SANDIA REPORT
SAND2019-10319

Printed September 2019

Sandia National Laboratories

An Initial Investigation of the Design Challenges Associated with Reliable 100GigE Packet Capture

Haoda Wang, Gavin M. Baker, Joseph P. Kenny, Craig D. Ulmer

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods

ABSTRACT

Network security researchers often rely on Emulytics™ to provide a way to evaluate the safety and security of real world systems. This work involves running a large number of virtual machines on a distributed platform to observe how software and hardware will respond to different types of attacks. While Emulytics™ software such as minimega [2] provide a scalable system for conducting experiments, the sheer volume of network traffic produced in an experiment can easily exceed the rate at which data can be recorded for offline analysis. As such, researchers must perform live analytics, narrow their monitoring scope or accept that they must run an experiment multiple times to capture all the information they require.

In support of Sandia's commitment to Emulytics™, we are developing new storage components for the Carnac cluster that will enable researchers to capture significantly more network traffic from their experiments. This report provides a summary of Haoda Wang's initial investigation of how new AMD Epyc storage nodes can be adapted to perform packet capture at 100Gbps speeds with minimal loss. This work found that the NVMe storage capabilities of the Epyc architecture are suitable for capturing 100Gbps Ethernet traffic. While capturing traffic with existing libraries was surprisingly challenging, we were able to develop a DPDK-based software tool that recorded network traffic to disk with minimal packet loss.

ACKNOWLEDGMENTS

This work was performed during Haoda Wang's internship in the Summer of 2019. We gratefully acknowledge the help and guidance of several coworkers. Jerry Friesen supported this work out of Institutional Computing funds and oversaw the work performed by the interns. Jason Gao answered a number of technical questions about how high-performance Ethernet hardware works, and made several insightful recommendations about packet capture packages. Matthew Troglia provided network traces from his own Emulytics project that were used as workloads in several of the benchmarks. Finally, Sam Knight helped troubleshoot network performance problems.

CONTENTS

1. Introduction 7
   1.1. Carnac Cluster 7
   1.2. Full-Packet Capture Challenges 8
   1.3. AMD Epyc Nodes 9
   1.4. Outline 9

2. Epyc Node Performance 9
   2.1. Processor Benchmarks 9
   2.2. Local Storage Experiments 12
   2.3. ConnectX-5 NIC Performance 14
   2.4. RAM 15

3. Analysis of Commodity Packet Capture Software 15
   3.1. Comparison of Available Packet Capture Frameworks 16
   3.2. DPDK Architecture 17
   3.3. Comparison of DPDK-based Packet Capture Tools 17

4. Packet Capture with libpcap 17
   4.1. libpcap with DPDK 17
   4.2. libibverbs Dependency on libpcap 18
   4.3. libpcap with VMA 18
   4.4. Systemtap Instrumentation 18

5. Packet Capture Tests 19
   5.1. DPDK Setup 19
   5.2. Traffic Generation 20
   5.3. DPDK pdump Packet Capture 22
   5.4. DPDK with libpcap and null Interfaces 25

6. Primary Problem 26

7. Lessons Learned and Next Steps 26
   7.1. Lessons Learned 26
   7.2. Future Ideas 26

References 28

LIST OF FIGURES

Figure 1-1. Sandia's 100GigE Carnac Cluster 8
Figure 2-1. Scimark2 performance results (higher is better) 10
Figure 2-2. LLVM build time results (lower is better) 10
Figure 2-3. C-ray test performance results (lower is better) 11
Figure 2-4. Stockfish chess performance results (higher is better) 11
Figure 2-5. Write latency by filesystem used (lower is better) 12
Figure 2-6. Write speed by filesystem used (higher is better) 13
Figure 2-7. XFS RAID0 latency results (lower is better) 13
Figure 2-8. XFS RAID0 transfer speeds (higher is better) 14
Figure 2-9. Optimizations and their effect on network speed 15
Figure 5-1. Maximum IMIX bandwidth attained by threads assigned 21
Figure 5-2. Mlx5 throughput characteristics in Trex 21
Figure 5-3. Testpmd performance by RX queue cound 22
Figure 5-4. Testpmd performance by frame size 23
Figure 5-5. IMIX capture performances in various configurations 24
Figure 5-6. Packets captured by packet size 25

1. INTRODUCTION

Sandia National Laboratories uses Emulytics™ as a way to evaluate how enterprise networks perform under different operating conditions. In this work a framework such as minimega [2] is used to launch a large number of virtual machines (VMs) on a cluster computer and establish routing between the VMs in a way that emulates a specific environment of interest. The benefit of Emulytics™ over simply simulating the network is that software components behave more faithfully because they are based on the software and firmware found in actual systems. Over the last decade, Sandia researchers have used Emulytics to answer a variety of questions, ranging from *"How will this attacker's software spread?"* to *"Will our enterprise network be able to support a video stream with a thousand consumers?"*.

Data management is an ongoing challenge for the community. Given the sheer volume of network data that an enterprise-class experiment can produce, most researchers either limit their captures to a very small number of connections of interest or focus on distilling data down to real-time statistics. While these approaches are acceptable in many situations, many researchers would benefit from the ability to have full packet capture. Packet captures offer proof that systems performed in a certain manner and allow analysts to sift through the results after an experiment and obtain better insight into what events took place.

1.1. Carnac Cluster

Sandia has a dedicated cluster named Carnac that is used for Emulytics™ projects. Carnac has 288 compute nodes, each with 32 physical cores, 512GB of memory, 2TB of local SSD storage and a single-port Connect-X 5 100GbE network card. Experiments communicate through a 100Gbps Ethernet (100GigE) network that uses an Arista DCS-7512N core switch. A custom bare-metal scheduling service allows users to reserve cluster nodes for an extended period of time. Users can install an operating system of their choosing onto node-local storage or PXE boot the nodes into a diskless kernel/initramfs image.

Figure 1-1 Sandia's 100GigE Carnac Cluster

1.2. Full-Packet Capture Challenges

Full-packet capture is challenging for a number of reasons:

Increased Network Speeds: Platforms such as Sandia's Carnac cluster feature 100Gbps Ethernet for experiments. While most emulations take place at much lower speeds, a packet capture system must take into account high-bandwidth bursts. Very few packet capture libraries have been successfully tested beyond 40Gbps.

Storage Speed Challenges: A full-packet capture system must be able to store packets at a rate that matches network bandwidth. Until recently, it was difficult to create a storage solution in a node that could handle multi-gigabyte-per-second speeds.

Storage Capacity Challenges: Packet capture systems also need a large amount of capacity to store incoming data. Current NVMe cards offer only a fraction of the storage of rotational hard drives. Additionally, a host system has limited expansion bus (PCIe) lanes for hosting multiple NVMe cards in a single system. As such it can be difficult to establish enough storage for capturing long sessions of 100Gbps traffic.

1.3. AMD Epyc Nodes

Given the usefulness of full-packet capture, we explored different alternatives for constructing a computing system that could capture and store 100GigE network traffic for sustained periods of time. AMD's new Epyc architecture provides a compelling system for this work. Epyc nodes offer many cores, a large number of PCIe lanes, and support large amounts of memory.

