Network Replay and Consistency Across Testbeds

Alexander Wolosewicz
Illinois Institute of Technology
Chicago, Illinois, USA
awolosewicz@hawk.illinoistech.edu

Vinod Yegneswaran
SRI
Menlo Park, California, USA
vinod.yegneswaran@sri.com

Ashish Gehani
SRI
Menlo Park, California, USA
ashish.gehani@sri.com

Nik Sultana
Illinois Institute of Technology
Chicago, Illinois, USA
nsultana1@iit.edu

Abstract

Shared network testbeds are critical for systems and networking research. However, their shared hardware can introduce variability—like increased jitter or loss—that may impact experiment fidelity or reproducibility.

We present Choir, the first 100 Gbps replay tool designed to run on commodity hardware and shared infrastructures. Choir enables precise replay and measurement to observe how closely a testbed reproduces expected behavior. We also introduce a metric for quantifying consistency, designed to support comparison across time, configurations, and environments.

We evaluate our approach on FABRIC and a local, bare-metal testbed. We show that FABRIC, even with dedicated resources and low background utilization, has greater variability in inter-packet arrival times and latency compared to the local testbed. With high utilization on shared hardware, this variability increases by an order of magnitude. Our findings demonstrate how tools like Choir can help researchers better understand and mitigate the effects of shared infrastructure.

CCS Concepts

• **Networks → Network performance analysis**; **Network experimentation**.

Keywords

Testbed, instrumentation, Network Traffic, Reproducibility

ACM Reference Format:
Alexander Wolosewicz, Vinod Yegneswaran, Ashish Gehani, and Nik Sultana. 2025. Network Replay and Consistency Across Testbeds. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3731599.3767447

1 Introduction

Network testbeds play a vital role in systems and networking research by providing experimenters with flexible environments that are devoid of the constraints of production networks, such as fixed topologies or rigid protocol stacks. To support broad access and scalability, modern testbeds rely heavily on resource sharing and virtualization [9]. These abstractions are intentionally hidden from users, who interact with the testbed as if they have exclusive access to dedicated hardware. Yet this infrastructure can introduce performance artifacts such as congestion, jitter, and packet loss, especially in a federated environment where QoS guarantees may be limited or inconsistent.

Such inconsistencies undermine reproducibility—a cornerstone of scientific research—by making it difficult to confirm or corroborate experimental results. Non-deterministic failures can be misinterpreted as bugs, complicating the process of building, validating, and debugging experiments. The ability to consistently replay traffic is thus ideal both for scientific reproducibility and for debugging. For generating and replaying network traffic, the current state-of-the-art systems provide accurate generation of Constant Bit Rate (CBR) traffic [6, 14, 19], traffic replayed from packet captures [12, 24], and traffic generated by specified qualities such as a variable rate or TCP connection records [6, 10]. These tools are effective in tightly controlled, dedicated hardware environments and can emulate specific traffic patterns well. However, they fall short in supporting high-fidelity, repeatable traffic replay on shared, virtualized testbeds. This would be useful for debugging or evaluating testbed behavior itself, where an evaluation of the testbed could allow researchers to more accurately describe the environment their results were achieved in, and others could then reproduce these results on similarly-evaluated networks.

First, many replayers and traffic generators capable of excceeding 10 Gbps rely on non-commodity hardware like Tofino-based P4 switches [14]. In contrast, the NICs available on testbeds today can support up to 100 Gbps [8], but achieving high-throughput replay on such hardware remains challenging. Second, commodity-based replay techniques typically assume dedicated hardware and stable conditions; they break down in shared testbed environments where link utilization is variable and full line rate may not be achievable (see Section 9). Third, existing tools are not well-suited for use as debugging tools as they require intrusive setup steps, such as changing the topology or altering the workflow, before replaying traffic. An ideal tool would function in-situ; i.e., it can quickly exist in an invisible stand-by mode and then come online for replaying without requiring a topology rebuild. Such a capability would serve as a foundation for more interactive debugging primitives, such as breakpointing and backtracing.

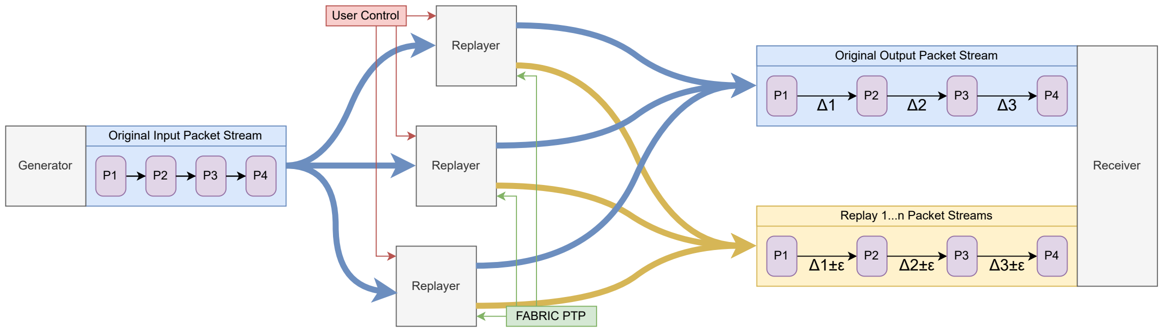


Figure 1: A high-level representation of Choir's goal. An incoming packet stream is divided between three separate replay nodes, and the outputs are later received at a single point in some order. On each replay, this ordering should remain constant, but with some variance in the time deltas. User control commands and FABRIC's PTP synchronization occur out-of-band to the experimental traffic.

Beyond reproducing bugs, researchers need to have an understanding of what impact the back-end infrastructure can have, including experimental bounds and environments for reproducibility. There is no standard metric that characterizes experiment environment consistency; instead, prior works that discuss consistent replay [12, 24] use a collection of low-level metrics such as inter-packet arrival times (IATs), latency variations, and packet drops.

This paper introduces Choir, a Data Plane Development Kit (DPDK) [5] application for producing measurably consistent replays of network traffic for reproducibility and debugging, with low variability in IATs and latency, as illustrated in Figure 1. Using this application, we explore the consistency of the FABRIC testbed, including IATs, jitter, drops, and packet ordering.

This paper makes the following contributions:

- A compound metric κ of packet presence, ordering, IATs, and latency for comparing the consistency of network environments.
- Design and development of Choir, to our knowledge, the first 100 Gbps packet replayer for commodity hardware.
- A novel evaluation of consistency on a federated testbed by comparing a topology on FABRIC to a similar setup on our local testbed.

2 Background

2.1 The FABRIC Testbed

FABRIC [2] is a state-of-the-art network testbed that provides an intercontinental distribution of 33 sites. It allows for flexible, researcher-defined topologies, programmable network resources, and large amounts of compute and storage available throughout the network.

Experiments on FABRIC use the concept of *slices* introduced in PlanetLab [18] and adopted by other previous testbeds like GENI [4]. A slice is a reservation of virtual and physical resources across the federated environment. A slice will contain nodes, representing VMs or hardware, and network services [20], which represent connections between nodes. Users can use an L2 (layer 2) network service, an abstraction that gives the appearance of nodes being directly connected, or an L3 (layer 3) network service, which connects nodes to FABRIC's internal IPv4 or IPv6 network. Many experiments are managed using Jupyter [11] notebooks, which can use FABRIC's Python management API, FABlib [7], to control slice reservation and running commands on nodes.

