Recent Linux Improvements that Impact TCP Throughput: Insights from R&E Networks

Marcos Schwarz
Rede Nacional de Ensino e Pesquisa
Brazil
Email: marcos.schwarz@rnp.br

Brian Tierney, Kiran Vasu, Eli Dart
Lawrence Berkeley National Laboratory
Berkeley, CA, USA
Email: {bltierney,kvasu,dart}@es.net

Christian Esteve Rothenberg
Universidade Estadual de Campinas
Brazil
Email: chesteve@unicamp.br

Jeronimo Bezerra, Italo Valcy
Florida International University
Miami, FL, USA
Email: {jbezerra,idasilva}@fiu.edu

***Abstract*—This paper reviews recent enhancements to the Linux kernel that impact network throughput and their potential impact on Data Transfer Node (DTN) performance. In particular, we explore the benefits of MSG_ZEROCOPY and BIG TCP in controlled testbed environments at AmLight and ESnet. We compare performance on three different Linux kernel versions, on Intel vs AMD processors, and over multiple round trip times. We conclude with recommendations for both host benchmarking and production-ready DTN configuration.**

I. Introduction

The Linux kernel, now over three decades old, continues to undergo significant performance enhancements, particularly in the context of high-speed networking. Although saturating a 10G network with a single TCP stream has become routine, the emergence of 100G, 200G, and 400G networks is revealing new limitations in the network stack of operating systems.

This paper provides a comprehensive review of recent Linux kernel improvements with a focus on their impact on Data Transfer Node (DTN [1]) performance, specifically for network speeds of 100 Gbps and higher. In particular, we explore the benefits of MSG_ZEROCOPY [2] and BIG TCP [3], comparing these settings for both Intel and AMD-based systems across Linux 5.X and 6.X kernels. Furthermore, we explore current best practices for DTN tuning, identifying which settings remain relevant as newer kernel versions are adopted. The goal of this paper is to be a practical guide to those who are benchmarking and tuning a DTN for optimal network performance. We do not cover file system performance, which is a much bigger and more complex topic that is even more dependent on what hardware is used.

Our study centers on the advancements introduced between the stock Ubuntu 22.04 kernel (5.15, released in April 2022) and the current Ubuntu 24.04 kernel (6.8, released in April 2024). We demonstrate that many performance gains can only be realized through the careful combination of additional tuning parameters and specific kernel capabilities. We present test results from three Linux kernel versions, emphasizing differences in network throughput performance. A robust test methodology is outlined to ensure consistent and reproducible results, verified through controlled environments on the AmLight and ESnet testbeds.

Notably, our experimental research extends beyond prior studies that do not consider WAN latencies [4][5][6], rely on emulated latencies [7], or operate at link speeds of no more than 25 Gbps [8]. Unless explicitly stated otherwise, all the experimental results in our work are based on 100 Gbps or faster R&E network testbeds in AmLight and the ESnet. Another contribution of this paper is releasing all data collected [1] to foster open science and reproducible research. The findings and practical guidelines offered in this work are intended to assist those involved in benchmarking and tuning DTNs for optimized performance through the provided insights into the most recent kernel advancements and their practical application.

The paper is organized as follows. Section II provides background information and discusses relevant related work. Section III presents the testing tools, methodology, and experimental environment. Section IV analyzes the obtained results. Section V provides conclusions and future work directions.

II. Background and Related Work

A. Recent kernel enhancements

Linux 6.X kernels include many performance improvements covering a wide range of areas that may impact DTN throughput. In particular, we focus on improvements impacting network speeds of 100 Gbps and above. Recent relevant kernel enhancements include:

- Support for 'zerocopy' networking, as described below
- Support for 'BIG TCP', or TCP packets larger than 64 KB, as described below.
- Driver Updates and Optimizations: The Nvidia network drivers (formerly Mellanox) have been updated to better support high-speed networking.

[1] https://github.com/marcosfsch/INDIS-2024

- Enhanced Use of AVX-512 [9] for Networking Tasks (on newer Intel processors): Linux 6.x has started to leverage these instructions for tasks such as checksum calculations and packet processing, offering a significant speedup in these operations, which directly impacts TCP throughput.
- Improved Buffer Management: The kernel's memory management has been tuned for better handling of network buffers, reducing overhead and improving the efficiency of data transfer operations.
- Memory Bandwidth Reduction: Efforts have been made to reduce the memory bandwidth consumed by the networking stack, which helps improve throughput by freeing up resources for data transfer tasks.
- NUMA-Aware Scheduling Enhancements: On NUMA (Non-Uniform Memory Access) systems, the scheduler has been enhanced to make better decisions about placing tasks on CPUs closer to the memory they access frequently. This reduces memory latency, which is crucial for networking tasks that require fast access to memory for packet processing, potentially improving TCP throughput on large, multi-socket servers.

B. MSG_ZEROCOPY

MSG_ZEROCOPY [2] is a feature in Linux that allows for more efficient data transfers between user space and kernel space in network sockets. It enables data to be sent or received without unnecessary copying between these spaces, reducing CPU overhead and improving performance, especially in high-throughput scenarios like networking applications. An alternate form of "*zerocopy*" is the Linux *sendfile* call, whose purpose is to transfer data from a file descriptor (typically a file) to a socket directly, without moving the data to user space. *iperf3* has supported *sendfile* for many years, but the newer MSG_ZEROCOPY method is more general purpose.

C. BIG TCP

TSO/GRO are standard offload techniques that use a super-sized TCP packet of 64KB to reduce CPU cycles and interrupts in the Linux network stack. This super-sized packet will be fragmented to the MTU size, usually 1500B or 9000B by the NIC driver. BIG TCP [3] increases TSO/GRO packet sizes above the standard value up to 512KB, thus providing additional reduction of CPU usage. The initial implementation of BIG TCP was released in Linux 5.19, and supported IPv6 only. IPv4 support was added in the 6.3 kernel. While BIG TCP is disabled by default, it can be configured with *iproute* tool (v6.2 and above). Currently, BIG TCP and zerocopy cannot be used simultaneously without a custom built kernel. Compiling Linux with the kernel configuration MAX_SKB_FRAGS=45 is required since both BIG TCP and MSG_ZEROCOPY use SKB fragments, as described in [10]. This limits the near term viability of BIG TCP in production DTN deployments.