Sandia procured 16 AMD Epyc nodes to serve as a high-performance storage array for both the Carnac cluster and a neighboring high-performance data analytics (HPDA) system. An individual Epyc node contains the following components:

Table 1-1 AMD Epyc Node Hardware

Component	Description
Processor	Dual-Socket Epyc 7551 (32 cores per socket)
Memory	1TB 2667MHz DDR4
NIC	Mellanox Dual-Port ConnectX-5
Storage	10 x 2TB U.2 NVMe Drives

1.4. Outline

The remainder of this paper will focus on our work preparing these nodes for use in high-speed packet capture. First, we conducted performance benchmarks to help evaluate the hardware and make decisions about how it should be configured. Second, we explored different packet capture packages and evaluated their performance on this hardware. Finally, we summarize some of the lessons we learned in this work and outline next steps in this research.

2. EPYC NODE PERFORMANCE

As a first step in understanding the strengths and weaknesses of the Epyc node hardware, we ran a number of standard benchmarks for CPU, memory and I/O performance. This section provides a summary of observations from these experiments.

2.1. Processor Benchmarks

The new Epyc nodes are well-suited for running heavily-multithreaded programs due to the large amount of physical cores, as well as the extremely large cache sizes. However, they lag behind Intel nodes currently available in our datacenter in single-core processing speed. Scimark2, a single-threaded benchmark for scientific computing, was run using the Phoronix Test Suite on the

various compute architectures available within our datacenter (Carnac and Kahuna Intel-based clusters, Satya GPU machines, and Epyc nodes). The results of a composite benchmark can be seen below. The Epyc node matched the Kahuna nodes' performance in Monte Carlo integration and FFT, while falling behind on Gauss-Seidel relaxation, sparse matrix multiplication and dense LU factorization.

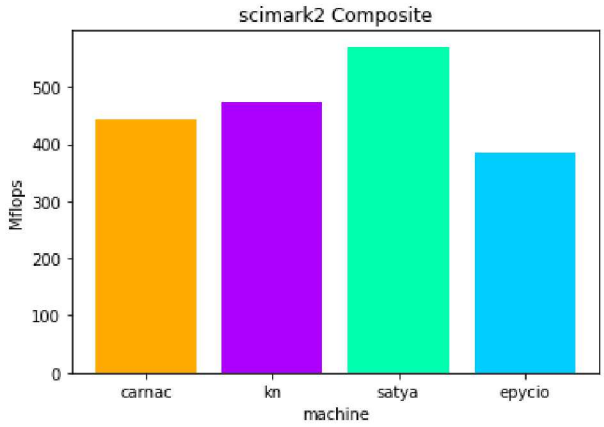

Figure 2-1 Scimark2 performance results (higher is better)

When running benchmarks supporting multithreading, the Epyc nodes show performance results that surpass the other compute nodes. This can be seen in the three benchmark results below, which reflect multithreaded workloads that may be run on a large compute cluster, including code compilation and simulation.

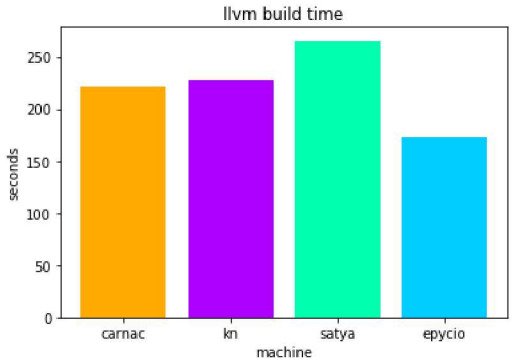

Figure 2-2 LLVM build time results (lower is better)

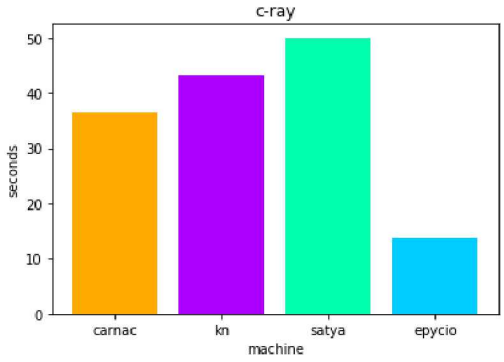

Figure 2-3 C-ray test performance results (lower is better)

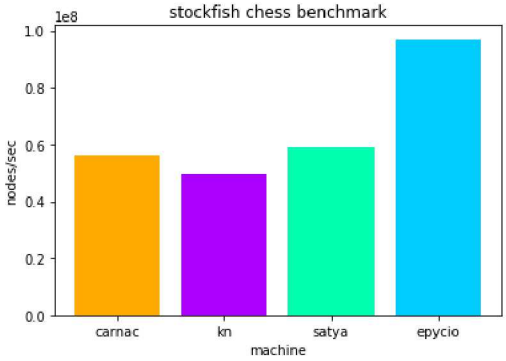

Figure 2-4 Stockfish chess performance results (higher is better)

Users should be aware that the AMD Epyc processors have a different NUMA layout than traditional Intel CPUs. Each physical Intel chip is its own NUMA node, while a single Epyc processor contains 4 distinct dies [1], each with 2 core complexes, which share an L3 cache. Each core complex holds 8 physical cores, with every 2 cores sharing a L2 cache. The complexity of this memory hierarchy necessitates that more care should be taken to ensure that NUMA-sensitive software runs on the correct NUMA regions.

Another interesting architecture choice involves RAM placement. While on traditional systems the maximum memory throughput is attained by populating all DIMM slots, Epyc performs best when the DIMM slots are only ***half populated***[1]. We observed a slight performance improvement in benchmarks when removing half of the memory. However, this improvement was not observed in dpdk_pdump experiments. Given that memory capacity is an important aspect of packet capture, we chose to fully populate the DIMM slots for the remainder of the experiments in this paper.

[1] `https://developer.amd.com/wp-content/resources/56301.pdf`

2.2. Local Storage Experiments

The Epyc nodes are equipped with ten 2TB Samsung 935 DCT NVMe storage devices that use U.2-based PCIExpress to connect to the host. Our performance experiments focused on finding a storage configuration for these devices that would maximize both bandwidth and minimize latency for packet capture. We experimented with different filesystems and different RAID approaches. Testing various configurations of filesystems with fio showed that XFS [9] provided the highest throughput, while Btrfs [7] provided lower latency. ZFS [8] surprisingly provided the lowest write speed, while also incurring the highest latency per write. F2FS [5] is a new filesystem designed for flash devices, but the size was limited to 16TB and write speed did not exceed that of XFS.

SPDK [10], a kernel bypass library providing direct access to a drive's block device, provided a write speed below XFS and F2FS, but also showed the minimum latency, on the order of tens of microseconds, while all the traditional filesystems exceeded 2,000 µs latency. SPDK was benchmarked with an independent benchmark tool shipping with the repo as it was not a filesystem and required compilation. As the perf tool does not cache writes or rely on any optimizations beyond SPDK, one can assume that the results from SPDK most accurately reflect the maximum performance of these SSDs.