2.2 PTP on FABRIC

The Precision Time Protocol (PTP) [1] is a widely-deployed tool for providing accurate time synchronization between connected network resources. 23 of FABRIC's sites provide PTP time synchronization for their VMs. With this setup, the physical host has a NIC that receives PTP-timestamped packets from a GPS source, which synchronizes its system clock. Linux VMs can then synchronize with the host's system clock using the kernel's ptp_kvm driver, which the original patch claims has sub-microsecond error [23]. On FABRIC, an Ansible script [17] uses the VM's synchronized system clock to synchronize the NICs.

2.3 DPDK

The Data Plane Development Kit (DPDK) [5] is a highly portable C library for writing user-space applications that can bypass the kernel's network stack and interact with the NIC through Direct Memory Accesses (DMAs). DPDK uses message buffers to store packets alongside metadata. Packets received by the NIC are stored in buffers allocated from shared memory. When threads transmit these packets, they notify the NIC that there are packets to transmit. However, they are not pushed to the wire immediately; instead, the NIC pulls these packets through a DMA at a future time. This delay is not unique to DPDK, but is a restriction for ours and related work.

3 Measure of Consistency in Network Behavior

The word *consistency* in networking can refer to jitter (as in [24]), service uptime and availability, drops and service quality, predictability, or the closeness to constant bit rate (as in [6, 14]). In this paper, the meaning of consistency is precisely defined in terms of determinism: a consistent network is *deterministic*, and therefore running the same trial multiple times produces identical results across the network. We use four metrics to quantify how close to identical two trials are:

- U: Variation in uniqueness (missing/extra packets, which includes drops or corrupted packets)
- O: Variation in packet ordering (reordering)
- L: Variation in latency (jitter)
- I: Variation in inter-arrival time (IAT)

All of these metrics are *variations*, describing differences between two trials. Do the trials have the same packets (U)? The same ordering of those packets (O)? Do those packets arrive at the same time (L) with the same spacing (I)? Any non-zero variation in a metric represents an inconsistency, and all being zero means the trials are identical.

We call a metric *normalized* when it is constrained to a fixed range for any input. We normalize our metrics to the range $[0-1]$, where 0 denotes complete consistency and 1 complete inconsistency, by dividing by the maximum possible value for a given measurement. This will be seen as the denominator of the following formulas.

Let A and B denote two trials, formed as a sequence of packets received by a receiver. We define each normalized metric for the consistency between A and B below.

Equation 1 is the normalized consistency of uniqueness, representing how much overlap there is between A and B. Packets between A and B are the same ($A \cap B$) if they are identical in all regions the evaluator determines define a packet. For later evaluation, we stamped each packet with a unique trailer and used that to define a packet.

$$U_{AB} = 1 - \frac{2 \times |A \cap B|}{|A| + |B|} = U_{BA} \tag{1}$$

As an example, let A be a trial of 10 packets. During trial B, one packet is dropped, and $U = \frac{10+9-2\times 9}{10+9} = \frac{1}{19}$.

Equation 2 is the normalized consistency of ordering, representing how many packets in trial B arrive in the same order as in A. The Longest Common Subsequence (LCS) between A and B is the largest overlapping subsequence of each sequence that appears in the same order in both. Each trial can be considered a permutation of unique packets—where packets are completely identical in data, they can be tagged with their occurrence (so 0 for the first, 1 for the second, and so on) to make them unique. This means the LCS is also the Longest Increasing Subsequence (LIS) of the indices of each unique packet, so the LCS is findable in $O(n \log n)$ time [21]. The value d_i is the distance the ith packet $p_i \in B$ is moved (absolute difference between index of deletion and index of reinsertion) in the minimum edit script that transforms B into A, with the minimum edit script derived alongside the LCS [16]. If $p_i \notin A$ then $d_i = 0$. Since a packet being in B but not in A is an inconsistency already covered by U, we can focus just on inconsistencies in the overlap. The maximum possible value is where B is the reverse of A, and is a constantly increasing length of moves to swap the items around one end.

$$O_{AB} = \frac{\sum_{i=0}^{|B|} d_i}{\sum_{n=0}^{|A\cap B|} n} = O_{BA} \tag{2}$$

Equation 3 is the normalized consistency of latency, where $t_{Xn} \in \mathbb{R}$ is the arrival time of packet n in trial X. For a packet $p_i \in \{A \cap B\}$, with j the index of p_i in A and k the index in B, $l_{Ai} = t_{Aj} - t_{A0}$ and $l_{Bi} = t_{Bk} - t_{B0}$. For example, if a common packet p_n arrives 9 ns

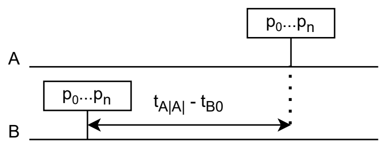


Figure 2: The maximum possible L situation.

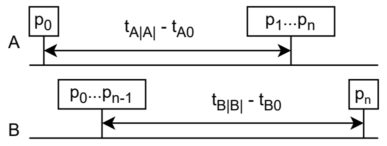


Figure 3: The maximum possible I situation.

after the start of A and 8 ns after the start of B, then $l_{An} = 9$ and $l_{Bn} = 8$. The maximum possible value is where all common packets arrive at one end of A and at the opposite end of B, seen in Figure 2.

The numerator is identical to the "cumulative latency" metric used in the GapReplay paper [24]; our denominator normalizes this metric so it is comparable between trials.

$$L_{AB} = \frac{\sum_{i=0}^{|A\cap B|} \operatorname{abs}(l_{Ai} - l_{Bi})}{|A \cap B| \cdot \max(t_{B|B|} - t_{A0}, t_{A|A|} - t_{B0})} = L_{BA} \tag{3}$$

Equation 4 is the normalized consistency of IAT, where for a packet p_i, using the same definitions of p_i and t_{Xn} from Equation 3, $g_{Ai} = t_{Aj} - t_{A(j-1)}$ and $g_{Bi} = t_{Bk} - t_{B(k-1)}$. For example, if common packet p_n is the fifth packet of A and the fourth packet of B, then $g_{An} = t_{A5} - t_{A4}$ and $g_{Bn} = t_{B4} - t_{B3}$. To cover the base case of a packet being first in A or B, we have that $t_{X0} = t_{X(-1)}$ so that $g_{X0} = 0$. This numerator is identical to the "IAT deviation" metric used in GapReplay, and again our denominator serves to normalize and allow for comparisons between trials.

To find the maximum possible value, consider the instance where the first common packet in A is at t_{A0}, then all others are at $t_{A|A|}$, then in B the last common packet is at $t_{B|B|}$ and all others are at t_{B0}, and the order is consistent with more than 2 common packets (since 2 is trivial, the variation is for a sole IAT), seen in Figure 3.

