SANDIA REPORT
SAND2019-13774
Unlimited Release
November 2019

Networked-based Cyber Analysis using Deep Packet Inspection (DPI) for High-Speed Networks

Brian Van Leeuwen, Jason Hao Gao, Kevin Haikuo Yin, Benjamin Anthony, and Vincent Urias

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

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and 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.

SAND2019-13774
Unlimited Release
November 2019

Networked-based Cyber Analysis using Deep Packet Inspection (DPI) for High-Speed Networks

Brian Van Leeuwen (9315), Jason Hao Gao (8766), Kevin Haikuo Yin (8766), Benjamin Anthony (9315), and Vincent Urias (9315)

Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS0813

Abstract

Today's networked systems utilize advanced security components such as Next Generation Firewall (NGFW), Intrusion Detection Systems (IDS), Intrusion Prevention Systems (IPS), and methods for network traffic classification. A fundamental aspect of these security components and methods is network packet visibility and packet inspection. To achieve packet visibility, a compute mechanism used by these security components and methods is Deep Packet Inspection (DPI). DPI is used to obtain visibility into packet fields by looking deeper inside packets, beyond just IP address, port, and protocol. However, DPI is considered extremely expensive in terms of compute processing costs and very challenging to implement on high speed network systems.

The fundamental scientific paradigm addressed in this research project is the application of greater network packet visibility and packet inspection at data rates greater than 40Gbps to secure computer network systems. The greater visibility and inspection will enable detection of advanced content-based threats that exploit application vulnerabilities and are designed to bypass traditional security approaches such as firewalls and antivirus scanners. Greater visibility and inspection are achieved through identification of the application protocol (e.g., HTTP, SMTP, Skype) and, in some cases, extraction and processing of the information contained in the packet payload. Analysis is then performed on the resulting DPI data to identify potentially malicious behavior. In order to obtain visibility and inspect the application protocol and contents at high speed data rates, advanced DPI technologies and implementations are developed.

ACKNOWLEDGMENTS

The authors would like to acknowledge that this work was funded by the U.S. Department of Energy under Sandia's Laboratory-Directed Research and Development (LDRD) Program.

CONTENTS

FIGURES

NOMENCLATURE

C2	Command and Control
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DMA	Direct Memory Access
DNS	Domain Name System
DPI	Deep Packet Inspection
DSP	Digital Signal Processor or Digital Signal Processing
DUT	Device Under Test
FPGA	Field Programmable Gate Array
GbE	Gigabit Ethernet
HW	Hardware
IoT	Internet of Things
IP	Internet Protocol
M&S	Modeling and Simulation
NIC	Network Interface Card
RPS	Receive Packet Steering
RSS	Receive-Side Scaling
SDN	Software Defined Networking
SDR	Software Defined Radio
SNL	Sandia National Laboratories
SW	Software
TS	Traffic Source
OS	Operating System
VM	Virtual Machine
WAN	Wide Area Network

1. INTRODUCTION

DPI is a technology used to inspect packets transported over a network and is widely used for computer network security. The inspection provides visibility into the headers of the packet but also the payload of the packet. Initially with DPI the focus was to perform a content inspection function to seek out viruses or malware that could be embedded in packet flows using a signature-based detection approach. The signature-based approach is done using pattern-matching algorithms and is effective at detecting known threats. However, the signature-based approach used alone is ineffective against unknown threats, such as zero-day attacks and threats that evolve over time or mutated threats. Additionally, the signature-based approach is not effective in stopping threats that use evasion techniques that obfuscate malware.

Another approach is to use DPI for network analysis. In addition to pattern matching detection, the DPI capability is used to enable full packet (i.e., through Layer 7) protocol and statistical analysis. When performing network analysis, the DPI will collect metadata from fully decoded protocols used in the packet flows and the metadata will be used in the overall analysis. Thus, a more effective approach in applying DPI for cyber security is to not only perform signature-based detection, but to employ flow correlation analytics and behavior analysis. This full-visibility approach using DPI does come with the significant challenge of: How do you implement and perform DPI on today's high-speed networked systems used in enterprises and our Nation's critical infrastructures?

Our contribution:
Current domain knowledge indicates challenges with implementing an inline capability of packet visibility and inspection at the high-speed network data flows and performing the DPI-based network analysis. Our research identifies the key subsystems, both hardware (HW) and software (SW), and compares and measures different options available to achieve the full-packet (i.e., through Layer 7) pattern-matching and metadata extraction in real-time. Various SW solutions exist for performing DPI each with their benefits and limitations. No current SW solution will fully meet our objectives and thus the SW solutions are extended, in careful concert with HW, to meet our objectives.