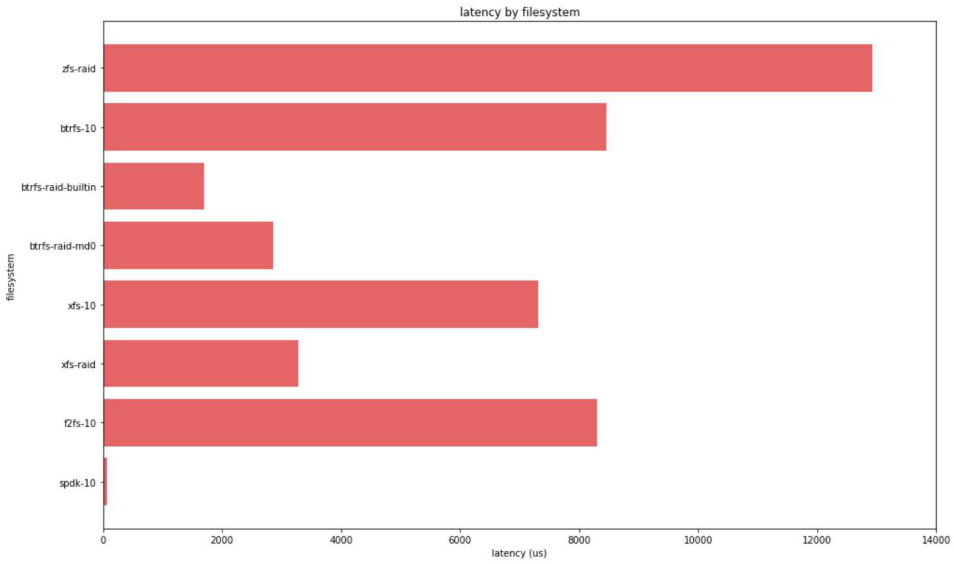

Figure 2-5 Write latency by filesystem used (lower is better)

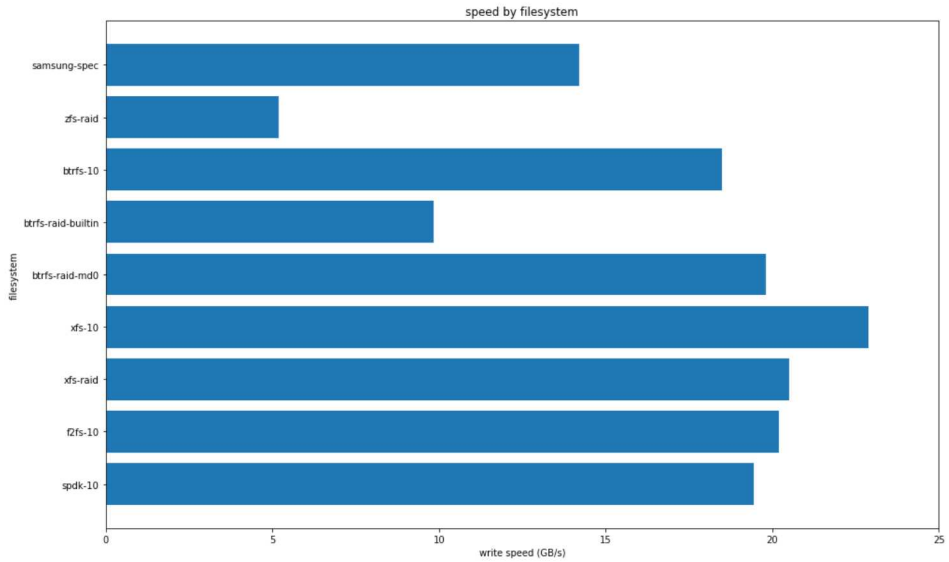

Figure 2-6 Write speed by filesystem used (higher is better)

Of these filesystems, extra benchmarks were done on F2FS and XFS to determine optimal block size. At sizes of 4KB the latency is at a minimum, which may be an indicator of the physical block size under the FTL. However, write speed was maximized at 1MB for F2FS and 512KB for XFS, which implies that these filesystems may be doing write caching in the background to optimize speed. The results for XFS, a more mature and well supported filesystem than F2FS, are shown below.

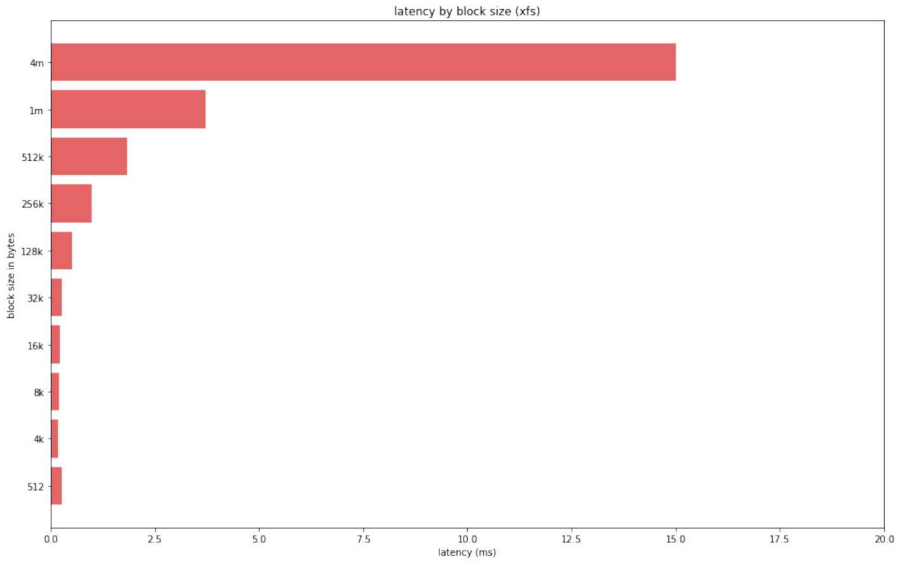

Figure 2-7 XFS RAID0 latency results (lower is better)

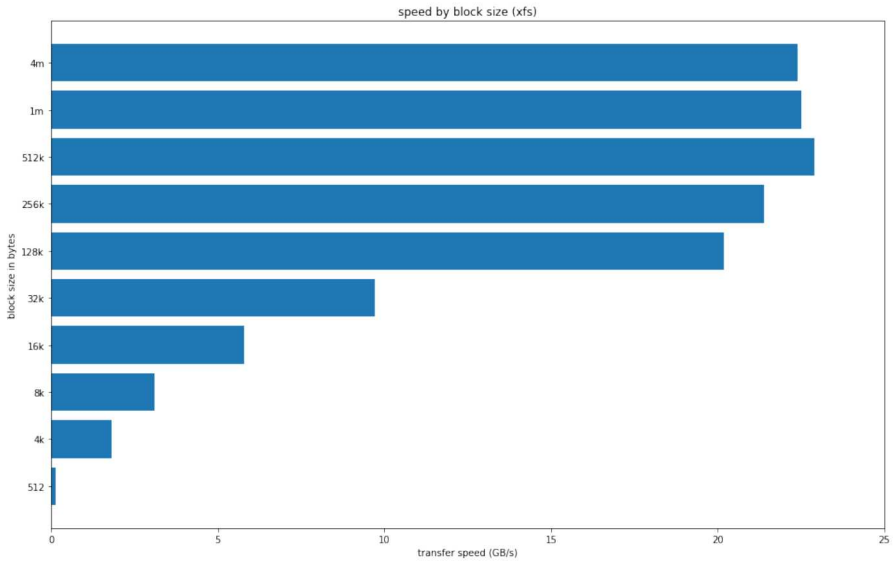

Figure 2-8 XFS RAID0 transfer speeds (higher is better)