The *iproute* commands to enable BIG TCP for IPv4/IPv6 are:

```
# for IPv4
ip link set dev ethX gso_ipv4_max_size 150000 \
                     gro_ipv4_max_size 150000
# for IPv6
ip link set dev ethX gro_max_size 185000 \
                     gso_max_size 185000
```

IPv6 supports slightly larger values for GRO/GSO. We tested BIG TCP for both IPv4 and IPv6, but found no significant difference, so the results reported in this paper are for IPv4 only.

D. Flow Control

In a network that supports IEEE 802.3x flow control, if the receiving host cannot keep up, it sends 'pause frames' to the network device telling it to slow down a bit and buffer the data. Not all networking hardware supports IEEE 802.3x, so it is important to check this when benchmarking 100 Gbps and above hosts.

DTNs that are designed for ingesting high-speed flows should always be connected to network devices supporting IEEE 802.3x. Otherwise, the NIC will likely drop packets, as shown in Section IV. This is particularly true in a WAN environment, as increases in hop count and path latency create longer packet trains (groups of packets with little to no gaps between them) that arrive at the host back-to-back. But even in a LAN we see improvements by using pacing. When flow control is not available on the network, you can also apply pacing on the send host to limit the sending rate, providing similar results, as demonstrated in our results.

E. Related Work

A number of recent publications have analyzed TCP performance for data transfers of 100G and above, unveiling current bottlenecks and presenting strategies to overcome them. However, the literature is commonly restricted to LAN tests or emulated latencies and lower throughput. For example, Cai et al. [5] present a detailed analysis of the Linux network stack, concluding that for high-performance data transfers, the performance bottleneck is shifting from protocol processing to data copy. The work also points to zerocopy as a future direction to solve part of these performance issues. Hock at. al. [6] analyzes the impact of NUMA/CPU affinity and different strategies for placing the application and interrupt to CPU cores, concluding that, for optimal performance results, the applications should not be pinned to cores that handle interrupts from the NIC. Skiadopoulos at. al. [4] present a high-performance zero-copy NIC prototype for hardware and software by benchmarking Linux default networking stack with and without MSG_ZEROCOPY. Their performance findings are similar to our work, but limited only to LAN tests without considering the impact of long latencies.

Other works, such as [8], use real WANs but at lower transmission rates. Their work presents insightful comparisons of different CCAs fairness over different router buffer sizes and AQM algorithms on a 62ms path using the FABRIC testbed [11][12] at 25 Gbps. Rao at. al. [7] present a study of TCP throughput profiles with and without loss, considering multiple congestion control algorithms, highlighting different dynamics that BBR presents over loss and congestion. Their tests were run on testbed that can emulate up to 376 ms RTT, but limited to 10 Gbps.

To the best of our knowledge, this paper is the first to evaluate the performance of MSG_ZEROCOPY and BIG TCP at 100 Gbps and higher link speeds over multiple real WAN latencies, providing a repeatable methodology validated on two independent environments.

III. Testing Methodology and Environment

We run experiments over two testbeds, one on Amlight, and one on ESnet. We ran tests on both environments to validate the proposed methodology and strengthen our conclusions.

A. Test Consistency and Core Selection

Early in this work, we found it surprisingly hard to get repeatable results. Much of the performance variability was related to the CPU scheduler. By default, *irqbalance* distributes NIC interrupts across all cores, and the scheduler distributes processes across all cores as well. The performance of a single 100G flow can vary from 20 Gbps to 55 Gbps on the same hardware, depending on which cores and which NUMA node get assigned. We configured our test hosts following the standard DTN advice from [13] of disabling *irqbalance* and assigning IRQs and user process cores to the correct NUMA node. For the results presented in this work, we used the following settings on all hosts for all tests:

```
set_irq_affinity_cpulist.sh 0-7 ethN
numactl -C 8-15 iperf3 (client and server)
```

where `set_irq_affinity_cpulist.sh` is a script provided by Nvidia[2] to automate assigning the NIC interrupts to a given list of CPUs. Using this method provided reasonable test consistency without the need to explicitly map every core.

B. Test Tool Selection

We used a modified version of *iperf3* v3.17 with patch 1690 [14], in addition to patches from the authors to allow pacing above 32 Gbps and support affinity for multiple CPUs. Starting with v3.16, *iperf3* supports multi-threaded parallel streams, which are required for the parallel stream tests. Patch 1690 includes two new options, inspired by the *neper* tool from Google [15]. The first is the '--skip-rx-copy' flag, which uses the recv system call 'MSG_TRUNC' option to not actually copy the data to user space, but discard it instead. While this would never be used by a real application, it is useful when you want to test and tune sender performance when the receiver is the bottleneck, which is often the case. The second new option is the '--zerocopy=z' flag, which tells *iperf3* to set the MSG_ZEROCOPY flag in the send system call. This option was introduced in Linux 4.17 kernel and is a more general-purpose alternative to the more traditional 'sendfile' method for zero-copy TCP. *iperf3* also supports JSON output, simplifying the results' parsing. We also modified *iperf3* to allow setting pacing (--fq-rate) to values greater than 32 Gbps. Patch [16] was contributed to the upstream *iperf3* project.

C. Kernel Selection

We chose the following list of kernels for testing:

- Linux 5.15 : default for Ubuntu 22.04
- Linux 6.5.0: Ubuntu supported HWE[3] kernel for v22.04
- Linux 6.8: default kernel for Ubuntu 24.04, available as a beta HWE kernel for v22.04

D. Host Tuning

The website fasterdata.es.net [17][18] was used as a reference for our base tuning. We tested a wide range of settings for *sysctl*, *ethtool*, etc, and found that only the following effectively made a difference, so all tests use these settings unless otherwise indicated:

```
# /etc/sysctl.conf
net.core.rmem_max=2147483647
net.core.wmem_max=2147483647
net.ipv4.tcp_rmem=4096 131072 2147483647
net.ipv4.tcp_wmem=4096 16384 2147483647
net.ipv4.tcp_no_metrics_save=1
net.core.default_qdisc=fq
# needed for MSG_ZEROCOPY
net.core.optmem_max = 1048576
# other tuning
# increase ring buffer size (AMD hosts)
/usr/sbin/ethtool -G eth100 rx 8192 tx 8192
# turn off SMT (aka Hyper Threading)
echo off > /sys/devices/system/cpu/smt/control
# set CPU performance governor
cpupower frequency-set -g performance
```

The ring buffer setting above only seemed to help on AMD hosts, not Intel hosts. *tcp_no_metrics_save* is used to prevent caching of previous CWND values. In addition to the above settings, we used a MTU of 9000 bytes, CUBIC as TCP congestion control algorithm, and disabled *irqbalance*. We use a *qdisc* of 'fq', rather than the default of 'fq_codel', as that is recommended for better pacing in a high throughput environment.