We identified open-source software solutions (e.g., nDPI, Zeek/Bro, PF_RING, etc.) and hardware solutions that can be deployed as high-throughput network packet processing systems. We target the use of open-source software since commercial and closed-source DPI software toolkits are often not extensible by end users and come with very expensive license and yearly maintenance costs. In previous work, these open-source DPI software toolkits have been deployed on non-specialized hardware and have resulted in DPI solutions with limited network speeds (e.g., maximum of 10 Gbps [6]). Our objective is to deploy the DPI software on specialized hardware architectures designed for network processing to achieve DPI at higher data rates.

2. BACKGROUND

As processors become increasingly multi-core and parallelized, achieving higher-performance network traffic analysis will require software that can take advantage of these multicore processors by distributing work across many different instances of the traffic analysis software, known as workers. Extensions to take advantage of multicore processors include one using Zeek [1]. They describe scheduling strategies in this work [1] that take existing DPI analyses and automatically parallelize their processing. The previously available open-source DPI toolkit known as OpenDPI had issues with deployment as a multi-threaded application, and consequently, the successor open-source project nDPI has addressed this problem and thus is better positioned for deployment on modern, more parallelized computer systems [6]. Existing system architectures for distributing traffic analysis work across workers to achieve high line-rates (e.g., 40 Gbps or 100 Gbps), however, require the use of many levels of hardware and software parallelization [2], and thus expensive compute clusters, racks, specialized network switches, and computational overhead, as shown in Figure 1.

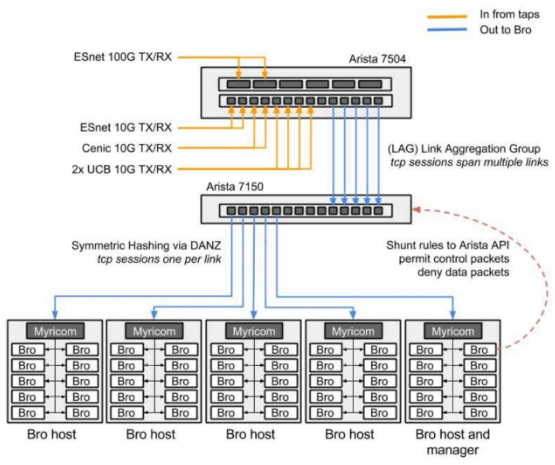

Figure 1. A typical cluster architecture for high-speed network traffic analysis. From [2].

3. DPI HARDWARE AND SOFTWARE STUDIED

By leveraging specialized network interface hardware, increasing integration of modern computer systems, and the high core counts of modern processors, our work eliminates most of the expensive network switching and cluster infrastructure necessary to perform DPI at high line-rates, and reduces the overhead of network traffic distribution to software workers performing the traffic analysis.

General architecture for Deep Packet Inspection

To perform DPI on a stream of network traffic a system must perform a variety of tasks. These tasks can be implemented in either HW or SW depending on the system configuration and capabilities. At a high level, the system tasks include:

1. *Packet ingest:* Read incoming packets off the wire so they can be processed by the system.
2. *Flow steering:* Determine which packets belong to which traffic flows.
3. *Load-balancing:* Distribute packets to various CPUs, CPU cores, and worker threads, according to the flow to which the packet belongs.

4. *Flow analysis:* Execute worker threads, in a parallel fashion, to analyze the flows and perform the DPI functions.

Further details on each of the above tasks are described below.

Task 1 - Packet ingest:
Packet ingest is performed by the NIC which has connectivity to the high-speed data stream.

Task 2 - Flow steering:
Flow steering can be performed in either HW or SW. In the case of HW it is performed on the NIC. In the case of SW, flow steering is done on the host system.

Many NICs include HW features such as Receive-Side Scaling (RSS), Receive Packet Steering (RPS), or other vendor-proprietary features that can be utilized to determine, in a reasoned manner, which packets should be assigned to which flows and/or processing queues.

Flow steering can also be performed in SW running on the host system instead of utilizing the HW NIC features. In this case, a SW program on the host machine examines every packet to determine which flow it belongs to. This is often done in SW because HW-based features may not be available, may not be flexible enough, or may consume other limited system resources (e.g., HW interrupts).