2.3. ConnectX-5 NIC Performance

The Epyc nodes also include Mellanox ConnectX-5 NICs that are rated for 100Gbps speeds. We used iperf to help observe performance between a pair of hosts and tune our settings. Similar to previous experiments we have conducted with other 100Gbps Ethernet cards, we found that it was challenging to saturate the network link unless multiple, simultaenous transfers were performed by different threads on the source and destination nodes. It is important to note that packet size plays a significant role in performance. While it is trivial to reach line rate with many threads using 1518B packets, the NICs struggled to reach an aggregate speed of 20Gbps when using 64B packets. AMD's performance-tuning guide[2] provided excellent guidance on maximizing performance. Below is a set of commands that were used to tune the system at startup for high throughput network performance.

```
sudo systemctl stop irqbalance
sudo mlnx_tune -p HIGH_THROUGHPUT
sudo ethtool -L enp81s0f1 combined 8
sudo set_irq_affinity_cpulist.sh 40-47 enp81s0f1
sudo ifconfig enp81s0f1 mtu 9000
sudo ethtool -G enp81s0f1 rx 8192 tx 8192
sudo ethtool -C enp81s0f1 adaptive-rx off
```

[2] https://developer.amd.com/resources/epyc-resources/epyc-tuning-guides/

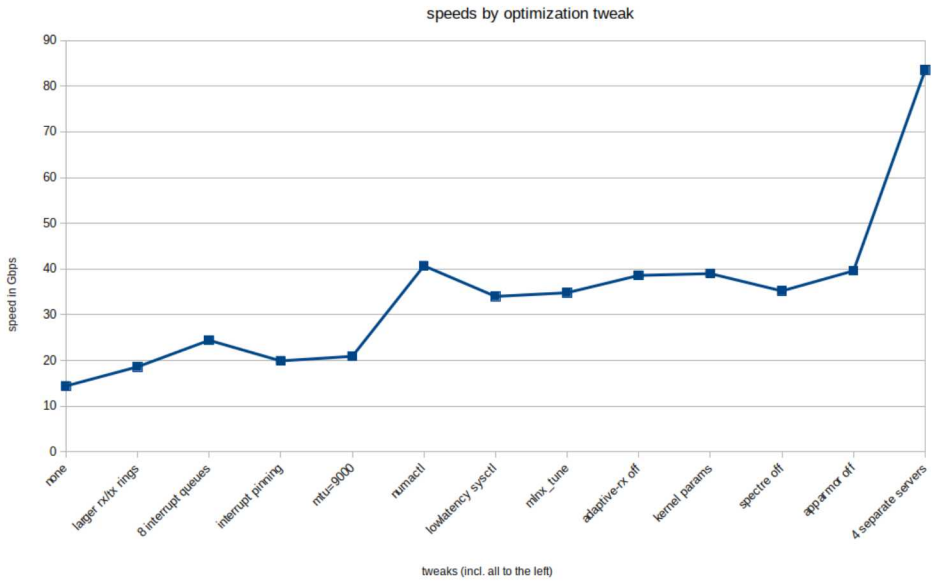

Figure 2-9 Optimizations and their effect on network speed

2.4. RAM

In the case of a fully populated set of DIMMs, the Epyc nodes show higher bandwidth when making a copy of a memory region than Carnac nodes, though a fill command has comparable speed. Interestingly, memset can be done at nearly twice the speed on a Epyc node, though a Carnac node performs memcpy about 1.5 times faster.

3. ANALYSIS OF COMMODITY PACKET CAPTURE SOFTWARE

For decades network researchers have relied on tools such as tcpdump [4] to capture and record network traffic. While tcpdump works with generic Ethernet NICs, it is common for users to experience packet loss at higher network speeds because packets must be routed through the kernel's network stack. In order to improve performance, researchers have constructed alternate packet capture libraries that bypass the kernel, enabling the NIC to send packets directly into the memory of userspace applications. The Mellanox ConnectX-5 card found in the Epyc nodes is one of the few commodity network cards that supports this kernel bypass capability. This section summarizes the characteristics of current packet capture libraries.

3.1. Comparison of Available Packet Capture Frameworks

The table below lists some commodity packet processing frameworks and some of their characteristics.

Table 3-1 Packet Capturing Libraries and Tools.

Tool	Maturity Level	Open Source	Rated Bandwidth
tcpdump	Production	Yes	<10Gbps
Wireshark	Production	Yes	<10Gbps
Bro/Zeek	Production	Yes	<10Gbps
OpenOnLoad	Production	No	>40Gbps
VMA	Production	Yes	100Gbps
Netmap	Production	Yes	>10Gbps
PF_RING	Production	Partially	40Gbps
Snabb	Production	Yes	>10Gbps
DPDK	Production	Yes	100Gbps

Tcpdump, Wireshark and Zeek all use libpcap's PF_PACKET kernel interface to process packets by default, which is significantly slower than using AF_PACKET, but using AF_PACKET requires a rebuild of the library. AF_PACKET only supports up to 9 Mpps with tuning[3], which is still too slow for high speed capture purposes, where packet rates can go up to 40Mpps. libpcap also supports using DPDK [3] or netmap [6] as a backend.

PF_RING, which works on Mellanox and Intel NICs, tries to solve this problem with a different approach. It uses NAPI so that instead of having interrupts, the capture process continuously polls the kernel, which allows it to capture at 10Gbps speeds. However, ntop also provides a closed-source zero-copy module for PF_RING, which allows only Intel cards to capture at 40Gbps.

OpenOnLoad is a set of libraries specific to SolarFlare NICs that provide a drop-in replacement for the traditional Linux network stack, bypassing the kernel entirely. VMA is similar, but works only for Mellanox cards. Both require only a LD_PRELOAD of the VMA libraries when running a program. A more general solution is Snabb, which supports Mellanox, Intel, SolarFlare, Broadcom and Chelsio cards. There is no packet capture software readily available for Snabb, so you would have to write your own.

DPDK is another kernel bypass framework that relies on poll-mode drivers. Intel NICs require a separate kernel module for DPDK support, while Mellanox cards can work with mlx5_core in a bifurcated mode that leaves the card available to the kernel. The API changes with each major LTS release, so building a pre-existing DPDK-based program may require some trial and error to find the right version. However, DPDK has the largest ecosystem of commodity software based on the library and will be the focus of the next subsection of this paper.