This gives $L = \operatorname{abs}(t_{A0} - t_{A|A|}) + \operatorname{abs}(t_{B|B|} - t_{B0})$ as the second common packet has the first absolute IAT difference in A and 0 in B (p_1 in Fig 3), and the last common packet has the second absolute IAT difference in B and 0 in A (p_n in Fig 3). To add more IAT difference to another packet would require reducing either of these IAT differences. For example, increasing the IAT between p_0 and p_1 in B in Fig 3 necessitates subtracting the same amount from the IAT between p_{n-1} and p_n, as $t_{B|B|} - t_{B0}$ is the total time of B and the sum of the IATs cannot exceed that. Thus, this is the maximum value we use for normalizing.

$$I_{AB} = \frac{\sum_{i=0}^{|A\cap B|} \operatorname{abs}(g_{Ai} - g_{Bi})}{(t_{B|B|} - t_{B0}) + (t_{A|A|} - t_{A0})} = I_{BA} \tag{4}$$

These metrics can be normalized and combined into a 4-dimensional vector $\boldsymbol{v} = \langle U, O, L, I \rangle \in \mathbb{R}^4$ that measures the inconsistency between different trials. This vector's magnitude $|\boldsymbol{v}| = \sqrt{U^2 + O^2 + L^2 + I^2}$ forms a composite consistency score that lies between 0 (complete consistency) and $\sqrt{1+1+1+1} = 2$ (complete inconsistency). We scale this into a $[0-1]$ ranged score κ between two trials A and B, with 1 as complete consistency, as:

$$\kappa_{AB} = 1 - \frac{\sqrt{U_{AB}^2 + O_{AB}^2 + L_{AB}^2 + I_{AB}^2}}{2} = \kappa_{BA} \tag{5}$$

Sections 6 and 7 will provide examples of using this metric alongside the raw measurements, to show how those measurements are reflected in the compound score.

4 Design of Choir

FABRIC and similar testbeds are by their nature highly flexible. We take advantage of this flexibility to allow for introducing extra resources for observing in ways that might not be practical in traditional networks. The core of Choir is introducing transparent middleboxes on links between nodes. These middleboxes are transparent since they forward traffic, unmodified, at line rate. All middleboxes are joined out-of-band for inter-communication and receiving user commands. At the user's instruction, they will begin to record replays.

While recording, the middlebox remains transparent, and it will record with given start-stop times—future work can add recording in a rolling manner. A recording is made by holding forwarded packets in memory after their transmission without making a copy. While expensive in RAM, avoiding disk writes or copy operations allows an accurate recording to be made without slowing the packet forwarding. Besides the packets, which are stored as the burst they were transmitted as, the recording also stores the time of transmission through reading the Time Stamp Counter (TSC), a constantly-increasing counter on the CPU. Given constant TSC frequencies (which for our implementation, FABRIC nodes have), this provides a quick and accurate method of time recording.

Quick timestamping is important for the next step, running replays. In pursuit of accuracy, the replay will transmit packets near to the same relative time as they were recorded, using the recorded TSC values. The user command to run a replay specifies a future time to start the replay. With this future time and the start time of the replay, a TSC delta can be calculated using the CPU frequency. The replay is then run by looping over a TSC read, transmitting each packet burst in the replay when the TSC read is greater than or equal to the burst's stored TSC time plus the delta. This fast iteration over the time check requires a quick time read, and the counter must always increase, which is why TSC reads are used over other time sources. Of course, the time delay between notifying the NIC that packets are to be transmitted and them hitting the wire, described in Section 2.3, bounds the potential accuracy in time of the replays. Section 6 provides an evaluation of these bounds, and shows that the IAT deviation is low (majority within 10 ns). Section 9 details why existing techniques that would allow for highly-accurate replay timings are not used in this environment. In short, they constantly fill the NIC's transmission queue with invalid packets, but on shared NICs often the full device bandwidth is unavailable or saturation would negatively impact other users of the testbed.

5 Implementation

Choir is built very generally with the goal of being highly portable. It is implemented as a 850-line C program using DPDK as the only library. It does not use any hardware-specific features. When running normally, it transmits packets in up to 64-packet bursts. During replays, it sends bursts to the NIC identically to when it originally transmitted them as a transparent middlebox.

Using larger bursts helps to achieve line-rate performance using fewer hardware resources, which is ideal for portability and for using multiple replay nodes in a topology. The compiled artifact is less than 32 MB with static linkage. The primary restriction is RAM, which only controls how large the replay buffer is. For situations in which a shallow buffer is sufficient, or where the traffic is lower bandwidth, this is not impactful; the program can run with a minimum of 1 GB. It normally runs with 3 CPU cores and 3 network interfaces, for the 2 bridged interfaces and the control interface, but can run with just the 2 bridged interfaces if the control signals run in-band, as we do in our evaluations to conserve resources.

6 Evaluation on a University Testbed

To evaluate the consistency of Choir, we first used it locally and explored both the consistency measure described in Section 3 and metrics such as the distributions of IAT deviation and latency. The ideal functionality is shown in Figure 1, where each replay would consist of the same packets, in the same order, with an IAT variance of 0. The test setup consisted of a generator, replayer, and recorder, with traffic flowing from the generator through the replayer to the recorder. On our local testbed, the replayers and generator each used a Mellanox ConnectX-5 NIC while the recorder used one port of an Intel E810 with the other disconnected. All elements were connected through a AS9516-32D Tofino2 switch running a simple ingress to egress port forwarding program. We used Pktgen-DPDK as the generator.

Locally, we run the user commands and PTP synchronization in-band between the generator and the replay nodes to reduce the number of NICs required. The generator connects to a local stratum 1 Network Time Protocol (NTP) [15] server, and its system clock serves as the PTP grandmaster (what all PTP clocks sync towards). For these evaluations, the generator created a 40 Gbps stream of 1,400-byte packets, each feeding one replayer. This 40 Gbps limit arises from the recorder on our local testbed—later tests on FABRIC revealed the replayer could sustain 100 Gbps. For evaluation purposes, the packets were stamped with unique 16-byte tags in the replayer, which included the replay node they were emitted by.

6.1 Single Replayer

For the trials that we evaluated, the replays consisted of 1,055,648 packets captured from 0.3 seconds of the generator stream, for a rate of 3,518,826 packets per second.

Our local testbed experienced no drops or reordering; put another way, for all trials the values of metric U and O were 0. For 5 trials, we found the variation in IAT was relatively low, but with notable outliers. Between 92.23% (Run D) and 92.51% (Run E) of packets were within 10 ns IAT of the baseline run, visible in Figure 4a. For latency, most packets had a variation between 500 ns and 5 µs, shown in Figure 4b.

In terms of the metrics described in Section 3, U (Uniqueness) and O (Ordering) were both 0. Meanwhile, I (IAT) had the values 0.0290, 0.0309, 0.0308, and 0.0268 and L (Latency) had the values 2.62×10^{-6}, 2.51×10^{-6}, 2.92×10^{-6}, and 9.04×10^{-6} respectively for runs B, C, D, and E. In terms of the combined score, this gives κ values of 0.9855, 0.9845, 0.9846, and 0.9866. We discuss in 8.2 why these metrics are so close to 1.

6.2 Parallel Replayers

To explore Choir and our metrics in a parallel case like in Figure 1, we set up a test topology where the generator sent traffic out of one port each to two replayers, which forwarded to one port of the recorder (still with the switch connecting everything). In this case, the total traffic was still 40 Gbps (3.52 Mpps), 20 Gbps to each replayer. These four runs had IAT distributions shown in Figure 5, which has a similar shape to the distribution in Figure 4a but with noticeably longer tails. In metrics, the percentage of packets with IAT deltas within 10 ns was between 92.75% (Run C) and 92.90% (Run B), and there were I values of 0.311, 0.172, 0.177, and 0.149. Even though the distribution was similar, the far outliers have resulted in metric values nearly an order of magnitude greater than the single-replayer runs. With latency, the L values were 0.0051, 0.0101, 0.0113, and 0.0122, a few order of magnitudes greater than the single-replayer runs as they have nearly half of all packets with latency variations of around -10×10^7 ns. Since our PTP setup synchronizes to within 10s of nanoseconds, there is substantial reordering, and the latency outliers follow this reordering.