Task 3 - Load-balancing:
The load-balancing task is the assignment of flows to specific worker threads. This task is closely related to Task 2 and may be performed with the support of NIC HW features. In the HW case the NIC itself may assign flows to specific queues that steer packets to various worker threads. In the SW case, this task is performed purely in SW. The SW creates queues of packets in main memory and make these queues available to various worker threads.

The advantage of performing load-balancing in SW is the increased flexibility of how flows are assigned to worker threads. But there is also a disadvantage in that it requires a dedicated processor core to run the SW load-balancer. This dedicated processor may not be able to keep up at high throughputs. The advantage of HW load-balancing is that the NIC has dedicated hardware for high-speed hashing of packets to assign them to worker threads. However, this HW approach has the disadvantage of not being as flexible. Utilizing HW load-balancing requires creating multiple NIC queues, which consume additional host processor resources such as HW interrupts.

The worker threads may be running on the same or different CPUs and/or cores as the CPU that is receiving packets from the NIC. Even if HW flow steering is utilized in Task 2 the NIC is still electrically connected to a specific CPU socket even in a multi-socket system. Thus packets must transit that CPU before arriving at the eventual destination.

Task 4 - Flow analysis:
Flow analysis is the execution of the actual analysis SW and algorithms on a traffic flow. The SW execution for each individual flow is run on a worker thread, which is a subset of the entire set of

traffic flows. The worker threads performing flow analysis operate in parallel to achieve the total aggregate throughput of the system.

Hardware used for Deep Packet Inspection

Specialized HW was researched and tested with the objective of enabling a high-speed DPI SW solution. Next, SW approaches that can be implemented on the specialized HW to implement full packet parsing and production of metadata descriptions was researched. The SW approaches had to have support to work with the range of protocols included in the packet flows. Current literature describes advances being made in the domain of intelligent packet processing on hardware platforms [3] [4] that support customization through software application programming interfaces (APIs) as well as physical connectivity. However, there remain significant challenges to perform DPI on high-speed networks with the full range of protocols. HW challenges include:

- *Cache and memory latency:* At speeds of 40 Gbps, packets are received in times shorter than cache line access latencies [5].
- *PCIe bandwidth:* 40 Gbps and 100 Gbps line rates begin to approach the theoretical maximums of a PCI Express 3.0 x16 slot, which provides a theoretical maximum of 15.75 GB/s, or 126 Gbps [6].
- *Interconnect bandwidth:* Packets that are steered to processors cores on other sockets must traverse the processor-to-processor interconnect. Intel’s UltraPath Interconnect (UPI) provides a maximum theoretical unidirectional bandwidth of 20.8 GB/s, or 166.4 Gbps [7].
- *Memory bandwidth:* If load-balancing and steering of packets to other worker threads is performed in software on a single core, this core must load and hash packet data at high speed. The single-threaded memory bandwidth of a recent Intel Xeon Scalable processor is only around 100 Gbps [8].
- *Software architecture:* Existing DPI software, such as Zeek, were designed in an era before extreme parallelism, and is not architected to leverage modern hardware systems that contain as many as 48 to 72 cores.

In our research and implementation of high-speed DPI we focused on two specific HW multicore processors. First, we investigated and tested the Cavium OCTEON III MIPS64 processor. After obtaining disappointing results from the Cavium OCTEON III we investigated and tested Intel x86 multicore processors.

Cavium OCTEON III:

Our initial HW starting point was to leverage the Cavium OCTEON III, which supports over 100 Gbps of application processing in a single chip and supports multiple DDR3/4 channels and over 500 Gbps of I/O connectivity [9]. This HW platform provides a SW development kit that can leverage four DDR3/4 memory controllers, a large L2 Cache, and complete application acceleration for packet processing, encryption and decryption, and DPI with RegEx.

The OCTEON III is a family of processors based on MIPS64 compute cores and associated hardware accelerators. In our research the specific processor that was evaluated is the Model CN7890. This model is the largest configuration of the OCTEON III with 48 general-purpose MIPS64 compute cores. The CN7890 is installed on a Liquid IO II Smart NIC in a PCI Express daughter-card form factor. The OCTEON III NIC was installed in an x86 host computer for our testing. Note that the specifications of the x86 host machine are not important as we performed all our development and testing on the OCTEON III system itself, and the x86 host machine was only used to interact with the OCTEON III bootloader to load binaries.