[3] https://kukuruku.co/post/capturing-packets-in-linux-at-a-speed-of-millions-of-packets-per-second-without-using-third-party-libraries/

3.2. DPDK Architecture

DPDK primarily consists of drivers, which may or may not map to actual interfaces. The three investigated here are the mlx5, pcap and null drivers.

The mlx5 driver binds with the Mellanox OFED kernel module and copies over data from the RX queues of the ConnectX-5 NIC into a ring buffer. A separate ring buffer is used with the TX queue on the NIC. The pcap driver is similar, but the RX buffer is linked to a pcap file that is being written to, while the TX buffer is linked to a pcap file that is being read from. The null device writes random data into the TX ring buffer and clears the RX ring buffer after being written to.

Testpmd is a forwarding program that directly transfers data between these ring buffers, while dpdk-pdump mirrors a ring buffer for its own pcap interface, like a tap on a switch.

3.3. Comparison of DPDK-based Packet Capture Tools

The table below lists open source software supporting packet capture based on DPDK and some of their characteristics.

DPDK pdump: A built-in DPDK packet capture framework that needs to make a copy of the network buffer in a separate mempool.

VPP: A software-defined router that supports packet capture and tracing, but does not record UDP packets lost and crashes when recording TCP streams >50Gbps.

Trex: A traffic generator that supports packet capture and tracing, but does not scale above 20Gbps and is limited to one RX queue.

HPCAP40G, DPDK2Disk: Dedicated research-grade packet capture software that claims to support 40Gbps speeds, but does not support the mlx5 driver. Unmaintained as of May 2018.

FlowScope/libmoon: A research-grade Lua-based packet analysis framework claiming to support 100Gbps traffic, but packets must be saved in memory before going into disk. Unmaintained as of October 2018.

DPDKCap: A research-grade packet capture tool built before DPDK's librte_pdump and unmaintained as of March 2017.

4. PACKET CAPTURE WITH LIBPCAP

4.1. libpcap with DPDK

libpcap provides support for DPDK if built with DPDK libs in /usr/lib and /usr/include. Running tcpdump with a DPDK-enabled libpcap can be done with the following command:

```
sudo DPDK_CFG="-dlibrte_mempool_ring.so -dlibrte_pmd_mlx5.so -w 51:00.1 -l 40,41,42,43,44" tcpdump -i dpdk:0 -v
```

Performance seems to match with that of dpdk_pdump, which isn't surprising considering that the code either calls memcpy on the network buffer for each iteration or passes a pcap_tmp_buf with the capture data to libpcap[4], which results in a memcpy later in the code. This is just like how librte_pdump on the DPDK library creates a copy of the buffer on each iteration.

4.2. libibverbs Dependency on libpcap

When building libpcap from source, an interesting entry in the output of the configure script mentioned checking for compatibility with libibverbs, which may be increasing the capture rate of tcpdump. However, no difference was found when using tcpdump with and without libibverbs support and the contribution history for libpcap shows that the libibverbs dependency is for offloaded RDMA traffic[5].

4.3. libpcap with VMA

Mellanox's VMA framework claims to support all Linux programs as long as they use the Linux kernel stack[6]. However, running tcpdump with the library results in identical statistics to normal tcpdump, which shows that VMA is not able to support libpcap's AF_PACKET interface. It seems to require the application to use the Linux IP stack. This may still be a solution for assembling packet captures from multiple computers onto one due to its high bandwidth and low latency.

4.4. Systemtap Instrumentation

In order to accurately pinpoint the bottleneck for packet capture, Systemtap was used to analyze the timing of the function calls in the tcpdump library, libdpdk.so, librte_pmd_mlx5.so, librte_mempool_ring.so and libpcap.so, as well as other factors affecting the system. This allows for the relative runtime of functions to be ascertained. DPDK, libpcap and tcpdump needed to be built with debug symbols, so the following line was added to dpdk/mk/rte.sdkroot.mk

```
MAKEFLAGS += -gdwarf
```

and the following was added to Makefiles in the libpcap and tcpdump directories.

```
CXXFLAGS += -gdwarf
```

[4] https://github.com/the-tcpdump-group/libpcap/blob/master/pcap-dpdk.c
[5] https://github.com/the-tcpdump-group/libpcap/pull/585
[6] http://www.mellanox.com/related-docs/prod_acceleration_software/VMA_8_6_10_User_Manual.pdf

Systemtap was built from the upstream Git directory[7] without the currently broken Dyninst support. Debug symbols[8] for the linux-image package need to be installed as well. A standard configure, make and make install was sufficient.

libpcap runs on DPDK in this manner: first, mlx5_rx_burst_vec is called to create a copy of a filled RX buffer into a ring buffer is made by DPDK. After the ring buffer is full, pcap_dpdk_dispatch is called which makes a copy of the buffer for libpcap. Then it is run through a BPF program allowing all packets through with the pcap_filter function and finally written to file with pcap_dump, which opens a standard Linux file descriptor and writes to it with fwrite. As tcpdump is single threaded, writing to the file takes up many cycles that could have been used to grab more packets and results in a sizable amount of packet loss.

DPDK pdump also uses a single thread for writing to disk, but is able to use multiple threads to receive packets from the ring buffer. This creates a bottleneck at the thread that writes to disk as its calls to pcap_dump are costly. Interestingly, the packet buffer is copied twice, from the host application to dpdk-pdump and then another time to a DPDK pcap virtual interface before finally going into disk. The writer must also work as a single thread, which incurs huge delays in processing when dpdk_pdump is called. Interestingly, when the pcap_dump call is removed from the librte_pmd_pcap code, performance goes back up to become the same as before, meaning that pcap_dump is the main bottleneck here. This method is also slower than forwarding to a libpcap device, as it uses two ring buffers that are copies of each other instead of just one, greatly increasing the number of writes to memory needed.

In order to get the probe latency to a minimum, only librte_pmd_mlx5.so was instrumented on a later run of testpmd. Here one can see that each call of the mlx5_rx_burst_vec function runs in slightly over 5,000 cycles, which is less than the maximum possible runtime of ~4,900 cycles for lossless capture.

5. PACKET CAPTURE TESTS

Packet capture at these high speeds is tricky due to the low amount of cycles allotted for each packet. Given the max line rate of 40Mpps for 64-bit packets, the 2.40GHz clock speed of the Epyc CPU and the 8 physical cores in the NUMA node adjacent to the NIC, each queue is allotted 4915 cycles to process, given that the NIC instantaneously sends packets to the CPU. That means that each packet only has about 0.6 cycles to be processed if runtime is not amortized correctly.

5.1. DPDK Setup

When using DPDK, the Mellanox OFED stack needs to be installed with the following command for DPDK 17.11 and above to build properly:

```
./mlnxofedinstall --dpdk --upstream-libs
```

[7] https://sourceware.org/systemtap

[8] https://wiki.ubuntu.com/Debug\%20Symbol\%20Packages

After install, download the latest DPDK from http://www.dpdk.org/ and unzip it. Then, run the dpdk/usertools/dpdk-setup.py script and select x86_64-native-linuxapp-gcc as the build option. Immediately cancel the build and go into the new dpdk/x86_64-native-linuxapp-gcc folder. Edit the .config file so that these lines match:

```
CONFIG_RTE_BUILD_SHARED_LIB=y
CONFIG_RTE_MAX_MEM_MB_PER_LIST=131272
CONFIG_RTE_LIBRTE_PMD_PCAP=y
CONFIG_RTE_LIBRTE_MLX5_PMD=y
```

Now run make and all test apps should show up in dpdk/x86_64-native-linuxapp-gcc/app.
The test was repeated with AMD's clang-based AOCC[9] compiler. While DPDK builds without errors, the optimizations make DPDK access uninitialized memory when starting up, resulting in an error when running testpmd or DPDK-based tcpdump.