To quantify the reordering, there are few other metrics available besides O we can use for comparison. There are 525,824 packets in each run's edit script (which transforms that run's packet capture into the run A's packet capture). This is 49.8% of the captured packets. Of these packets, the distances are characterized by Table 1; however, this neglects to show that most packets that move are moved a similar distance, and that is strongly correlated to burst and replayer. Due to the synchronization, some bursts from a replayer arrive relatively earlier or later, and then some subsequent bursts from the other replayer count as also arriving out of place—meanwhile, the string of packets at the start and end of the trace which are effectively single-replayer form the LCS. Using O we have values of 0.0137, 0.0270, 0.0301, and 0.0326, and the overall κ values were 0.9290, 0.9275, 0.9276, and 0.9287. These are, understandably, worse than the single-replayer tests by a measurable amount as this synchronization issue causes a substantial leap in O and L while I is also transformed, but to a lesser degree.

Run	Mean (σ)	Abs. Mean (σ)	Min	Max
B	1790.54 (8111.16)	7240.23 (4071.35)	-5632	16573
C	3487.95 (16011.25)	14277.30 (8042.66)	-11072	32925
D	3873.69 (17843.43)	15908.56 (8961.64)	-12352	36735
E	4179.75 (19305.66)	17209.84 (9695.35)	-13378	39809

Table 1: Distances packets were moved in the edit scripts transforming each run to run A in Section 6.2.

7 Evaluation on FABRIC

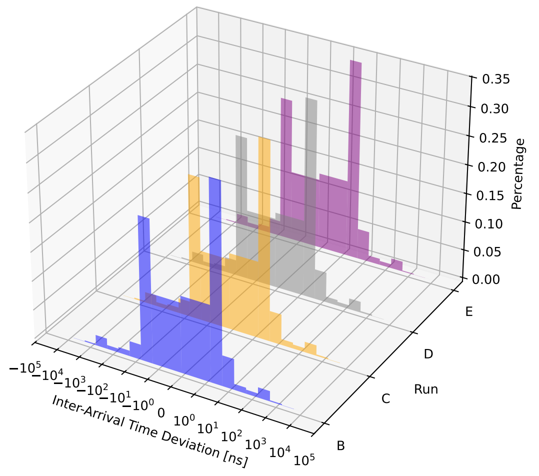
We apply the same techniques used on our local testbed in Section 6 to explore consistency in various environments on FABRIC, mixing shared or dedicated hardware, higher or lower traffic throughput, and the presence or absence of background noise. We ran these tests in a large yet barely used site, which only had allocated 2% of available CPU, 1.1% of RAM and 0.8% of disk space. For the tests, an L2Bridge network service was used to connect the nodes. This is one of the L2 network abstractions described in Section 2.1 which can connect multiple resources within a site. For the first test, the replayer-recorder connection was between two dedicated Mellanox ConnectX-6 smart NICs, while for the second test the connection was between two shared NICs.

In the first test, as in the single-path local case, the replays consisted of 0.3 seconds of a 40 Gbps stream of 1,400-byte packets, totaling 1,052,268 packets (3.51 Mpps). There was no reordering or drops, so U and O are 0. With the IATs, only 30.64% (Run B) to 48.44% (Run D) of packets had a delta within 10 ns, clearly showing higher variation than the local test. Figure 6a shows a greater variance in IATs when compared to the local runs in Figure 4a under similar conditions. This produces I values of 0.5138, 0.5019, 0.4942, and 0.4886, over 10 times greater than the values in the local test. With latency variation, a similar clustering is seen in Figure 6b, but most fall around an order of magnitude higher than in the local tests. This produces L values of 2.13×10^{-5}, 2.78×10^{-5}, 2.54×10^{-5}, and 4.82×10^{-5}, and overall κ scores of 0.6516, 0.8232, 0.8175, and 0.6782.

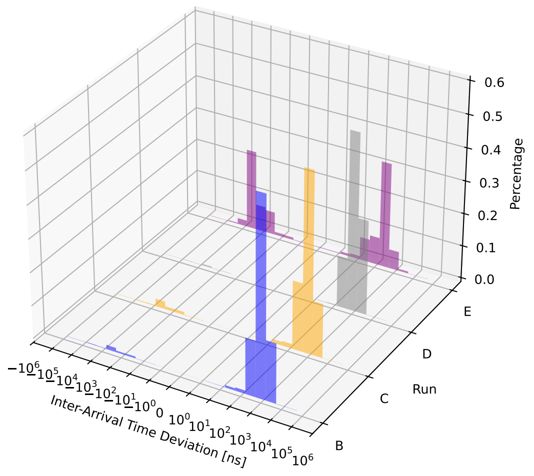
For the second test, the replays were the same overall statistics but consisted of 1,054,046 packets. Again, there were no reordering or drops, so both U and O are 0. In this run, fewer packets had small IAT deltas, ranging from 26.44% (Run D) to 29.15% (Run C) within 10 ns, but most were still closer to 0 than in the previous test. The I values were 0.0698, 0.0597, 0.0667, and 0.0686, just over double the values in the single-stream local test. The latency deltas were again significantly clustered, to a similar order of magnitude of the previous test; they produce similar L values of 1.51×10^{-5}, 1.10×10^{-5}, 3.97×10^{-5}, and 2.40×10^{-5}. The overall κ values are 0.9651, 0.9701, 0.9666, and 0.9657. The IAT and latency distributions are shown in Figure 6.

The second test having better results than the first is a surprising result—one would expect the dedicated hardware to perform more consistently than the virtualized hardware. To confirm this, a third test was run that reused the dedicated hardware, with the same replays. This time, the runs were a more even level of consistency, but this was similar on average to the first test case. The IATs ranged from 24.01% (Run C) to 27.18% (Run D) within 10 ns deltas, like the shared NIC test, but still had noticable outlier values that pulled the I values to 0.514, 0.502, 0.494, and 0.489. In this run, the latency variations were also much worse, with the spikes at over 10^4 ns. The L figures were much higher: 4.49×10^{-4}, 4.55×10^{-4}, 3.97×10^{-4}, and 3.78×10^{-4}. The overall κ values were 0.7431, 0.7490, 0.7529, and 0.7557.

To see if there was any impact from increasing the traffic rate, we ran the tests again on both the dedicated and shared NICs at 80 Gbps.



(a) The percentage of packets with a given IAT delta.



(b) The percentage of packets with a given latency delta.

Figure 4: Histograms showing the variations in IAT and latency for four runs (B, C, D, E) on the local testbed, when compared to the first run (A).

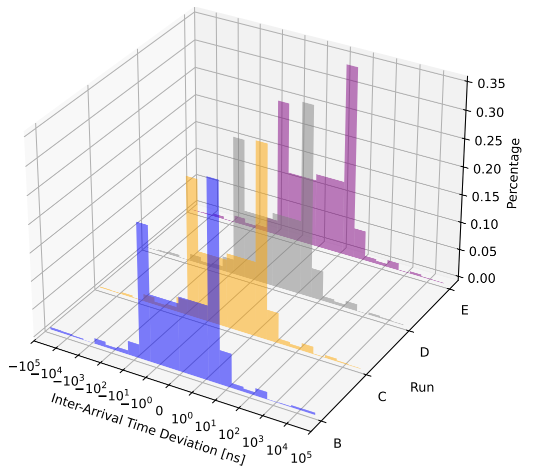


Figure 5: The variations in IAT for four runs (B, C, D, E) compared to the first run (A) on the local testbed, with two parallel replayers.