Another important tuning setting was to set 'iommu=pt', which increased single-stream throughput from from 29 Gbps to 32.3 Gbps on the ESnet AMD hosts, and multi-stream throughput from 120 Gbps to 180 Gbps on the ESnet 200G WAN path. See [19] for information on how to configure IOMMU.

Setting iommu=pt (IOMMU passthrough mode) can help network throughput by reducing the overhead associated with IOMMU (Input-Output Memory Management Unit) operations. By setting iommu=pt, devices can access memory directly without translation, which can benefit high-performance networking devices like high-speed NICs. This direct access can lead to lower latency and higher throughput.

[2] https://github.com/Mellanox/mlnx-tools/

[3] Ubuntu Hardware Enablement (HWE) provides the newer kernel support for existing Ubuntu LTS releases

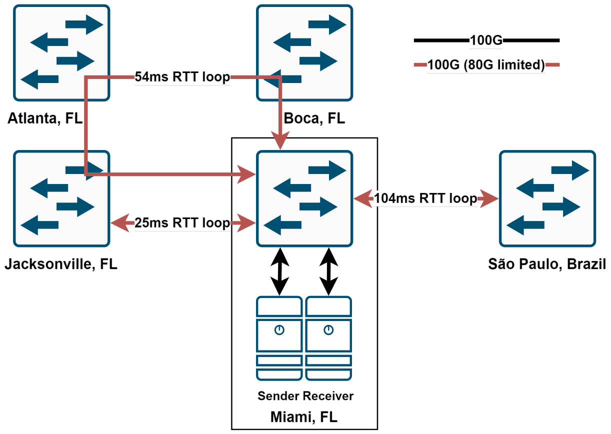

Fig. 1: AmLight testbed resources used in this work.

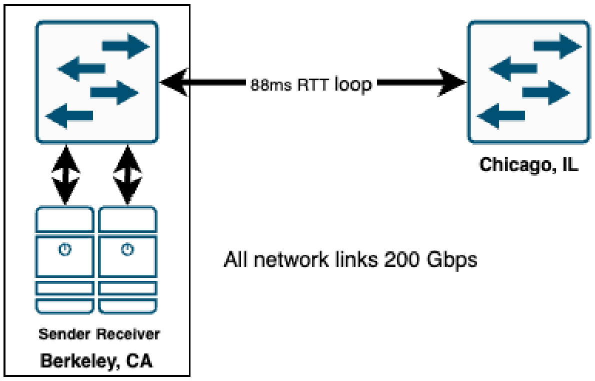

Fig. 2: ESnet testbed resources used in this work.

E. AmLight Testbed

Figure 1 shows the AmLight Testbed resources used in this paper [20]. In this environment, we were able to run LAN tests, as well as on real WAN paths at 25ms, 54ms, and 104ms round trip times. LAN tests were at 100 Gbps, and WAN testing was limited to 80 Gbps to not impact production traffic. Test traffic was not completely isolated from production traffic, which was estimated to be at 16 Gbps on average during our experiments, so it is possible that micro-bursts of packets from production traffic may have some impact on our results, but we believe these impacts to be minimal. The sender and receiver hosts used in AmLight had 32 cores of type 3.1GHz/3.6GHz clock (base/max) Intel Xeon 6346 in a dual-socket configuration, 128GB DDR4 RAM, and the NIC was Nvidia ConnectX-5 (firmware-version: 16.35.3502). All switches are NoviFlow WB-5132D-E based on Edgecore model Wedge 100BF-32X running NoviWare OS.

F. ESnet Testbed

Figure 2 shows the components of the ESnet Testbed used for the experiments described in this paper. All testbed source and destination hosts used in the ESnet testbed had 32 cores of type 3.5GHz/4GHz (base/max) AMD EPYC 73F3 in a dual-socket configuration, and the NIC was Nvidia ConnectX-7. The Edgecore model AS9716-32D switch is used to interconnect all the hosts at 200G, and has a maximum shared buffer size of 64MB. The network switches in both testbeds do not support flow control. Therefore, our results intend to demonstrate that pacing is especially important in environments without flow control.

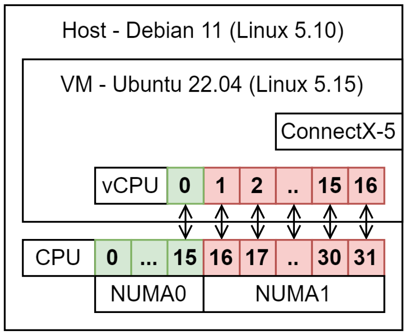

Fig. 3: Virtual testing environment used at AmLight

G. Test Harness Tool

Results were collected using the ESnet 'Network Test Harness' [21], which was described briefly in a previous INDIS paper [22]. All tests were run for 60 seconds, and run a minimum of 10 times. The test harness includes the ability to run *mpstat* along with *iperf3* to monitor CPU usage.

H. Virtual Testing Environment

To allow replicating scenarios between different environments without having to change the host OS/kernel, we have used a virtual machine with additional tuning that minimizes performance overheads. For instance, at AmLight, servers run Debian 11 (kernel 5.10), and ESnet runs Ubuntu 22.04 (kernel 5.15). To make the tests comparable we are running a Ubuntu 22.04 VM at AmLight with 16 vCPUs, 16GB RAM, using PCI-Passthrough of the NIC and tunings to provide close to physical performance. The required tunings are enabling 'iommu=pt' and 'intel_iommu=on' on the host, configure vCPU pinning so that each virtual CPU is fixed on a dedicated physical CPU on the same NUMA as the NIC, as shown in Figure 3

We ran an initial test to validate that the performance of our test methodology inside a virtual machine is comparable to running it directly on the physical server. Figure 4 shows the results for single stream tests using the default settings, and zerocopy plus pacing, using the same Linux distribution (Debian 11) and kernel (5.10). All results present similar performance, and the throughput differences are less than the standard deviation of the tests. We also observe a similar amount of variability between both environments.

