path Property Emulation Using DPDK". https://github.com/tumi8/moonem-artifacts Last accessed: 2025-10-01.

[2] 2025. MoonGen. https://github.com/tumi8/moongen Last accessed: 2025-10-01.

[3] Jeff Ahrenholz, Claudiu Danilov, Thomas R Henderson, and Jae H Kim. 2008. CORE: A real-time network emulator. In *MILCOM 2008-2008 IEEE Military Communications Conference*. IEEE, 1–7.

[4] Shuhei Aketa, Takahiro Hirofuchi, and Ryousei Takano. 2017. DEMU: A DPDK-based network latency emulator. In *2017 IEEE International Symposium on Local and Metropolitan Area Networks, LANMAN 2017, Osaka, Japan, June 12-14, 2017*. IEEE, 1–6. doi:10.1109/LANMAN.2017.7972145

[5] Mahdi Arghavani, Haibo Zhang, David M. Eyers, and Abbas Arghavani. 2024. SUSS: Improving TCP Performance by Speeding Up Slow-Start. In *Proceedings of the ACM SIGCOMM 2024 Conference, ACM SIGCOMM 2024, Sydney, NSW, Australia, August 4-8, 2024*. ACM, 151–165. doi:10.1145/3651890.3672234

[6] Adel Belkhiri, Martin Pépin, Mike Bly, and Michel R. Dagenais. 2023. Performance analysis of DPDK-based applications through tracing. *J. Parallel Distributed Comput.* 173 (2023), 1–19. doi:10.1016/J.JPDC.2022.10.012

[7] E. O. Elliott. 1963. Estimates of error rates for codes on burst-noise channels. *The Bell System Technical Journal* 42, 5 (1963), 1977–1997. doi:10.1002/j.1538-7305.1963.tb00955.x

[8] Paul Emmerich, Sebastian Gallenmüller, Gianni Antichi, Andrew W. Moore, and Georg Carle. 2017. Mind the Gap - A Comparison of Software Packet Generators. In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS 2017, Beijing, China, May 18-19, 2017*. IEEE Computer Society, 191–203. doi:10.1109/ANCS.2017.32

[9] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. MoonGen: A Scriptable High-Speed Packet Generator. In *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, October 28-30, 2015*, Kenjiro Cho, Kensuke Fukuda, Vivek S. Pai, and Neil Spring (Eds.). ACM, 275–287. doi:10.1145/2815675.2815692

[10] Paul Emmerich, Daniel Raumer, Sebastian Gallenmüller, Florian Wohlfart, and Georg Carle. 2018. Throughput and Latency of Virtual Switching with Open vSwitch: A Quantitative Analysis. *J. Netw. Syst. Manag.* 26, 2 (2018), 314–338. doi:10.1007/S10922-017-9417-0

[11] Joachim Fabini, Peter Reichl, Christoph Egger, Marco Happenhofer, Michael Hirschbichler, and Lukas Wallentin. 2008. Generic access network emulation for NGN testbeds. In *4th International Conference on Testbeds & Research Infrastructures for the DEvelopment of NeTworks & COMmunities (TRIDENTCOM 2008), March 18-20, 2008, Innsbruck, Austria*, Miguel Ponce de Leon (Ed.). ICST, 43. doi:10.4108/TRIDENTCOM.2008.3135

[12] International Organization for Standardization. 2023. *ISO 5725-1:2023: Accuracy (trueness and precision) of measurement methods and results-Part 1: General principles and definitions.*

[13] Sebastian Gallenmüller, Dominik Scholz, Henning Stubbe, and Georg Carle. 2021. The pos Framework: A Methodology and Toolchain for Reproducible Network Experiments. In *CoNEXT '21: The 17th International Conference on emerging Networking EXperiments and Technologies, Virtual Event, Munich, Germany, December 7 - 10, 2021*. ACM, 259–266. doi:10.1145/3485983.3494841

[14] Sebastian Gallenmüller, Florian Wiedner, Johannes Naab, and Georg Carle. 2023. How Low Can You Go? A Limbo Dance for Low-Latency Network Functions. *J. Netw. Syst. Manag.* 31, 1 (2023), 20. doi:10.1007/S10922-022-09710-3

[15] E. N. Gilbert. 1960. Capacity of a burst-noise channel. *The Bell System Technical Journal* 39, 5 (1960), 1253–1265. doi:10.1002/j.1538-7305.1960.tb03959.x

[16] Stephen Hemminger et al. 2005. Network emulation with NetEm. In *Linux conf au*, Vol. 5. 2005.

[17] IEEE. 1997. IEEE Standards for Local and Metropolitan Area Networks: Specification for 802.3 Full Duplex Operation. *IEEE Std 802.3x-1997 and IEEE Std 802.3y-1997 (Supplement to ISO/IEC 8802-3: 1996/ANSI/IEEE Std 802.3, 1996 Edition)* (1997), 1–324.