The software did sustain 100 Gbps, but occasionally the rate over the path would drop so we chose 80 Gbps to further investigate these timing inconsistencies. For these tests, we omit the latency graph since the pattern is well-established: either one spike far to one side or two spikes symmetrically across 0. The increased variation in IAT compared to the local testbed is interesting, and we explore that further.

At 80 Gbps (6.97 Mpps), the IATs get a little more consistent. With the dedicated NICs (IAT distribution in Figure 9a), we see the following:

- B: 30.12% IAT ±10 ns, I 0.108, L 9.89×10^{-6}, κ 0.9460
- C: 30.19% IAT ±10 ns, I 0.106, L 3.83×10^{-6}, κ 0.9469
- D: 30.11% IAT ±10 ns, I 0.106, L 1.04×10^{-5}, κ 0.9468
- E: 30.17% IAT ±10 ns, I 0.109, L 8.69×10^{-6}, κ 0.9456

With the shared NICs (IAT distribution in Figure 9b):

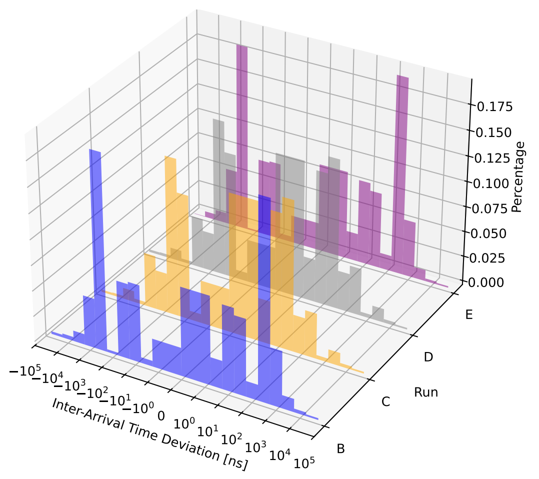
- B: 30.14% IAT ±10 ns, I 0.110, L 2.61×10^{-5}, κ 0.9451
- C: 30.16% IAT ±10 ns, I 0.111, L 2.98×10^{-5}, κ 0.9443
- D: 30.12% IAT ±10 ns, I 0.111, L 1.68×10^{-5}, κ 0.9447
- E: 30.20% IAT ±10 ns, I 0.110, L 1.75×10^{-5}, κ 0.9451

7.1 Evaluation with Noise

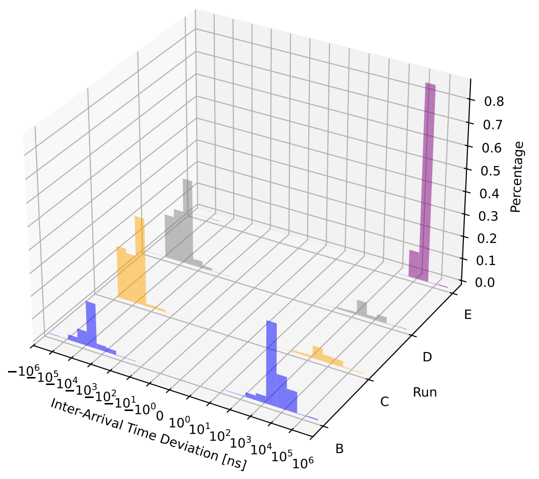
The above tests represent the most ideal situation, with the site virtually unused. To explore any changes that could arise from high loads by other users, we created a second slice on the same site and the same machines, and had one VM (co-located with the replayer) run as an iperf3 client with 8 TCP streams towards a second VM (co-located with the recorder). For this test, the dedicated hardware was again tested at 80 Gbps (6.97 Mpps), since the there was no bandwidth impact from the noise. However, the noise did impact the shared NICs, so we ran that test at the 40 Gbps (3.51 Mpps) rate. This was stable (the iperf3 stream bounced between 35 Gbps and 50 Gbps, mostly around 40 Gbps), so the results showcase minor impacts that can arise while the bandwidth appears steady.

With the dedicated NICs, the result of this test was almost identical to the earlier 80 Gbps test. The IAT graphs especially are virtually identical. In metrics, the packets with IAT deltas within 10 ns ranged from 30.15% to 32.16% (previously 30.12% to 30.19%), I from 0.105 to 0.114 (previously 0.106 to 0.109), and L from 4.91×10^{-6} to 1.78×10^{-5} (previously 1.68×10^{-5} to 2.98×10^{-5}).

Figure 10 shows the histograms for the runs on the shared NIC. In metrics, the IAT deltas within 10 ns ranged from 9.31% (Run D) to 13.81% (Run C). The I values were 0.475, 0.485, 0.530, and 0.521 and the L values were 2.11×10^{-4}, 2.13×10^{-4}, 1.77×10^{-4}, and 2.14×10^{-4}. This is the first trial to have non-zero U values due to occasional drops; only 2 runs (A and C) captured the full replay, the other 3 were respectively missing 1,230, 238, and 205 packets, for U values of 5.84×10^{-4}, 1.13×10^{-4}, and 9.73×10^{-5}. However, as these are relatively few drops given the total replay size was 1,053,824 packets, it has very little impact on the κ values, which were 0.7627, 0.7576, 0.7352, and 0.7397.

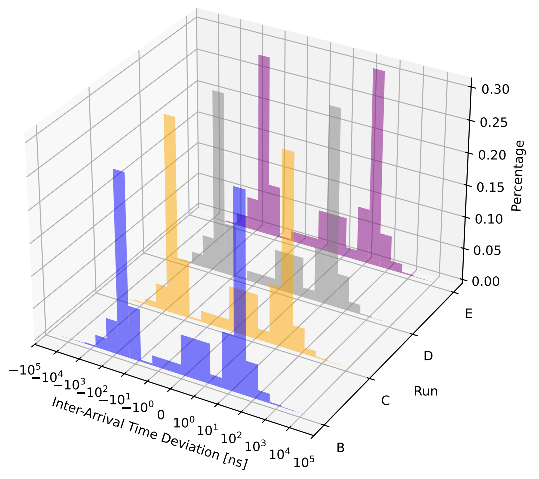


(a) The percentage of packets with a given IAT delta.

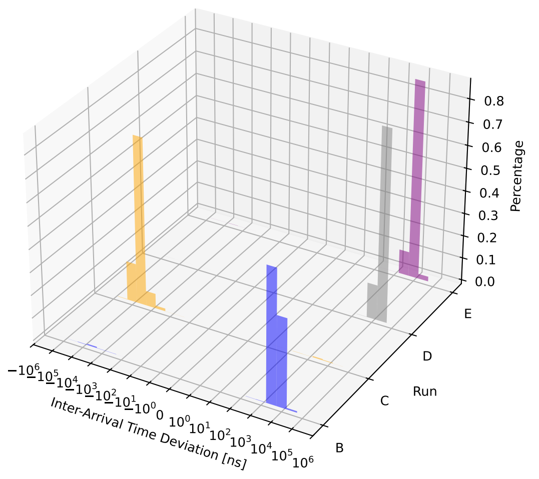


(b) The percentage of packets with a given latency delta.