IV. Results

A. Single Stream Results

We start by examining single stream throughput. Even though DTNs always use some form of parallel streams,

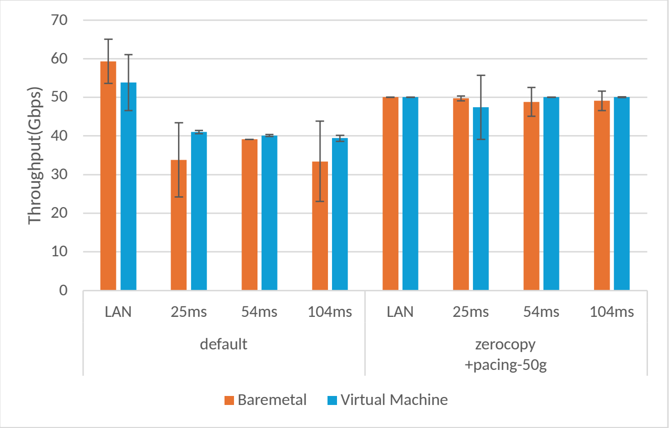

Fig. 4: Baremetal vs VM results on AmLight Testbed (Intel host, single stream, Debian 11, kernel 5.10)

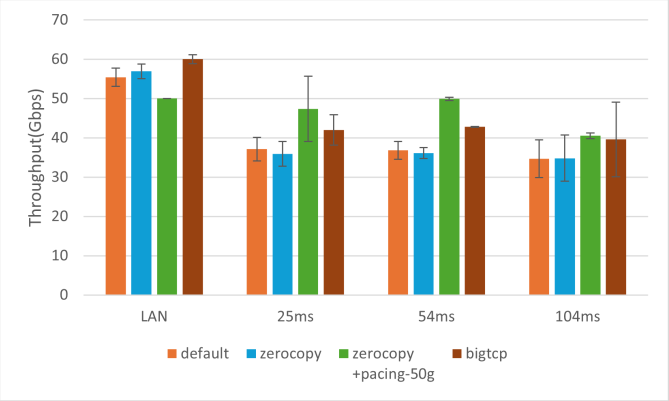

Fig. 5: Single-stream results at AmLight Testbed (Intel host, kernel 6.8)

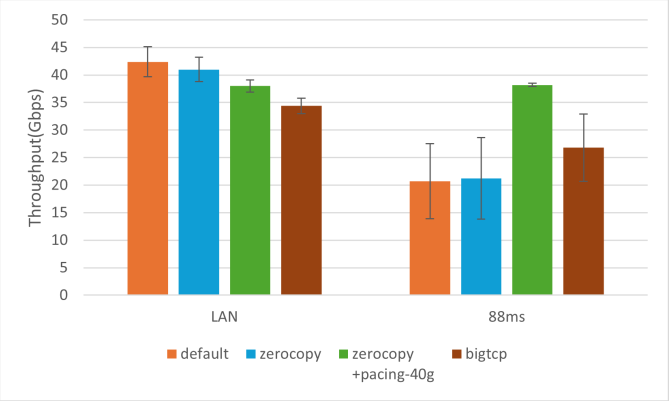

Fig. 6: Single-stream results at ESnet Testbed (AMD host, kernel 6.8)

optimizing single streams is useful for understanding the hardware capacity.

We first discovered in our testing that the AmLight Intel-based hosts were considerably faster for single stream tests than the ESnet AMD-based hosts. On LAN tests using the 6.8 kernel and default tuning, the Intel hosts achieved single stream throughput of 55 Gbps (Fig. 5), while the AMD hosts struggled to do 40 Gbps (Fig. 6). We believe this may be due to the Intel's 6346 series support for '*AVX-512*' instructions [9], which is known to help significantly with network performance. The Intel hosts also have a very different L3 cache architecture, which might contribute to the difference in TCP performance. Therefore, for our single stream testing, we focus mainly on the Intel hosts on the AmLight testbed and note whether and where AMD host performance differs.

We first present results on the AmLight testbed for MSG_ZEROCOPY and BIG TCP. Figure 5 shows the throughput at various latencies for default *iperf3* flags, default plus zerocopy (*--zerocopy=z*), default with both zerocopy and packet pacing (*--fq-rate*) and default with BIG TCP (*gso_ipv4_max_size* and *gro_ipv4_max_size* set to 150 KB). The value for pacing used was manually chosen, considering a value sustained on all WAN RTTs: 50 Gbps for AmLight Testbed and 40 Gbps for ESnet. The kernel used for the results of this plot was 6.8. We observe that MSG_ZEROCOPY by itself does not improve throughput, but combined with pacing (at 50 Gbps), provides up to 35% improvement in all WAN tests and up to 16% faster with BIG TCP. We also ran tests with default settings and BIG TCP plus pacing, but neither showed any performance improvement, so these results were omitted from the plot. The thin line at the top of each result in this plot and subsequent plots indicates the one standard deviation of the values seen in the set of tests and hence indicates the stability of the results.

It is important to note that using MSG_ZEROCOPY on a network path with an RTT over a few milliseconds requires increasing net.core.optmem_max, otherwise it can actually hurt performance and increase sender CPU utilization. This is set using the *sysctl* command. For example: *sysctl -w net.core.optmem_max=1048576*. *optmem_max* is the maximum ancillary buffer size allowed per socket. Ancillary data is a sequence of *struct cmsghdr* structures with appended data. MSG_ZEROCOPY also uses *optmem_max* as a limit for its internal structures. For the results in this section optmem_max of 1MB was used on all tests, on the following section we also study the CPU Utilization and performance impact of different optmem_max sizes.

We also tested MSG_ZEROCOPY and BIG TCP on the ESnet Testbed. Single stream results with AMD hosts are all slower on the ESnet Testbed due to the above-mentioned issues. However, the improvements from using zerocopy combined with packet pacing follow the same pattern, providing 85% improvement on the WAN path and matching the performance of the LAN test, as seen in Fig. 6.