The OCTEON III is designed for high-performance packet switching tasks, and thus, its hardware architecture, shown in Figure 2, is optimized for efficiently moving packets from the network interfaces to main memory. Packets in main memory can then be operated on by the MIPS64 cores. After processor operation, the packets are then sent back out to the network with rewritten IP headers if it is performing IP forwarding and routing. For example, OCTEON III has dedicated hardware blocks to handle packet input to and packet output from main memory called PKI and PKO, respectively. Dedicated hardware in OCTEON III is used for hashing packet headers to assign packets to flows and for multiplexing. Assigning packets from various flows to different queues and cores is done in such a manner as to avoid the need for high-overhead memory access synchronization mechanisms, such as locks.

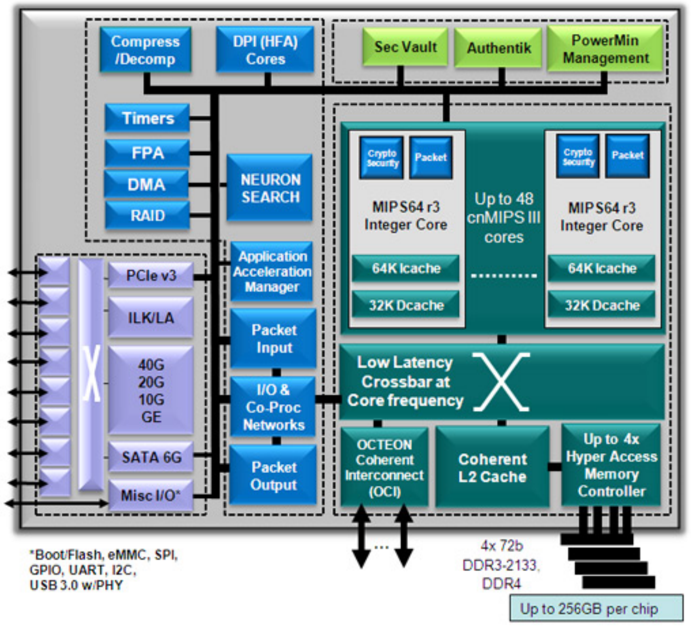

Figure 2. OCTEON III block diagram. From [9].

Although the OCTEON III is designed for packet switching and not necessarily packet inspection, the availability of many compute cores motivated our investigations into its potential as a DPI platform. Unfortunately, these compute cores are not as performant as typical x86 compute cores. Additionally, the existing OCTEON III products available have 40 GbE support but do not support 100 GbE. Thus, testing was done at 40GbE and no testing at 100 GbE was done on the OCTEON III. Although Cavium believes that future releases of OCTEON III products will support 100 GbE.

Intel x86 Processor:

After investigating and testing the Cavium OCTEON III processor we determined it would not suffice for 40 Gbps or 100 Gbps of DPI due to being compute-bound. Next, we turned to investigating how to leverage commodity x86 processors to achieve the scale and throughput needed. The Intel Xeon Scalable family of x86 processors are commodity processors widely available on a variety of standard x86 server platforms, supporting multi-socket configurations with up to 28 x86-64 cores per socket. Although a single Xeon CPU contains fewer cores than the OCTEON III CN7890, each of these cores is more powerful (as shown in our results). Furthermore, it is possible to utilize multiple CPUs in a single system with a multi-socket server. However, the Xeon CPUs, being targeted at a broad range of applications, does not include network processing features that are included with the OCTEON III, such as specialized hardware accelerators that increase the speed of its packet processing.

For our tests at 40 Gbps, we purchased a current two-socket server with two Xeon Gold 6254 processors at 3.1 GHz base frequency and 192 GB of DDR4-2933 RAM. This server was equipped with an Intel XL710 40 Gbps NIC. For our tests at 100 Gbps, we purchased a current four-socket server with four Xeon Gold 6254 processors at 3.1 GHz base frequency and 1.5 TB of DDR4-2933 RAM. This server was equipped with a Mellanox ConnectX-5 100 Gbps NIC. In both systems, the NIC is electrically connected to a single CPU socket on the system via a PCI Express 3.0 x16 slot. This configuration provides a maximum of 15.75 GB/s of throughput.

Software used for Deep Packet Inspection

Several open-source SW approaches for DPI were researched. The SW approaches investigated were able to be implemented on the specialized HW with the objective to obtain full packet parsing and production of metadata. The SW is described in the following sections.

The Zeek Network Security Monitor:

The Zeek software distribution is an open-source network traffic analysis framework originally developed at Lawrence Berkeley National Lab [10]. It performs flow analysis as described in Task 4 above. Each Zeek worker is a thread of computation to analyze the traffic flows assigned to it. The various analyses that Zeek can run on traffic flows are called *analyzers*. Zeek includes multiple default analyzers such as application layer decoding, TCP connection analysis, DNS traffic analysis, signature detection, etc.