[18] Intel. 2023. Intel Ethernet Controller X550 Datasheet. Revision 2.7.

[19] Intel. 2024. Intel Ethernet Controller E810 Datasheet. Revision 2.8.

[20] Tony John, Piet De Vaere, Caspar Schutijser, Adrian Perrig, and David Hausheer. 2021. Linc: low-cost inter-domain connectivity for industrial systems. In *SIGCOMM '21: ACM SIGCOMM 2021 Conference, Virtual Event, August 23-27, 2021, Poster and Demo Sessions*, Marco Chiesa, David R. Choffnes, Athina Markopoulou, and Marinho P. Barcellos (Eds.). ACM, 68–70. doi:10.1145/3472716.3472850

[21] Audrius Jurgelionis, Jukka-Pekka Laulajainen, Matti Hirvonen, and Alf Inge Wang. 2011. An Empirical Study of NetEm Network Emulation Functionalities. In *Proceedings of 20th International Conference on Computer Communications and Networks, ICCCN 2011, Maui, Hawaii, USA, July 31 - August 4, 2011*, Haohong Wang, Jin Li, George N. Rouskas, and Xiaobo Zhou (Eds.). IEEE, 1–6. doi:10.1109/ICCCN.2011.6005933

[22] Paraskevas Karachatzis, Jan Ruh, and Silviu S. Craciunas. 2023. An Evaluation of Time-triggered Scheduling in the Linux Kernel. In *Proceedings of the 31st International Conference on Real-Time Networks and Systems, RTNS 2023, Dortmund, Germany, June 7-8, 2023*. ACM, 119–131. doi:10.1145/3575757.3593660

[23] Thomas Karagiannis, Mart L. Molle, Michalis Faloutsos, and Andre Broido. 2004. A Nonstationary Poisson View of Internet Traffic. In *Proceedings IEEE INFOCOM 2004, The 23rd Annual Joint Conference of the IEEE Computer and Communications Societies, Hong Kong, China, March 7-11, 2004*. IEEE, 1558–1569. doi:10.1109/INFCOM.2004.1354569

[24] Karamjeet Kaur, Japinder Singh, and Navtej Singh Ghumman. 2014. Mininet as software defined networking testing platform. In *International conference on communication, computing & systems (ICCCS)*. IEEE, 139–42.

[25] Daniel Lemire. 2019. wyhash64. https://github.com/lemire/testingRNG/commits/42a3a76feef1126d632f7a56181dacb77ba1ccc7/source/wyhash.h Last accessed: 2025-10-01.

[26] Ying Li, Radim Bartos, and Chunchao Liang. 2019. Are Containers Coupled with NetEm a Reliable Tool for Performance Study of Network Protocols?. In *2019 SoutheastCon*. 1–7. doi:10.1109/SoutheastCon42311.2019.9020466

[27] Robert Lübke, Peter Büschel, Daniel Schuster, and Alexander Schill. 2014. Measuring accuracy and performance of network emulators. In *IEEE International Black Sea Conference on Communications and Networking, BlackSeaCom 2014, Odessa, Ukraine, May 27-30, 2014*. IEEE, 63–65. doi:10.1109/BLACKSEACOM.2014.6849005

[28] Nvidia. 2024. ConnectX-6 Dx 100G/200G Ethernet NIC. Datasheet, JAN24.

[29] Nvidia. 2025. ConnectX-7 400G Adapters. Datasheet, JUL25.

[30] Nvidia. 2025. ConnectX-8 SuperNIC. Datasheet, SEP25.

[31] Chayapon Puakalong, Ryousei Takano, Vasaka Visoottiviseth, Assadarat Khurat, and Wudhichart Sawangphol. 2020. A Network Bandwidth Limitation with the DEMU Network Emulator. In *2020 IEEE 10th Symposium on Computer Applications & Industrial Electronics (ISCAIE)*. 151–154. doi:10.1109/ISCAIE47305.2020.9108794

[32] Luigi Rizzo. 2012. netmap: A Novel Framework for Fast Packet I/O. In *Proceedings of the 2012 USENIX Annual Technical Conference, USENIX ATC 2012, Boston, MA, USA, June 13-15, 2012*, Gernot Heiser and Wilson C. Hsieh (Eds.). USENIX Association, 101–112. https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/rizzo

[33] Luigi Rizzo, Giuseppe Lettieri, and Vincenzo Maffione. 2016. Very high speed link emulation with TLEM. In *IEEE International Symposium on Local and Metropolitan Area Networks, LANMAN 2016, Rome, Italy, June 13-15, 2016*. IEEE, 1–6. doi:10.1109/LANMAN.2016.7548841

[34] Stefano Salsano, Fabio Ludovici, Alessandro Ordine, and D Giannuzzi. 2012. Definition of a general and intuitive loss model for packet networks and its implementation in the Netem module in the Linux kernel. *University of Rome «Tor Vergata». Version* 3 (2012).