B. CPU Utilization

We explore CPU utilization in more detail. Figure 7 shows single stream sender and receiver CPU utilization for various latencies on AmLight Testbed hosts, for both default *iperf3* settings and for *iperf3* with zerocopy and pacing (at 50 Gbps). We see that with default settings on the LAN, throughput is

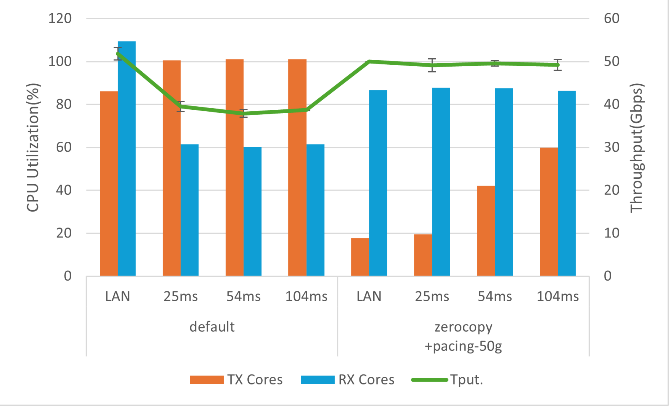

Fig. 7: CPU utilization at various latencies (Single Stream on Intel host)

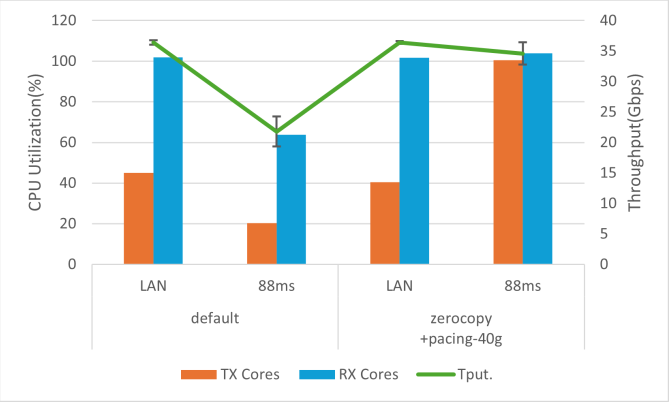

Fig. 8: CPU utilization at various Latencies (Single Stream on AMD host)

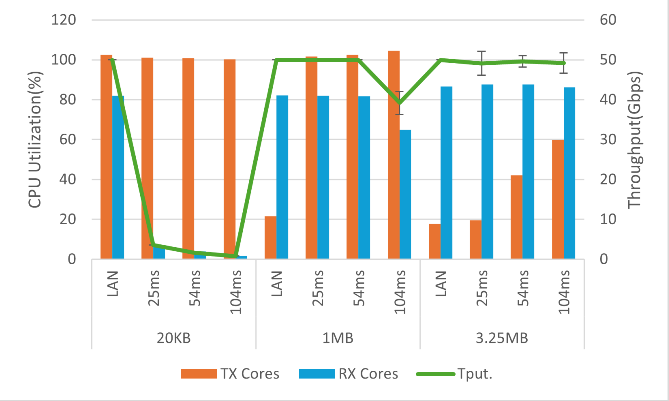

Fig. 9: Sender performance with zerocopy for various *optmem_max* values (Intel host, kernel 6.5)

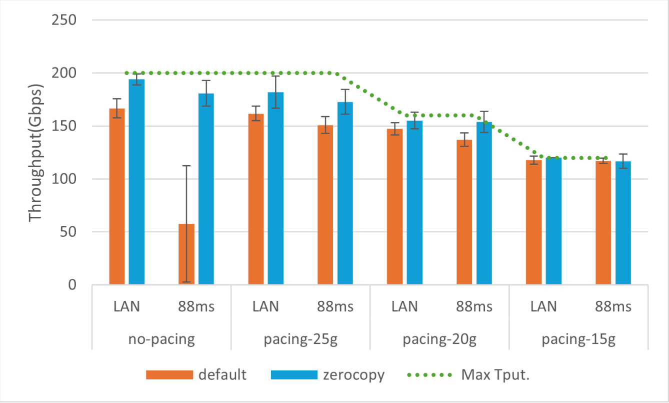

Fig. 10: Multi-Streams results for 8 flows on ESnet Testbed (AMD host, kernel 6.8)

limited by the receiver host CPU, while sender host limited on the WAN. But when using zerocopy with optimal settings for optmem_max and packet pacing, the sender CPU drops dramatically, and the receiver host becomes the bottleneck. Linux 6.5 was used for these results. Also note that with proper tuning, single stream throughput is identical on all paths tested, regardless of RTT. *TX/RX Cores* in the plot aggregate the CPU utilization of the cores used by both *iperf3* and NIC interrupts; therefore this value can go above 100%.

Figure 8 shows single stream sender and receiver CPU for LAN and WAN on ESnet AMD-based hosts, for both default *iperf3* settings, and for *iperf3* with zerocopy and pacing. Note that for the AMD-based hosts, we see similar results but at lower throughput. Without zerocopy and pacing, WAN throughput is about 40% slower than LAN throughput. But with zerocopy and pacing, single-stream WAN throughput is quite similar to LAN throughput. The main difference between these two plots is that the sender CPU on the WAN is much higher on AMD hosts than on Intel hosts.

Figure 9 shows CPU and throughput for various values of *optmem_max* on network paths with various latencies. The first thing to notice is that with the default value of 20KB for *optmem_max* we are completely CPU limited on the sender, and WAN performance is severely affected. After increasing *optmem_max* to 1048576 (which is the value recommended by MSG_ZEROCOPY developers [2]), the pacing rate now introduces a limit, but the sender CPU is the bottleneck on all WAN tests. On our highest latency path (RTT = 104ms), we could only achieve a throughput of 40 Gbps compared with 50 Gbps on the shorter paths. Increasing *optmem_max* even more, we see all WAN streams achieve the same throughput and reduce the sender CPU even further. After testing different values for *optmem_max* we found that 3405376 (approximately 3.25MB) provided the best WAN performance on all our test paths. Values above 3.25MB did not provide any noticeable benefit.