Zeek was deployed and tested on both the OCTEON III and x86 systems for testing the flow analysis capability.

PF_RING:

PF_RING [11] is set of software tools and components that perform Tasks 1, 2, & 3 noted above. The tasks are packet ingestion, flow steering, and load balancing. PF_RING includes open-source SW but also includes some proprietary and closed-sourced SW components. Some PF_RING modules and components are described below.

1. *PF_RING Zero Copy (ZC):* A Linux network driver for specific, supported NICs that enables more efficient packet ingest by bypassing the Linux kernel's network stack.
2. *PF_RING*: A Linux kernel module that improves the efficiency of packet ingest by applications from the kernel and provides a specific API to user-space applications.

3. *zbalance_ipc:* A SW load-balancing program provided as part of the PF_RING SW distribution. This load-balancer distributes packets to various worker threads via Inter-process Communications (IPC) in a flow-aware manner.

PF_RING was used only for the x86-based testing and was not used for the OCTEON III implementation and testing.

Netmap:

Netmap [12] is a SW framework and set of tools analogous to PF_RING. Netmap provides drivers for efficient packet ingest by bypassing the kernel network stack and SW load-balancing capabilities. Similar to PF_RING Zero Copy, Netmap only supports specific NICs, and utilizes various techniques to improve performance, such as careful memory management and pre-allocation, kernel bypass, and modification of device drivers.

Netmap was used only for the x86-based testing and was not used for the OCTEON III implementation and testing.

4. CAVIUM OCTEON III FOR DPI

We implemented and conducted testing of the Cavium OCTEON III in two phases: 1) using the bare-metal *Simple Executive* SDK to implement a basic Layer-1 through Layer-5 metadata extraction program, and 2) using the OCTEON Linux SDK to run the Zeek network analyzer SW.

Simple Executive-based testing

The Cavium-provided *Simple Executive* development environment provides a low-level C-based API that allows user programs to access the HW directly without going through an operating system kernel and the associated overhead. However, this comes with a cost of lacking the operating system libraries and APIs that are more familiar to developers and provide pre-built capabilities.

A typical operating system, such as Linux, on the other hand, provides a defined and cross-architecture set of programming interfaces to the operating system known as system calls. System calls are used when applications wish to interact with system resources, such as the network interfaces or the file system. This eases development and increases code reuse, at the cost of some indirection and overhead. Most existing network analysis tools, including Zeek, run on top of an operating system and thus make heavy use of the operating system kernel APIs.

Thus, with just the *Simple Executive* environment we could not test Zeek since Zeek expects to be run in an operating system environment. Zeek makes extensive use of kernel APIs and would require extensive porting and refactoring to work in *Simple Executive* environment. This extensive porting and refactoring were beyond the scope of our research. To overcome this we implemented a simplified network analysis tool that operates on L3/L4/L5 packet-level metadata, which we call our *metadata extraction engine.* With this simplified tool we can still test key aspects of the performance of the OCTEON III for network traffic analysis.

OCTEON Linux-based testing with Zeek

The OCTEON III also supports a Cavium-provided minimal Linux environment called OCTEON Linux. OCTEON Linux includes a MIPS64 Linux kernel and MIPS64 toolchain based on *gcc*. Zeek and its various dependencies were cross-compiled for this target environment so that the Zeek traffic analysis could run on the general-purpose MIPS cores of the OCTEON III.

There were several technical limitations to the OCTEON III environment that prohibited a full end-to-end test of Zeek. First, we had to use an older version of Zeek, version 2.4, because later releases of Zeek required language and compiler features that were not available in the provided toolchain. Second, the OCTEON Linux drivers for the Ethernet interface limited the maximum number of RSS queues to 32. This meant that it was not possible to use HW-based RSS to distribute packets to all 48 MIPS cores on the OCTEON III adapter. Also, the lack of MIPS64 supports in the SW load-balancers we considered meant that we were unable to run end-to-end full-load DPI measurements on the OCTEON III.

Thus, for our measurements, we characterized the performance of an individual MIPS core on a sample Zeek dataset. From this characterization, an estimate of the best-case linear scaling to 48 cores was obtained. Additionally, an upper bound on the hypothetical fully-loaded end-to-end DPI performance of Zeek on the OCTEON III was obtained.