5.2. Traffic Generation

Trex, an open source packet generator written by Cisco and based on DPDK, provides line-rate packet generation. On a Mellanox ConnectX-5, it can run at 95Gbps using 1518B frames and at 20Gbps and 60Mpps when using 64B frames. It provides a stateless mode that allows clients to connect to it using the web interface or trex-console.

Trex supports multiple threads for packet generation, as well as multiple TX queues. One needs to specify a target IP address as well as destination MAC address, which may require a gratuitous ARP request from the host for Trex to find. The figure below shows the average bandwidth attained as transmitting cores are increased (given each core has its own TX queue). The effects of NUMA can be seen in the slight decrease in output speed after all 8 cores on the NUMA node are used, while the drop between 5 cores and 6 can most likely be attributed to too much noise on the PCI lane.

The performance of Trex with various frame sizes is shown below as well. As Trex is based on DPDK, it uses the mlx5 poll-mode driver written for it. This driver happens to optimize for constant packet sizes[10], which may inflate the bandwidth and not reflect real world performance. All IMIX traffic is generated by the file in scripts/stl/imix.py, while all frame-based traffic comes from scripts/stl/udp_for_benchmarks.py.

[9] https://developer.amd.com/amd-aocc/
[10] http://mails.dpdk.org/archives/dev/2016-September/046705.html

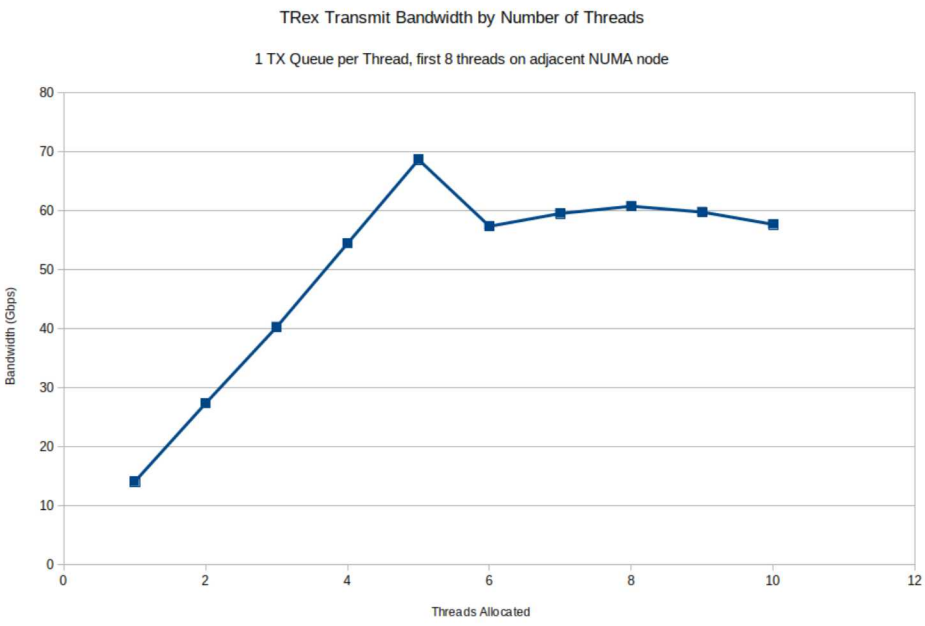

Figure 5-1 Maximum IMIX bandwidth attained by threads assigned

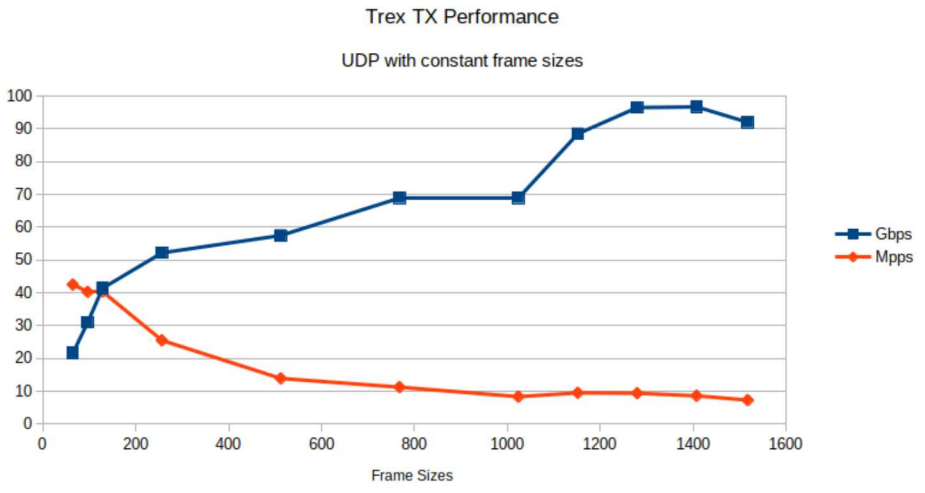

Figure 5-2 Mlx5 throughput characteristics in Trex

pktgen-dpdk is another solution to generate packets at line speed, but excels at replaying pcap files instead of generating traffic through a script. The build instructions in its documentation are well written and can be followed exactly, ignoring the DPDK build instructions. While a pcap file provided by CCD with about 160,000 streams fails to load after even 10 minutes on Trex, pktgen loads it within a minute. Shown below is a command to replay a packet at /mnt/1_5GbpsPCAP/testNetPcap using all 8 cores on NUMA node 5 on the 2nd port of the mlx5 card.

```
sudo -E ./app/x86_64-native-linuxapp-gcc/pktgen    0000:51:00.1 -l 40,41,42,43,44,45,46,47 -- -s 0:/mnt/1_5GbpsPCAP/testNetPcap -P -
    N -m [41-47].0
```

5.3. DPDK pdump Packet Capture

The dpdk-pdump library requires a separate DPDK process to be running from which it copies packet buffers. This results in increased processing times and therefore decreased packet capture rate[11]. The testpmd app was used as the main DPDK process for dpdk-pdump to capture from. A few benchmarks of testpmd forwarding without dpdk-pdump running are shown below. Packets were able to be forwarded losslessly around 500B and above, with the best performance being seen at around 3 RX queues. These performance numbers did not translate over to when dpdk_pdump was running.