C. Parallel Stream Results

Most R&E production data movement tools such as Globus [23] and Rucio/FTS [24] use parallel streams (or multiple files in parallel), so it is essential to see how parallel streams behave in this environment. We ran a set of tests with 8 parallel streams, with various pacing settings. Previous work has demonstrated the importance of pacing, especially for parallel stream WAN tests [25][26].

TABLE I: ESnet Testbed, LAN results, no Flow Control

Test Config	Ave Tput	Retr	Min	Max	stdev
unpaced	166 Gbps	242	154	177	8.1
25 Gbps / stream	166 Gbps	70	146	172	9.1
20 Gbps / stream	147 Gbps	83	115	153	12.3
15 Gbps / stream	80 Gbps	118	118	119	.1

TABLE II: ESnet Testbed, WAN results, no Flow Control

Test Config	Ave Tput	Retr	Min	Max	stdev
unpaced	127 Gbps	73K	119	137	7.2
25 Gbps / stream	136 Gbps	22K	104	157	15.8
20 Gbps / stream	131 Gbps	8K	118	142	8.9
15 Gbps / stream	115 Gbps	4K	108	119	4.7

We start by looking at the ESnet results, as there was more available bandwidth on the ESnet testbed. Figure 10 shows results for 8 TCP flows on the ESnet testbed paced at various rates. The line labelled *Max Tput* in the plot indicates the maximum possible based on NIC hardware speed or pacing settings. Using MSG_ZEROCOPY with pacing uses less sender CPU, and thus provides a consistent overall throughput of almost the maximum possible (200 Gbps to 120 Gbps, depending on pacing) on both the LAN and WAN tests, providing a huge overall performance improvement. Note that overall the standard deviation is smallest using 15 Gbps for pacing compared to the other pacing settings.

More details on the ESnet tests results, including packet retransmits and minimum/maximum values, are shown in Tables I and II. These results show that pacing is extremely important in environments without IEEE 802.3x flow control. These results are for the 5.15 kernel, and use default *iperf3* settings, other than `--fq-rate` option to set the pacing rate.

We note that while these results show that pacing is not required to achieve reasonable throughput for LAN applications, pacing still helps balance the flows. Without pacing, we commonly see single flow speeds ranging from 5-30 Gbps per flow during the same test run. With proper pacing, all flows get roughly the same throughput. This is shown by the by the high number of packet retransmits and high standard deviation shown in the LAN results table. It was only when pacing all the way down to 15 Gbps/flow did the standard deviation become negligible.

For WAN scenarios, pacing is even more relevant. For all tests with pacing above 15 Gbps/stream, the standard deviation was quite high. For example, we see the total throughput for 8 streams range from 31 Gbps to 120 Gbps for the same test configuration. These results show that high-speed parallel flows are very likely to interfere with each other, leading to a large number of retransmits, especially on the WAN. In the ESnet environment with 200G ConnectX-7 NICs, this occurs any time the total bandwidth attempted (number of streams * pacing rate) is over 120 Gbps.

The results on the AmLight testbed were similar, as shown in Figure 11. All tested scenarios consist of 8 TCP streams paced at both 10 Gbps and 9 Gbps per flow. Default *iperf3* settings is our baseline, which shows throughput decreases with increasing latency, dropping from roughly 60 Gbps to 50 Gbps, due to being CPU limited on the sender.

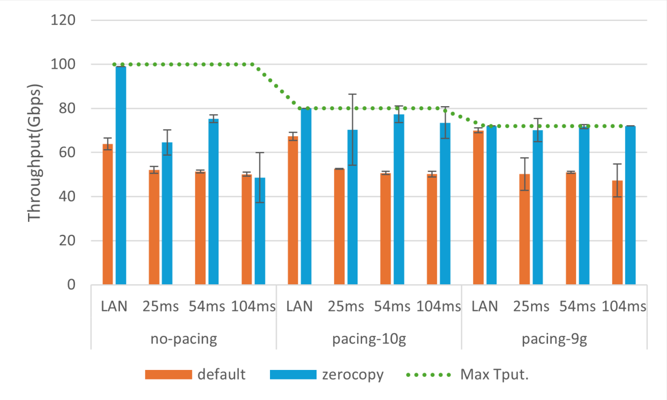

Fig. 11: Multi-Stream results for 8 flows on AmLight Testbed (Intel host, kernel 6.8)

TABLE III: ESnet Production DTNs, with Flow Control

Test Config	Ave Tput	Retr	Range
unpaced	98 Gbps	29K	9-16 Gbps
15 Gbps / stream	98 Gbps	27K	10-13 Gbps
12 Gbps / stream	93 Gbps	8K	11-12 Gbps
10 Gbps / stream	79 Gbps	1K	10-10 Gbps

But different from what was observed on the ESnet Testbed, AmLight results shows that zerocopy without pacing was not able to reach maximum performance. This may be explained by the fact that AmLight WAN paths had around 16 Gbps of production traffic in the background during our tests, so the unpaced zerocopy WAN tests may have suffered congestion, whereas in the ESnet testbed, there was no competing traffic. As on the ESnet testbed, overall the standard deviation is smaller using more pacing, in this case 9 Gbps per stream compared to the 10 Gbps pacing results.

D. Results with Flow Control

Results in an environment with flow control are quite different. Table III shows results between two ESnet production DTNs, RTT = 63ms. Pacing helps to reduce the number of retransmits and make results more consistent, but the average throughput is not impacted. In particular, note the high range of per-flow throughput (9-16 Gbps) without pacing, but when paced to 10 Gbps/stream, all flows where exactly 10 Gbps.

E. Kernel Version Results

We see significant improvement with newer kernels on the ESnet testbed (AMD-based hosts). Figure 12 summarizes the results. From this table, we see that the 6.5 kernel was around 12% faster than 5.15, and the 6.8 kernel was 17% faster than 6.5, for a total improvement of over 30% from 5.15 to 6.8 kernel.