Measurements and Results of OCTEON III Testing

Traffic Source Tested:

For the OCTEON III tests we replayed a packet capture of typical enterprise traffic. This traffic was labeled *realistic traffic* in our results. This packet capture was replayed at 40 Gbps to the QSFP Ethernet interface of the OCTEON III. The packet capture contains 6 million packets, with 56,540 non-flow packets (e.g. UDP, ARP, etc.) and 5,943,460 flow packets (e.g. TCP) in 145,255 flows. The average packet size was 1260 bytes.

Simple Executive-based testing:

We initially tested the OCTEON III with our simplified network analysis tool, the *metadata extraction engine*, described above. The tests ran under the Simple Executive-based environment, which means that the compiled binary code interfaced directly to the HW units on the OCTEON III without going through an operating system or system calls. Our tool reads and extracts source and destination IP addresses, TCP/UDP port numbers, DNS traffic metadata, and TLS/HTTPS traffic metadata from live traffic on the network interface of the OCTEON III.

The OCTEON III was able to analyze this traffic at the full **40 Gbps** line rate without any dropped packets. In this experiment the OCTEON III used two of the 48 MIPS cores. This scenario exemplifies what the OCTEON III is particularly well-suited for, which is simple inspection of packets at high packet rates, using the close-to-the-hardware Simple Executive environment to avoid the overhead of an operating system.

This result was promising and indicated that the OCTEON III HW is able to handle the high packet rates and raw throughput necessary to reach line rate. With these initial test results, we moved forward to measurements with the more full-featured Zeek DPI analysis framework.

OCTEON Linux-based testing with Zeek:
Our *metadata extraction engine* validated the ability of the network hardware to handle the packet rates and throughputs found in the *realistic traffic capture* but was not as full-featured as a tool such as Zeek. Our next step was to measure the performance of the OCTEON III for more extensive network analysis which would be more compute-bound. For our measurements of Zeek performance on OCTEON III, we used the same *realistic traffic* packet capture and instructed a single Zeek worker process to read the PCAP file from a filesystem mounted as a RAM disk. This eliminated the effect of the network interface and any storage system bottlenecks on the test so that we could isolate the compute-bound performance of Zeek on the OCTEON III.

On a single OCTEON III core, the Zeek worker achieve a throughput of **312 Mbps**. Assuming perfectly linear scaling and perfectly even load-balancing this indicates that the best-case compute throughput we can expect, if all 48 cores are fully utilized, is **15 Gbps.**

It was determined that 15 Gbps is an upper-bound on the total throughput of the OCTEON III for more extensive traffic analysis using the Zeek. Because of the 15 Gbps limitation, we did not move forward to conducting a full set of measurements with different traffic profiles, packet sizes, etc., as 15 Gbps was not within the ballpark of line rates of 40 Gbps or 100 Gbps. Additionally, the results caused us to eliminate the OCTEON III from consideration going forward.

Our experience with attempts to develop a more full-featured network analysis tool for the OCTEON III while remaining within the Simple Executive environment indicated that it would be difficult to port something like Zeek to run under Simple Executive. Furthermore, the vendor indicated to us that there are no plans for a future, higher performance successor to the OCTEON III and that some of the software development kit components were no longer supported. The vendor indicated they plan to take product development in a different direction. Thus, we concluded that any resulting software from such an effort would not be well-supported or easily maintained.

Conclusions of OCTEON III Testing

Even though the OCTEON III's raw packet throughput was able to achieve line rates, our decision was to not perform further development for the following reasons:

- Relatively-limited general-purpose compute performance of the MIPS cores for DPI,
- Lack of a mature network stack under Linux,
- Difficulty of developing software for the card due to a limited toolchain and development environment, and
- Lack of future releases of the OCTEON product line.

Thus, it was determined that it would not be viable for our purposes of developing robust, maintainable, and performant high-speed deep packet inspection systems with the OCTEON III.

5. INTEL X86 FOR DPI

We implemented and conducted testing of the Intel x86 processors described in the HW section of this report. For our experiments at 40 Gbps, the server is a recent (i.e., 2019) two-socket server with two Xeon Gold 6254 processors each having 18 cores. The processors each had a 3.1 GHz base frequency and the server included 192 GB of DDR4-2933 RAM along with an Intel XL710 40 Gbps NIC. A traffic source (TS) computer is used to generate test traffic and is also equipped with an XL710 NIC. The TS was connected directly to the x86 server with a 40 Gbps QSFP Direct Attach Copper cable. The server's NIC is electrically connected to a single CPU socket on the system via a PCIe 3.0 x16 slot that provides a maximum of 15.75 GB/s of throughput.