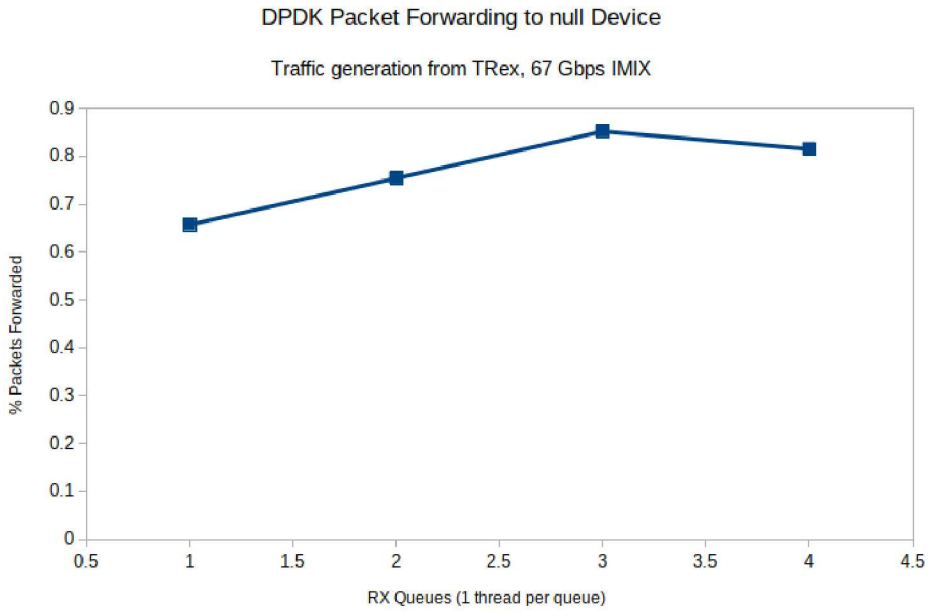

Figure 5-3 Testpmd performance by RX queue cound

[11] https://doc.dpdk.org/guides/howto/packet_capture_framework.html

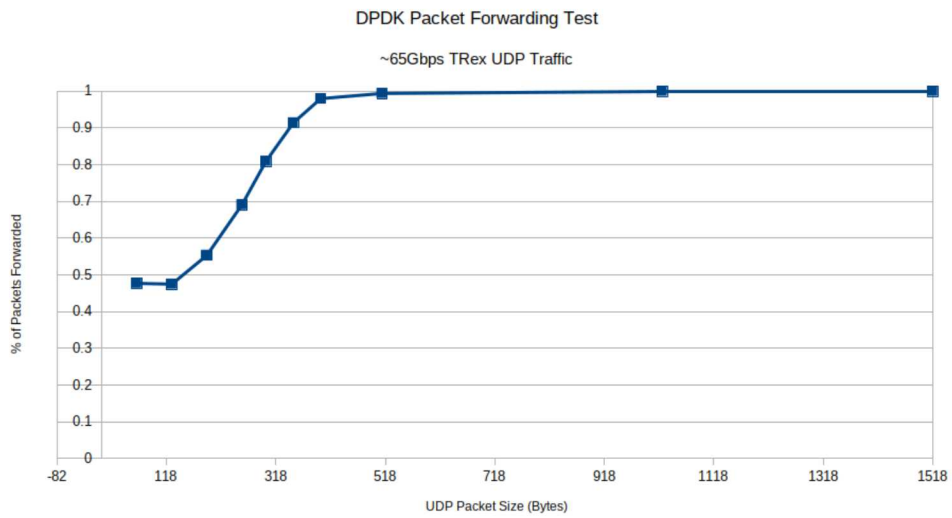

Figure 5-4 Testpmd performance by frame size

When dpdk-pdump is added onto the testpmd process, the performance degrades significantly, as shown in the charts below. Interestingly, running testpmd with RSS now results in significantly lowered performance than when running without RSS on a single RX queue. dpdk-pdump drops packets at a slower rate compared to tcpdump and shows increased performance over every network speed tested.

Intel CPUs have significantly worse performance, as the capture rate to ramdisk at 10Gbps of IMIX traffic was 36.5% on a Carnac node, compared to the 56% seen on an Epyc node, given the same number of threads. The decrease in performance is reflected in other speeds as well. With hyper-threading enabled and all cores used, the capture rate increases to 47.1%, which still fall short of the rate on Epyc. This may mean that the CPU itself is a bottleneck and shows the benefits of an Epyc processor's 4-die architecture, since the smaller number of cores in each node means that the latency of PCI devices is minimized when optimizing for NUMA. The memory bandwidth of each device may also be affecting the scores, but as Carnac's memory speed is only 20% slower than the Epyc nodes, it does not fully explain the 40% drop in performance. In addition, changing the size of the memory buffers do not change the results significantly.

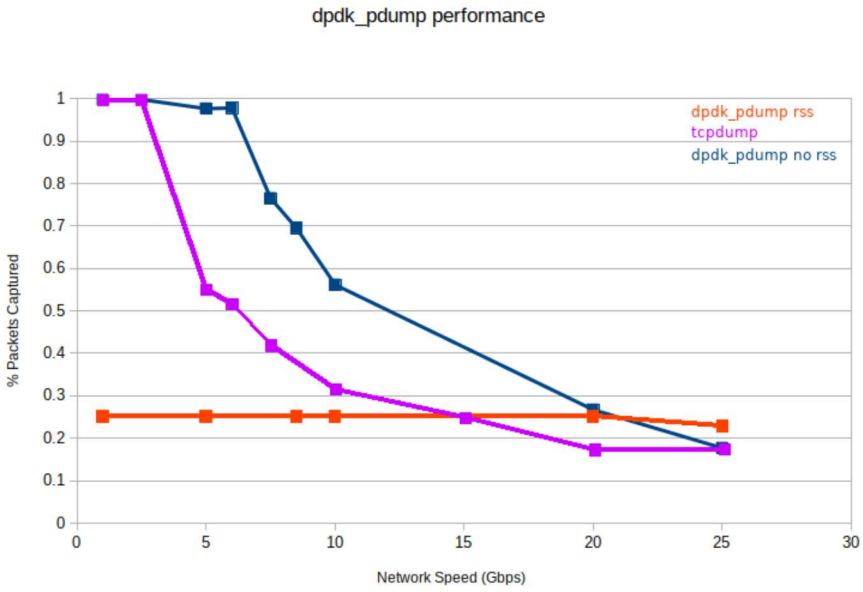

Figure 5-5 IMIX capture performances in various configurations

Packets also seem to be dropped at a uniform rate no matter the packet size. This was tested with packet captures which were collected at 1.5Gbps. These pcap files were captured losslessly when replayed at the same speed, but when replayed at line speed on a ConnectX-5 card with pktgen-dpdk(at about 35Gbps), the packet loss was significant. As shown in Figure 15, it was also evenly distributed between all packet sizes, so there does not seem to be a correlation between frame size and packets captured when dealing with realistic traffic. This implies that packets are not selectively dropped, since packet drop rate across all sizes is constant. The command for running pktgen correctly from the source directory is shown here.

```
sudo -E ./app/x86_64-native-linuxapp-gcc/pktgen \
     -w 0000:51:00.1 \
     -l 40,41,42,43,44,45,46,47 -- \
     -s P:/mnt/1_5GbpsPCAP/testNetPcap \
     -P -m [41-47].0
```