On the AmLight testbed (Intel-based hosts), Figure 13), the improvements are similar, though not quite as dramatic. For

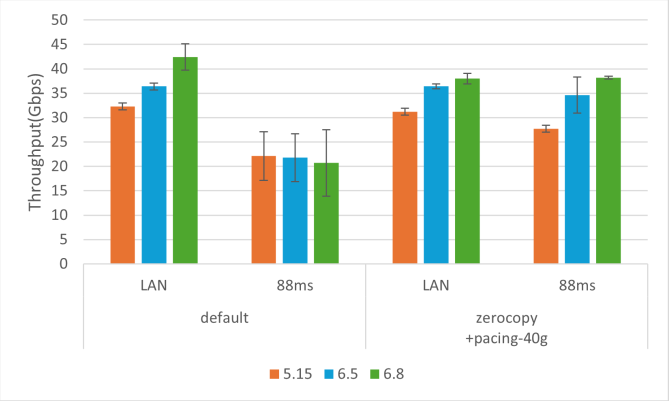

Fig. 12: Kernel version results on ESnet Testbed (AMD host, single stream)

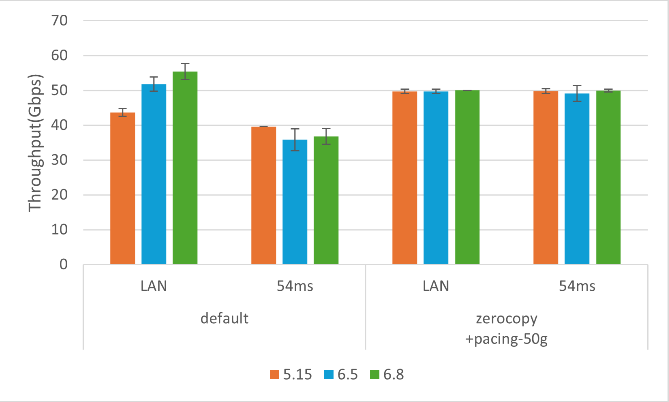

Fig. 13: Kernel version results on AmLight Testbed (Intel host, single stream)

example, single stream LAN tests on the 6.8 kernel were 27% faster than the 5.15 kernel. Single stream WAN performance was the same for all kernels, as they were limited to the 50 Gbps pacing setting required to prevent the receiving host from dropping packets.

Note that with Ubuntu it is quite easy to install the newer (HWE) kernels using *apt*:

```
# to install 6.5 on Ubuntu 22
apt install linux-generic-hwe-22.04
# to install 6.8 on Ubuntu 22
apt install linux-image-generic-hwe-22.04-edge
```

F. Congestion Control Results

Initially we used both CUBIC and BBR (BBRv1 and/or BBRv3, depending on the kernel version) for our testing. Overall, single stream performance was not significantly impacted by the choice of congestion control algorithm, as there is no congestion on our testbeds. As expected, TCP retransmit counts were higher with BBR, especially BBRv1. On the WAN tests, BBRv1/BBRv3 both ramp up faster than CUBIC. We also note that for parallel streams, the BBR flows greatly benefit from proper pacing, otherwise they tend to interfere with each other and back off.

Since the results for BBR were not all that different from CUBIC in our experiments, we do not include congestion control comparisons in this paper.

V. Conclusions and Future Work

Based on our extensive testing of single and parallel stream 100 Gbps flows in two different testbed environments, we derived recommendations separated into two use cases: maximum single flow benchmarking, and DTNs running parallel streams. The single flow benchmarking use case ensures that a 100G+ host is properly tuned for maximum throughput with minimum CPU consumption.

Note that many of the performance enhancements demonstrated in this paper are only realized when additional tuning parameters are used in combination with enabling specific capabilities. For example, MSG_ZEROCOPY does not provide much benefit without increasing optmem_max to appropriate values. It is important to ensure that high performance hosts are configured with the appropriate tuning parameters in addition to just enabling high-performance capabilities.

A. Single Flow Benchmarking

When performing single flow benchmarks, the following are recommended:

- Use tuning settings from https://fasterdata.es.net/host-tuning/linux/100g-tuning/;
- For maximum performance and flow stability use separate CPU cores for IRQ and your test tool;
- Use MSG_ZEROCOPY with increased *optmem_max setting*, along with pacing, for more stable large flows and up to 35% better performance;
- Use kernel 6.8 for up to 38% better performance on WAN and 30% better performance on LAN compared with kernel 5.15;
- Use network devices that support IEEE 802.3x flow control when possible. If not, be sure to use some level of packet pacing.

Note that pacing single flows above 32 Gbps is essential when using MSG_ZEROCOPY at speeds above 32G, but this requires a recent patch to *iperf3* [16].

B. DTN Use Case

When doing a large number of parallel streams, there is less to worry about. The biggest impact is from pacing the streams so that they do not interfere with each other.

For a production DTN it might be tricky to figure out the optimal pacing for your environment. If the 100G DTN is serving data to mainly 10G clients, it might be best to pace to 1 Gbps per flow. If it is mainly sending data to other 100G hosts, 5-8 Gbps/flow might be fine. Note that 'tc' can be used to pace all flows on the host (up to 32 Gbps only), as described at [27].

A heavily used DTN that is running out of CPU serving data to clients would benefit from using MSG_ZEROCOPY. Software that does user-level checksums, such as Globus, may benefit from the extra CPU cycles.

While our testing was memory-to-memory, and not disk-to-disk, we believe these recommendations will still hold in production data transfer environments.

C. Future Work

All our tests were executed on real (not emulated) networks, without loss and congestion. Further testing is necessary to evaluate MSG_ZEROCOPY and BIG TCP on networks with congestion or packet loss. We also intend to evaluate any interactions with congestion control algorithms such as BBRv3 in such an environment.

Another next step is to test the scalability of the parallel stream scenario on 400G gear. Based on our current results, we would expect that 20 flows paced at 20 Gbps would be possible and possibly 10x40G, but additional bottlenecks could be found on the Linux networking stack, and the tuning recommendations may need to be revised.

Further receiver side optimizations will be made available for Nvidia ConnectX-7 network cards on Linux 6.11 [28], which include hardware accelerated GRO and header-data split, they will enable zerocopy on the receiver side and preliminary results from the developer suggests up to 60% throughput improvement for single stream tests.