For all of our x86 experiments we refer to the source of network traffic as the traffic source (TS), and the server receiving and analyzing the traffic as the device under test (DUT). The DUT is the server on which we run the load balancer software and the "consumer" software (i.e., a Zeek worker process or a simple packet counter).

To perform a single experiment run, we start the load balancer and consumer on the DUT, start sending traffic from the TS, and observe the resulting packet drop rate.

Load Balancers Tested

The various load-balancing choices tested in our x86 experiments were:

- *zbalance_ipc*: A software load balancer that runs on PF_RING,
- *lb*: A software load balancer that runs on Netmap,
- *HW load balancing:* Done on the DUT NIC. This is implemented and controlled via the Receive-Side Scaling (RSS) hardware functionality of the NIC.

For tests with the software load-balancing:

- One core is dedicated to the load-balancing software, either *zbalance_ipc* or *lb*,
- One core is dedicated as a management core (i.e., Zeek management process), and
- Remaining thirty-four physical cores are dedicated to running the consumer software (Zeek or packet counter).

For the tests with hardware load-balancing, the NIC was configured to hash traffic to thirty-five RSS queues and dedicated each of the thirty-five cores to the consumer software. Thus, each available physical core (other than the management core) is running a consumer thread.

Consumer Software Tested

Experiments were performed with three choices of consumer:

- Zeek,
- Packet counter (*pfcount* for PF_RING or *pkt-gen* for Netmap), or
- No consumer.

pfcount and *pkt-gen* are programs that read traffic coming out of the load balancer but do not perform any analysis. These programs are used to test the performance with a lightweight consumer. *pfcount* and *zbalance_ipc* only work with PF_RING, and *pkt-gen* only works with Netmap and *lb*.

Running tests with Zeek as the consumer represents the full end-to-end analysis stack. This enables the measuring of the actual traffic inspection rate achieved, and at what packet loss rate. Running tests with a simplified consumer (i.e., packet-counter applications) enables the isolation and measuring of the performance of the network pipeline without the computational impact of performing DPI. Running tests with no consumer enables the gauging of performance of the load balancer and NIC by eliminating the impact of any interaction with the consumer application.

Traffic Sources used for Testing

Multiple traffic source types were used to measure different aspects of the system with regards to packet size, packet rate, total throughput, and even or uneven distribution of packet flows to the worker threads. The three different types of traffic were:

- *Random traffic*: Randomly generated traffic that is evenly distributed among randomized source-destination IP addresses to ensure even distribution of traffic to consumer queues and threads.
- *Imbalanced traffic:* Obtained from a test HTTP packet capture, and
- *Realistic traffic:* Obtained from a test enterprise traffic packet capture.

The *random traffic* can be varied with respect to packet sizes and throughputs. This enables measuring the impact of different packet sizes on system performance, and also gives us a uniform distribution of flows to the different worker threads during the load-balancing stage. The generated packets have randomized source and destination IP addresses such that packets are distributed evenly among the consumer queues and threads.

The *imbalanced traffic* is an HTTP packet capture with a few large “elephant flows” (i.e., singular flows that dominate the overall throughput and thus cause a few of the queues and consumer threads to receive many more packets than the rest). This enables the characterization of the performance in worst-case scenarios of uneven distribution of flows at the load-balancer to the worker threads. It contains 16.9 million packets in 88,042 total TCP flows. The average packet size is 1289 bytes.

The *realistic traffic* is a packet capture of typical enterprise network traffic and is used to test performance in a realistic environment likely to be seen when deployed. This is the same packet capture as used in our OCTEON testing and contains 6 million packets, with 56,540 non-flow packets (e.g., UDP, ARP, etc.) and 5,943,460 flow packets (i.e., TCP) in 145,255 flows. The average packet size is 1260 bytes.

Measurements and Results of 40 Gbps x86 Testing

The following subsections each describe one aspect of the DUT performance that we measured and the accompanying test setup.

Effect of Choice of Load-Balancer:

To measure the impact of the choice of load-balancing on the system packet processing performance and drop rate, we tested the following load-balancing setups:

- *pfring_1_q:* PF_RING with one RSS queue and software load-balancing via *zbalance_ipc.*
- *pfring_n_qs:* PF_RING with hardware load-balancing via multiple RSS queues.
- *netmap_1_q:* Netmap with one RSS queue and software load-balancing via *lb.*

Note that we do not include a setup for a "Netmap with hardware load-balancing via RSS queues" because Netmap still requires the use of the software load-balancer *lb* even when utilizing RSS queues. In our testing, Netmap with hardware load-balancing via RSS queues always performed worse than Netmap with software load-balancing via *lb.*

For each of these load-balancing setups we measured the system end-to-end packet drop rate while varying the throughput for the three traffic sources (i.e., random traffic, imbalanced traffic, and realistic traffic) and the consumer used (i.e., Zeek or the light consumers *pfcount*/*pkt-gen*). We calculated the end-to-end packet drop rate by subtracting the total packets received by the consumers from the number of packets sent by the traffic generator.