Figure 6: Histograms showing the variations in IAT and latency for four runs (B, C, D, E), compared to the first run (A), on the FABRIC testbed using dedicated NICs at 40 Gbps (3.51 Mpps).



(a) The percentage of packets with a given IAT delta.



(b) The percentage of packets with a given latency delta.

Figure 7: Histograms showing the variations in IAT and latency for four runs (B, C, D, E), compared to the first run (A), on the FABRIC testbed using shared NICs at 40 Gbps (3.51 Mpps).

8 Findings

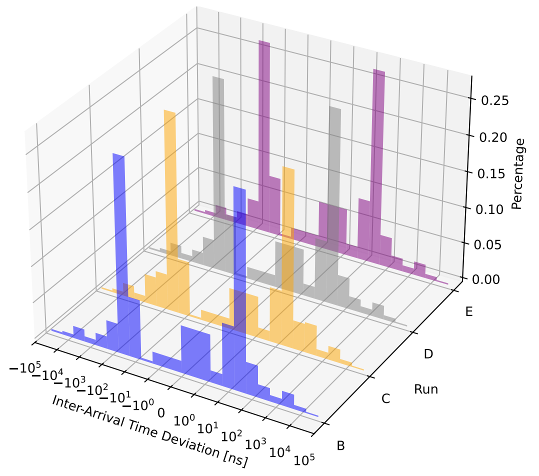
8.1 Local vs FABRIC

Looking at the FABRIC evaluations as a whole, it appears that the first dedicated NIC test was anomalous, and that there are some similarities between the dedicated and shared cases. The latter is expected given that we were running in a best-case environment where there was no resource competition, so the shared NIC could use all the bandwidth of the physical hardware that underlies it. Under load, the consistency noticeably decreases and drops appear.

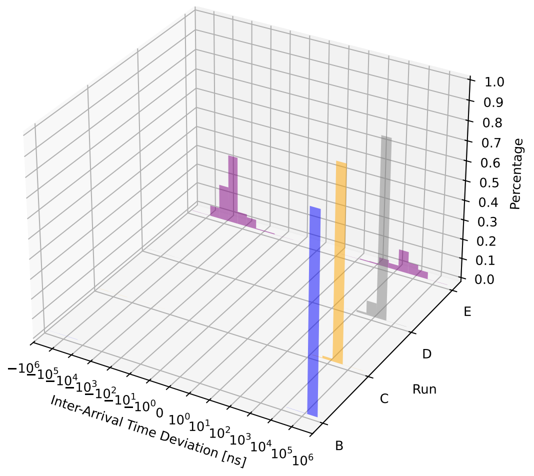
Comparing the similar IAT distributions in Figures 7a, 8a, 9a, and 9b, it is clear that using FABRIC adds extra IAT deviation when compared to the local run in Figure 4a. The known differences in these environments are:

- The local testbed is using 100 Gbps Mellanox ConnectX-5 cards, where FABRIC is using ConnectX-6 cards.
- Our local switch is a AS9516-32D Tofino2, whereas FABRIC sites have Cisco 5700s [8].
- Our applications ran in the host OS on the local testbed, but in virtual environments on FABRIC.
- Our local recorder was an Intel E810 card (which uses real-time HW timestamps), whereas on FABRIC it was a Mellanox ConnectX-6 card (which uses HW clock timestamps converted to ns by sampling the HW clock).

We ran past tests on the local testbed with the generator and recorder swapped, and the consistency was similar, so the last point is unlikely to be producing this difference. We do not have the

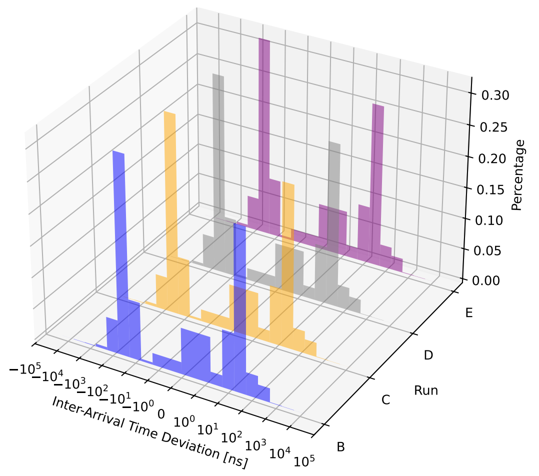


(a) The percentage of packets with a given IAT delta.

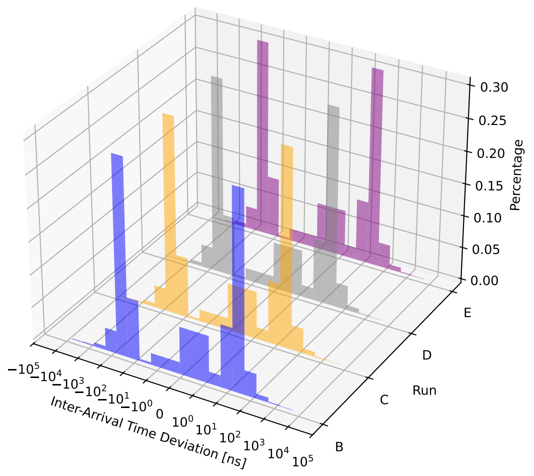


(b) The percentage of packets with a given latency delta.

Figure 8: Histograms showing the variations in IAT and latency for a second set of four runs (B, C, D, E), compared to the first run (A), on the FABRIC testbed using dedicated NICs at 40 Gbps (3.51 Mpps).



(a) The percentage of packets with a given IAT delta using dedicated NICs.



(b) The percentage of packets with a given IAT delta using shared NICs.

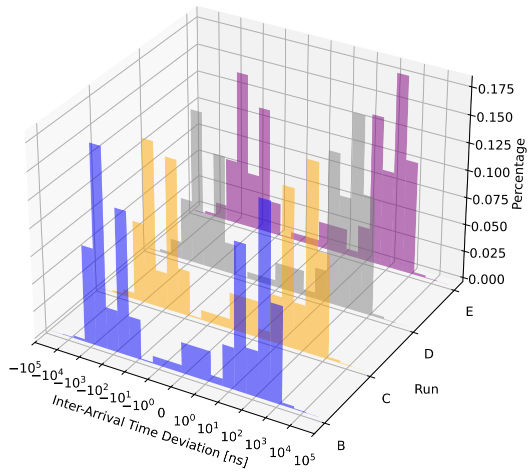
Figure 9: Histograms showing the variations in IAT for four runs (B, C, D, E), compared to the first run (A), on the FABRIC testbed at 80 Gbps (6.97 Mpps)

ability to clearly establish what component could be introducing the extra nanoseconds of variation.