Another area of further exploration is to evaluate using both BIG TCP and MSG_ZEROCOPY simultaneously on a custom kernel with MAX_SKB_FRAGS=45 [10]. In some preliminary tests we were able to achieve up to 65% improved performance with this configuration. But this setup required modifying the Nvidia ConnectX device driver (mlx5), and the results were not consistent. More BIG TCP testing is needed before we can be confident about its performance improvements, and production kernels need to support BIG TCP in an operationally supportable way before it can be deployed.

VI. Acknowledgments

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research (ASCR), of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

This manuscript has been authored by an author at Lawrence Berkeley National Laboratory under Contract No. DE-AC02-05CH11231 with the U.S. Department of Energy. The U.S. Government retains, and the publisher, by accepting the article for publication, acknowledges, that the U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for U.S. Government purposes.

The AmLight testbed is supported by the National Science Foundation under Grants No OAC-2029283 and OAC-2018754.

In addition, see the content disclaimer, below.[4]

References

[1] E. Dart, L. Rotman, B. Tierney, M. Hester, and J. Zurawski, "The science dmz: A network design pattern for data-intensive science," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2503210.2503245

[2] W. de Bruijn and E. Dumazet, "sendmsg copy avoidance with msg_zerocopy," in *The Technical Conference on Linux networking*, vol. 2, 2017.

[3] E. Dumazet, "tcp: BIG TCP implementation." [Online]. Available: https://lore.kernel.org/netdev/20220513183408.686447-1-eric.dumazet@gmail.com

[4] A. Skiadopoulos, Z. Xie, M. Zhao, Q. Cai, S. Agarwal, J. Adelmann, D. Ahern, C. Contavalli, M. Goldflam, V. Mayatskikh *et al.*, "High-throughput and flexible host networking for accelerated computing," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024, pp. 405–423.

[5] Q. Cai, S. Chaudhary, M. Vuppalapati, J. Hwang, and R. Agarwal, "Understanding host network stack overheads," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 65–77.

[6] M. Hock, M. Veit, F. Neumeister, R. Bless, and M. Zitterbart, "Tcp at 100 gbit/s–tuning, limitations, congestion control," in *2019 IEEE 44th Conference on Local Computer Networks (LCN)*. IEEE, 2019, pp. 1–9.

[7] N. Rao, "Experimental study of tcp throughput profiles and dynamics over dedicated connections," in *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023, pp. 776–784.

[8] I. Mahmud, G. Papadimitriou, C. Wang, M. Kiran, A. Mandal, and E. Deelman, "Elephants sharing the highway: Studying tcp fairness in large transfers over high throughput links," in *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023, pp. 806–818.

[9] Intel, "AVX-512." [Online]. Available: https://en.wikipedia.org/wiki/AVX-512

[10] E. Dumazet, "net: introduce a config option to tweak MAX_SKB_FRAGS." [Online]. Available: https://lore.kernel.org/netdev/20230323162842.1935061-1-eric.dumazet@gmail.com

[11] I. Baldin, A. Nikolich, J. Griffioen, I. I. S. Monga, K.-C. Wang, T. Lehman, and P. Ruth, "Fabric: A national-scale programmable experimental network infrastructure," *IEEE Internet Computing*, vol. 23, no. 6, pp. 38–47, 2019.

[12] "FABRIC Project." [Online]. Available: https://portal.fabric-testbed.net/

[13] "Fasterdata: Interrupt Binding," accessed: 2024-08. [Online]. Available: https://fasterdata.es.net/host-tuning/linux/100g-tuning/interrupt-binding/

[14] D. Bar-On, "iperf3 Pull Request #1690: Add support for SKIP-RX-COPY using MSG_TRUNC and Zero-copy using SO_ZEROCOPY/MSG_ZEROCOPY," 2024. [Online]. Available: https://github.com/esnet/iperf/pull/1690

[15] Google, "neper: a Linux networking performance tool," 2016. [Online]. Available: https://github.com/google/neper

[16] M. Schwarz and B. Tierney, "iperf3 Pull Request #1728: Changed fqrate from uint to uint64 to allow pacing above 32G," 2024. [Online]. Available: https://github.com/esnet/iperf/pull/1728

[17] ESnet, "Linux Host Tuning," accessed: 2024-08. [Online]. Available: https://fasterdata.es.net/host-tuning/linux

[4] This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

[18] "Fasterdata: 40G/100G Network Tuning," accessed: 2024-08. [Online]. Available: https://fasterdata.es.net/host-tuning/linux/100g-tuning

[19] "Fasterdata: IOMMU," accessed: 2024-08. [Online]. Available: https://fasterdata.es.net/host-tuning/linux/100g-tuning/iommu/

[20] AmLight, "AmLight Network." [Online]. Available: https://www.amlight.net/

[21] ESnet, "ESnet Network Test Harness: A harness for executing jobs across multiple hosts and collecting output." 2024. [Online]. Available: https://github.com/esnet/testing-harness

[22] B. Tierney, E. Dart, E. Kissel, and E. Adhikarla, "Exploring the bbrv2 congestion control algorithm for use on data transfer nodes," in *2021 IEEE Workshop on Innovating the Network for Data-Intensive Science (INDIS)*, 2021, pp. 23–33.

[23] I. Foster, "Globus Online: Accelerating and Democratizing Science through Cloud-Based Services," *IEEE Internet Computing*, vol. 15, no. 3, p. 70–73, May 2011. [Online]. Available: https://doi.org/10.1109/MIC.2011.64

[24] CERN, "FTS: File Transfer Service; Open source software for reliable and large-scale data transfers," 2019. [Online]. Available: https://fts.web.cern.ch/fts/

[25] N. Hanford, B. Tierney, and D. Ghosal, "Optimizing data transfer nodes using packet pacing," in *Proceedings of the Second Workshop on Innovating the Network for Data-Intensive Science*, 2015, pp. 1–8.

[26] M. Ghobadi and Y. Ganjali, "Tcp pacing in data center networks," in *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*. IEEE, 2013, pp. 25–32.

[27] "Fasterdata: Packet Pacing," accessed: 2024-08. [Online]. Available: https://fasterdata.es.net/host-tuning/linux/packet-pacing/

[28] T. Toukan, "net/mlx5e: SHAMPO, Enable HW GRO once more," 2024. [Online]. Available: https://lore.kernel.org/all/20240603212219.1037656-1-tariqt@nvidia.com/