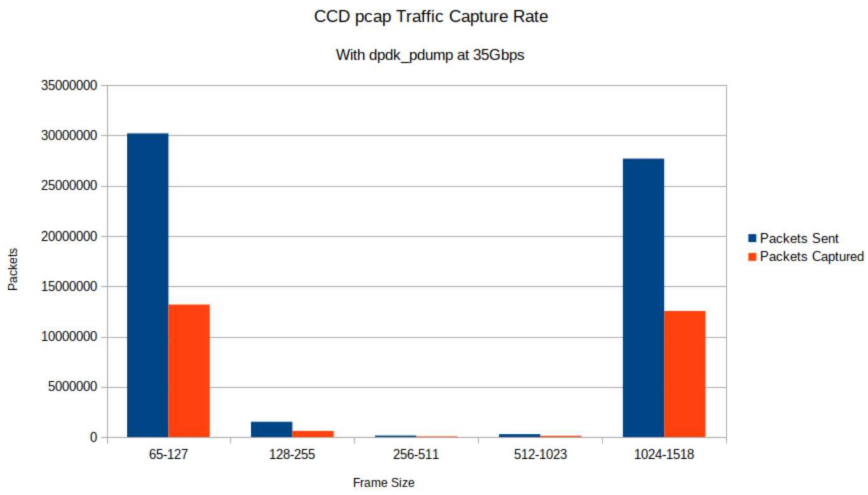

Figure 5-6 Packets captured by packet size

After adding a setvbuf call to sf-pcap.c in the libpcap source tree, performance increased from 13% of packets captured to 41% of packets captured.

5.4. DPDK with libpcap and null Interfaces

DPDK provides support for a virtual libpcap and virtual null driver, which allow DPDK to read or write packets to a pcap file as though it was another interface, or just transmit them to an empty buffer that gets cleared right after in the case of the null interface. When running testpmd with the null device, packet loss at line rate was about 3%, compared to 87% when capturing to a pcap interface, which strongly indicates that the pcap interface is a bottleneck. It may be a good idea to use this method over dpdk-pdump to capture packets as only one ring buffer is used here, compared to two identical buffers with dpdk-pdump.

Receive traffic, copy to a buffer and then clear the buffer:

```
sudo ./app/testpmd \
     -w 0000:51:00.1 \
     -l 40,41,42,43,44,45,46,47 \
     --vdev=eth_null0 -- \
     --nb-cores=7 --mbuf-size=2048 \
     --rxq=4 --txq=3 \
     --port-topology=chained \
     --forward-mode=rxonly
```

Receive traffic and run pcap_dump on it:

```
sudo ./app/testpmd \
     -w 0000:51:00.1 \
     -l 40,41,42,43,44,45,46,47 \
     --vdev="net_pcap,tx_pcap=/mnt/cap.pcap" -- \
     --mbuf-size=2048 \
     --port-topology=chained
```

6. PRIMARY PROBLEM

The primary problem with both DPDK and libpcap was the large amount of time it takes for the pcap_dump function to run, especially since it runs in a single-threaded fashion and synchronously. The two bottlenecks for this program are the kernel (since writing to a eth_null DPDK device is faster than writing to /dev/null) and the filesystem (since writing to /dev/null and ramfs is faster than writing to the RAID array).

To solve this problem, the eth_pcap_tx_dumper function in drivers/net/pcap/rte_eth_pcap.c was edited so that the mbuf itself was written to disk, instead of pcap_dump being called on individual packets. Implementation was trivial as the pcap_dumper_t struct used with pcap_dump can be cast to a FILE* for use with fwrite. This increased capture rate to 96%, though this file does not follow standard pcap format but is instead an array of rte_mbuf objects.

The experiment was done in a similar way to the previous ones, where testpmd was started with a virtual pcap device and then a pcap was replayed with pktgen-dpdk.

7. LESSONS LEARNED AND NEXT STEPS

7.1. Lessons Learned

- RAID0 though mdadm actually increases the latency of writes which is not ideal in a packet capture use case.
- While DPDK does provide nearly double the performance of Linux AF_PACKET, it still doesn't poll the RX queue fast enough for it to capture packets losslessly at line rate, though it does work at 97% of line rate on normal traffic.
- Mellanox VMA allows for any program using the Linux IP stack to use kernel bypass, but programs using AF_PACKET or PF_PACKET won't work with it.
- DPDK is fast enough to process packets at almost line rate, but the pcap virtual interface used by DPDK still relies on libpcap and that relies on fwrite, which drops the capture rate to 13% on an XFS RAID0 array.
- DPDK is not a suitable choice for packet capture as its pcap libraries rely on a single thread and sequential pcap_dump calls. However, its integration into libpcap makes it a good choice for getting a packet analysis framework like Bro to run at 100Gbps speeds.

7.2. Future Ideas

- Write metadata out to disk along with the mbuf, or make a separate program to read the mbuf file and transfer it into pcap format.

- Try using SPDK as the backend to DPDK's librte_pmd_pcap to write out directly to block devices, which will decrease latencies. This is the most complicated solution as SPDK writes directly to the block device without a filesystem backing it, but will provide the best performance as the metrics in section 1.2 show. This is better done at the DPDK side, since sending it through libpcap requires threading. An implementation of it rated up to 40Gbps can be found here[12].
- Fan out the packet capture so that multiple nodes receive it and then send aggregated packets back to an Epyc box using VMA.

[12] https://dl.acm.org/citation.cfm?id=3167238

REFERENCES

[1] AMD. Numa topology for amd epyc naples family processors. Technical report, 2018.

[2] Jonathan Crussell, Jeremy Erickson, David Fritz, and John Floren. minimega v. 3.0, version 00, 12 2015.

[3] DPDK Intel. Data plane development kit, 2014.

[4] Van Jacobson, Craig Leres, and S McCanne. The tcpdump manual page. *Lawrence Berkeley Laboratory, Berkeley, CA*, 143, 1989.

[5] Changman Lee, Dongho Sim, Jooyoung Hwang, and Sangyeun Cho. F2fs: A new file system for flash storage. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pages 273–286, 2015.

[6] Luigi Rizzo. Netmap: a novel framework for fast packet i/o. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 101–112, 2012.

[7] Ohad Rodeh, Josef Bacik, and Chris Mason. Btrfs: The linux b-tree filesystem. *ACM Transactions on Storage (TOS)*, 9(3):9, 2013.

[8] Ohad Rodeh and Avi Teperman. zfs-a scalable distributed file system using object disks. In *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings.*, pages 207–218. IEEE, 2003.

[9] Adam Sweeney, Doug Doucette, Wei Hu, Curtis Anderson, Mike Nishimoto, and Geoff Peck. Scalability in the xfs file system. In *USENIX Annual Technical Conference*, volume 15, 1996.

[10] Ziye Yang, James R Harris, Benjamin Walker, Daniel Verkamp, Changpeng Liu, Cunyin Chang, Gang Cao, Jonathan Stern, Vishal Verma, and Luse E Paul. Spdk: A development kit to build high performance storage applications. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 154–161. IEEE, 2017.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.