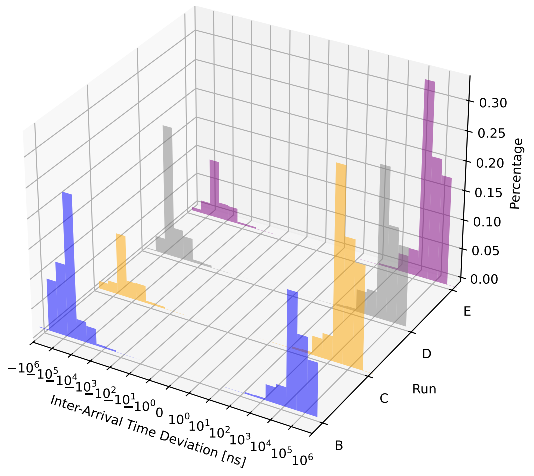
8.2 Consistency Metrics

For our consistency metrics introduced in Section 3, the evaluations have heavily explored the I and L components. We are not the first to use these specific metrics; as pointed out in that section, GapReplay [24] used the same metrics to quantify their latency and IAT. Our contribution here is to provide a proven max value to normalize them, and to combine them into a holistic metric. The mean metrics for each run are shown in Table 2. Looking at the evaluations, we show that the FABRIC instances are still generally consistent (most IATs fall within ±100 ns), but less so than the local environment and so are a few tenths lower in the combined κ score. Converted to percent (which we think fits natural language better), this would mean the FABRIC environments are "4% less consistent" than our environment. We think this is a reasonable conclusion, but it is possible that some weighting or non-linear scaling may be necessary if it is shown that insconsistent environments are close in metrics. Already, the IAT metric seems to somewhat overpower the latency metric in the given environments, since the metrics are linear and L varies within 1×10^{-5} while I varies within 1×10^{-1}.

With the ordering and uniqueness metrics O and U, our formulas are novel and also less evaluated. For the run which encountered packet drops, we see that U is quite low since it represents the percent of non-overlap, with on average 418 out of roughly 1.053 million packets being dropped. However, this value had a relatively minor effect on κ, so future work could explore non-linear scalings



(a) The percentage of packets with a given IAT delta.



(b) The percentage of packets with a given latency delta.

Figure 10: Histograms showing the variations in IAT and latency for four runs (B, C, D, E), compared to the first run (A), on the FABRIC testbed at 40 Gbps (3.51 Mpps) using shared NICs when there was another user sharing the physical hardware.

that would make the presence of any drops more heavily impact the score.

For the run with misordering, there is a noticeable O value and impact on κ. However, the metric hides some patterns we saw. Most packets which were moved, were moved as whole bursts and so have identical distances—put another way, traffic individually from each replayer was ordered, but due to time synchronization in each run the bursts from each replayer would be slightly offset. Some might argue that bursts being out of order should have less impact than if the bursts are internally disordered; however, calculating the edit distance where substrings can be moved as a whole has been shown to be NP-Complete [22], so we think it is better to take our metric with a proper understanding than to try and derive something more complex (and much more computationally expensive given large packet capture sizes). Like U, a non-linear scaling may also be desired so that the presence of any reordering makes a stronger impact, but deriving that factor is left to future work.

9 Related Work

MoonGen [6] is a packet generator that can be scripted to generate complex traffic patterns with a highly consistent bit-rate, demonstrated up to 10 Gbps (1M pps). To avoid the transmit delay issue described in Section 2.3, MoonGen uses invalid packets to control inter-packet gaps. This produces highly accurate gaps, on the order of nanoseconds with a minimum gap of 60 ns. However, this relies on the software being able to produce traffic at line-rate for the NIC and that the other devices can discard the invalid packets. On FABRIC, most available NICs are 100 Gbps SR-IOV Virtual Functions shared NIC, and guaranteeing line-rate is difficult given an inability to control for the load placed on the physical NIC by other researchers. Future work could integrate Moongen and our system for cases with dedicated hardware.

Other research in this area has produced tools that are focused on specific hardware or protocols. GapReplay [24] provides for replaying packet captures at high accuracy using a similar invalid packet technique to MoonGen, but extended with P4 programmable hardware to allow for more precise control of inter-packet arrival times. P4TG [14] utilizes the traffic generators in Intel Tofino switches alongside a custom P4 program. It is able to produce traffic at high speeds (100 Gbps up to 1 Tbps using multicast replication) and at high accuracy, but requires a Tofino switch. With protocols, TCP-Opera [10] and DETER [13] use captured packets and statistics to replay TCP connections. These are useful tools for evaluating larger TCP traces on wide or cross-internet topologies; however, TCPOpera does not replay the specific packets and DETER was demonstrated at 10 Gbps with a larger (5 µs) packet gap. Both are limited to TCP traffic. As a tool for a research testbed, we expect edge cases and exotic setups to be important to support. Choir thus has no reliance on specific hardware or protocols, which separates it from the above.

When it comes to metrics, work by Bellardo and Savage [3] explored measuring reordering in TCP streams and developed a metric showing reordering (as a probability) as a function of inter-packet spacing. This provides insight into what reordering is present, and their methods work on any TCP-supporting system. Our metrics capture the distance of reordering, and could also be shown as a function of spacing. If the presence of reordering is of significant concern, U could scale non-linearly (like using squared distance) such that any reordering is significant.

10 Conclusion

With this work, we start a discussion on how we can create a concise metric for comparing the consistency of various network environments. We demonstrate the use of one such metric when comparing simple environments on our local testbed and on the FABRIC federated testbed. Our metric concisely conveys ideas measurable through existing discrete measurements (such as of latency), like that the ideal FABRIC environments are only slightly (decrease of around 0.04 on a 0-1 scale) less consistent while the noisier environments are significantly (0.2365 decrease) less consistent, with

Environment	U	O	I	L	κ
Local Single-Replayer	0	0	0.0294	4.27×10^{-6}	0.9853
Local Dual-Replayer	0	0.0259	0.2022	9.68×10^{-3}	0.9282
FABRIC Dedicated 40 Gbps 1	0	0	0.4996	3.07×10^{-5}	0.7426
FABRIC Shared 40 Gbps	0	0	0.0662	2.24×10^{-5}	0.9669
FABRIC Dedicated 40 Gbps 2	0	0	0.4998	4.20×10^{-4}	0.7502
FABRIC Dedicated 80 Gbps	0	0	0.1073	8.20×10^{-6}	0.9463
FABRIC Shared 80 Gbps	0	0	0.1105	2.26×10^{-5}	0.9448
FABRIC Ded. 80 Gbps Noisy	0	0	0.1085	1.37×10^{-5}	0.9458
FABRIC Shd. 40 Gbps Noisy	1.99×10^{-4}	0	0.5024	2.04×10^{-5}	0.7488

Table 2: The mean values for each Section 3 metric for each run in the evaluations, in the order they were presented.

the breakdown indicating this is largely from worse IAT deviations. We envision future work that explores this metric in more varied environments to produce refinements, such as non-linear scaling for some metrics or weightings for each component.

This paper introduced Choir, an in-situ traffic replayer for the purposes of debugging, which can sustain peak speeds of 100 Gbps (8.9 Mpps), higher than previous work. We show that on a local testbed, in a linear topology, Choir is consistent in terms of having low variations in latency and IAT deviations generally within 10 ns. On the FABRIC testbed, the consistency is somewhat worse, with variations in latency and IAT increasing at least an order of magnitude on both shared and dedicated hardware, indicating there is something that adds several nanoseconds of IAT variation compared to a bare metal environment. We hypothesize this relates to using virtual machines, but leave it to future work to determine the exact cause.