[35] Kanon Sasaki, Takahiro Hirofuchi, Saneyasu Yamaguchi, and Ryousei Takano. 2019. An Accurate Packet Loss Emulation on a DPDK-based Network Emulator. In *AINTEC '19, Proceedings of the Asian Internet Engineering Conference, 7-9 August 2019, Andaman Cannacia Resort, Phuket, Thailand*. ACM, 1–8. doi:10.1145/3340422.3343635

[36] Markus Sosnowski, Florian Wiedner, Eric Hauser, Lion Steger, Dimitrios Schoinianakis, Sebastian Gallenmüller, and Georg Carle. 2023. The Performance of Post-Quantum TLS 1.3. In *Companion of the 19th International Conference on emerging Networking EXperiments and Technologies, CoNEXT 2023, Paris, France, December 5-8, 2023*, Dario Rossi, Stefano Secci, Olivier Bonaventure, and Lili Qiu (Eds.). ACM, 19–27. doi:10.1145/3624354.3630585

[37] Lars Stratmann, Brenton D. Walker, and Vu Anh Vu. 2020. Realistic Emulation of LTE With MoonGen and DPDK. In *WiNTECH@MobiCom 2020: Proceedings of the 14th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization, London, UK, September 25, 2020*. ACM, 87–94. doi:10.1145/3411276.3412183

[38] Ammar Tahir, Prateesh Goyal, Ilias Marinos, Mike Evans, and Radhika Mittal. 2024. Efficient Policy-Rich Rate Enforcement with Phantom Queues. In *Proceedings of the ACM SIGCOMM 2024 Conference, ACM SIGCOMM 2024, Sydney, NSW, Australia, August 4-8, 2024*. ACM, 1000–1013. doi:10.1145/3651890.3672267

[39] Mathieu Xhonneux, Fabien Duchene, and Olivier Bonaventure. 2018. Leveraging eBPF for programmable network functions with IPv6 segment routing. In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies, CoNEXT 2018, Heraklion, Greece, December 04-07, 2018*, Xenofontas A. Dimitropoulos, Alberto Dainotti, Laurent Vanbever, and Theophilus Benson (Eds.). ACM, 67–72. doi:10.1145/3281411.3281426

[40] Ping Yu, Yi Wang, Ming Li, Jianxin Du, and Raul Diaz. 2023. RMTS: A Real-time Media Transport Stack Based on Commercial Off-the-shelf Hardware. In *Proceedings of the 2nd Mile-High Video Conference, MHV 2023, Denver, CO, USA, May 7-10, 2023*, Ali C. Begen, Tamar Shoham, Alex Giladi, Christian Timmerer, and Dan Grois (Eds.). ACM, 39–45. doi:10.1145/3588444.3591002

[41] Junxue Zhang, Chaoliang Zeng, Hong Zhang, Shuihai Hu, and Kai Chen. 2022. LiteFlow: towards high-performance adaptive neural networks for kernel datapath. In *SIGCOMM '22: ACM SIGCOMM 2022 Conference, Amsterdam, The Netherlands, August 22 - 26, 2022*, Fernando Kuipers and Ariel Orda (Eds.). ACM, 414–427. doi:10.1145/3544216.3544229

A Appendix

Figure 14 shows the IPG histogram of traffic before passing through the emulator for the measurement runs using MoonEm SW, NetEm, TLEM, and DEMU with bursty traffic presented in Section 6.2.2. The IPG distribution is nearly identical to the IPG of generated traffic for MoonEm.

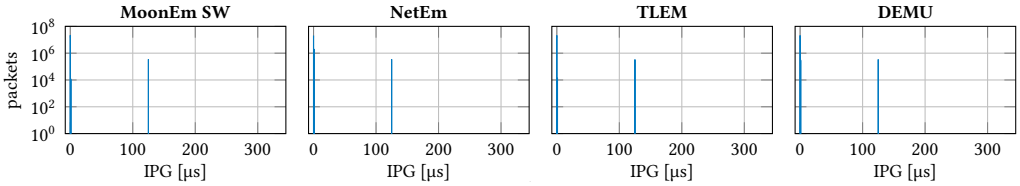


Fig. 14. IPG of generated traffic before passing through the emulator for measurements in Figure 10

B Artifacts

The source code for the updated version of MoonGen/MoonEm is available on GitHub [2]. All scripts and data necessary to recreate the figures in this paper are also published [1], including:

- the measurement scripts to create the data,
- a copy of the raw data used in the figures,
- the processing scripts to evaluate the data, and
- the plotting scripts to create the figures.

Received June 2025; accepted September 2025