Acknowledgments

We thank the anonymous reviewers for their feedback. We thank Nishanth Shyamkumar at IIT for help with DPDK, Mami Hayashida, Hussamuddin Nasir, and Jim Griffioen at the University of Kentucky for help with using precise timing on FABRIC, and Komal Thareja and Mert Cevik from RENCI for help with FABRIC. This work was supported by the National Science Foundation (NSF) under award 2346499. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of funders.

References

[1] 2020. IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. *IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008)* (2020), 1–499. https://doi.org/10.1109/IEEESTD.2020.9120376

[2] Ilya Baldin, Anita Nikolich, James Griffioen, Indermohan Inder S Monga, Kuang-Ching Wang, Tom Lehman, and Paul Ruth. 2019. FABRIC: A national-scale programmable experimental network infrastructure. *IEEE Internet Computing* 23, 6 (2019), 38–47.

[3] John Bellardo and Stefan Savage. 2002. Measuring packet reordering. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment* (Marseille, France) *(IMW '02)*. Association for Computing Machinery, New York, NY, USA, 97–105. https://doi.org/10.1145/637201.637216

[4] Mark Berman, Jeffrey S. Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. 2014. GENI: A federated testbed for innovative network experiments. *Computer Networks* 61 (2014), 5–23. https://doi.org/10.1016/j.bjp.2013.12.037 Special issue on Future Internet Testbeds – Part I.

[5] DPDK 2025. Data Plane Development Kit. https://www.dpdk.org/.

[6] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. MoonGen: A Scriptable High-Speed Packet Generator. In *Internet Measurement Conference 2015 (IMC'15)*. Tokyo, Japan.

[7] Fablib [n. d.]. Experiment Management APIs (FABLib). https://learn.fabric-testbed.net/article-categories/fablib-api/.

[8] FABRIC Site Hardware 2023. FABRIC Site Hardware Configurations. https://learn.fabric-testbed.net/knowledge-base/fabric-site-hardware-configurations/.

[9] Mike Hibler, Robert Ricci, Leigh Stoller, Jonathon Duerig, Shashi Guruprasad, Tim Stack, Kirk Webb, and Jay Lepreau. 2008. Large-scale virtualization in the Emulab network testbed. In *USENIX 2008 Annual Technical Conference* (Boston, Massachusetts) *(ATC'08)*. USENIX Association, USA, 113–128.

[10] Seung-Sun Hong and S. Felix Wu. 2006. On Interactive Internet Traffic Replay. In *Recent Advances in Intrusion Detection*, Alfonso Valdes and Diego Zamboni (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 247–264.

[11] Jupyter [n. d.]. Project Jupyter. https://jupyter.org/.

[12] Fred Klassen. [n. d.]. Tcpreplay - Pcap editing and replaying utilities. https://tcpreplay.appneta.com/.

[13] Yuliang Li, Rui Miao, Mohammad Alizadeh, and Minlan Yu. 2019. DETER: Deterministic TCP Replay for Performance Diagnosis. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 437–452. https://www.usenix.org/conference/nsdi19/presentation/li-yuliang

[14] Steffen Lindner, Marco Häberle, and Michael Menth. 2023. P4TG: 1 Tb/s Traffic Generation for Ethernet/IP networks. *IEEE Access* 11 (2023), 17525–17535.

[15] Jim Martin, Jack Burbank, William Kasch, and Professor David L. Mills. 2010. Network Time Protocol Version 4: Protocol and Algorithms Specification. RFC 5905. https://doi.org/10.17487/RFC5905

[16] Eugene W. Myers. 1986. An O(ND) difference algorithm and its variations. *Algorithmica* 1, 1–4 (Nov. 1986), 251–266. https://doi.org/10.1007/BF01840446

[17] Hussamuddin Nasir and Pinyi Shi. 2023. FABRIC PTP. https://github.com/fabric-testbed/ptp?tab=readme-ov-file.

[18] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. 2003. A blueprint for introducing disruptive technology into the Internet. *SIGCOMM Comput. Commun. Rev.* 33, 1 (Jan. 2003), 59–64. https://doi.org/10.1145/774763.774772

[19] Pktgen-DPDK 2024. Pktgen-DPDK: DPDK based packet generator. https://github.com/pktgen/Pktgen-DPDK.

[20] Paul Ruth, Ilya Baldin, Komal Thareja, Tom Lehman, Xi Yang, and Ezra Kissel. 2022. FABRIC Network Service Model. In *2022 IFIP Networking Conference (IFIP Networking)*. 1–6. https://doi.org/10.23919/IFIPNetworking55013.2022.9829810

[21] C. Schensted. 1961. Longest Increasing and Decreasing Subsequences. *Canadian Journal of Mathematics* 13 (1961), 179–191. https://doi.org/10.4153/CJM-1961-015-3

[22] Dana Shapira and James A. Storer. 2007. Edit distance with move operations. *Journal of Discrete Algorithms* 5, 2 (2007), 380–392. https://doi.org/10.1016/j.jda.2005.01.010 2004 Symposium on String Processing and Information Retrieval.

[23] Marcelo Tosatti, Paolo Bonzini, Radim Krcmar, Richard Cochran, and Miroslav Lichvar. 2017. PTP: add kvm PTP driver. https://lore.kernel.org/lkml/20170120122503.842086637@redhat.com/.

[24] Shuanghong Yu, Fenghua Li, Han Zhang, Kunling He, and Xingkun Yao. 2023. GapReplay: A High-Accuracy Packet Replayer. In *ICC 2023 - IEEE International Conference on Communications*. 1616–1621. https://doi.org/10.1109/ICC45041.2023.10279258

A Artifact Description

Our work introduces metrics for quantifying the consistency of network environments and Choir, a system for producing consistent replays of packet traces. Within this article we ran evaluations on both a local testbed and on the FABRIC national testbed. We have produced an artifact, available through FABRIC's artifact manager[1], which is a Jupyter notebook that will:

- Setup our test topology on FABRIC, and install DPDK
- Compile and run Choir
- Analyze packet captures and produce figures similar to those in the paper

This artifact enables others to reproduce our work by simply following the Jupyter notebook. The exact versioning is handled inside - it installs the used Ubuntu (22.04 for Choir, 24.04 for other hosts), DPDK (24.11.1), Pktgen-DPDK (24.10.3), as well as ships the version of Choir [2] and dpdkcap [3] used.

B Artifact Evaluation

Running through the notebook will:

- Create a FABRIC topology with three VMs, using two dedicated smart NICs
- Install needed dependencies and tools on each VM
- Execute commands that will record and run replays, save packet captures, and analyze those captures to produce figures and metrics like those in the paper

The setup time for the notebook should take no more than 30 minutes, which covers provisioning the slice and running setup scripts. From there, setting up and recording a replay should take less than 5 minutes, and each replay can be run and the packet capture recorded in a minute. Analyzing the packet captures and producing graphs for trials similar to what we evaluated should take no more than 5 minutes each, but the time scales with the length of the packet captures and with any reordering. It will produce figures of IAT and latency deviations formatted identically to those in our evaluation, and the metrics will be given in a text file.

[1]https://artifacts.fabric-testbed.net/artifacts/486cc061-05f3-440d-aa42-d49e9cc57551
[2]https://github.com/awolosewicz/fabrictestbed-extensions/blob/e466079829b2ee010bf8693a925395533e50ea9c/fabrictestbed_extensions/fablib/crease/crease_monitor.c
[3]https://github.com/awolosewicz/dpdkcap/commit/dbcb9e399fb9d60fcdacc726e7a6afe2d89367d8