Random traffic:
Using random traffic and a simple packet counter as the consumer, we observed minimal packet loss (i.e., 0 or < 1% in all cases) up to nearly line rate, which is shown in Figure 3. Our tests did not fully reach 40 Gbps due to limits on the traffic generation that maxes out at 37.6 Gbps for randomly-generated packets. Netmap's packet loss rates rose slightly higher as we approached 40 Gbps, but were still minimal at 0.03%.

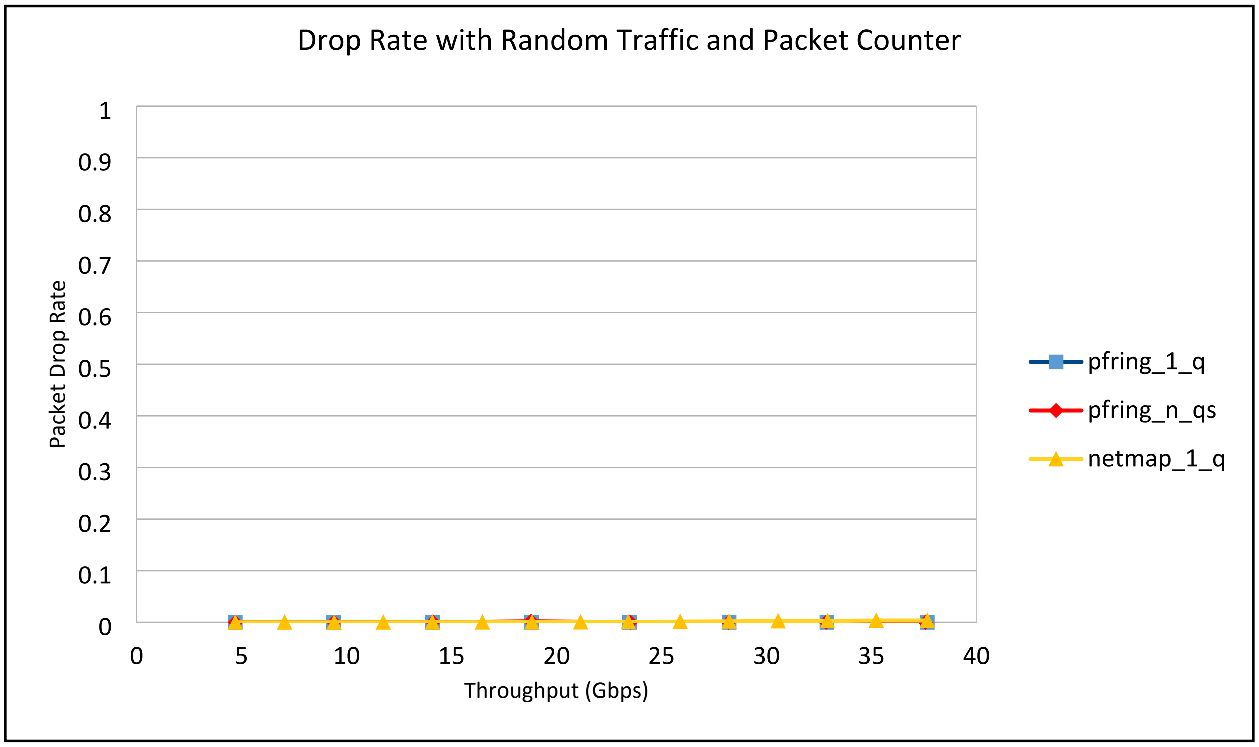

Figure 3. Drop Rate with Random Traffic and Packet Counter.

Figure 4 describes the results from Zeek workers as the consumer application when using PF_RING with hardware load-balancing. The result describes increased packet loss at all speeds and increasingly worse performance towards 40 Gbps. This indicates a consistent overhead caused by using PF_RING with hardware load-balancing and Zeek. The other two software load-balancing setups maintained less than 1% packet loss (at most 0.7%), even at up to 37.6 Gbps.

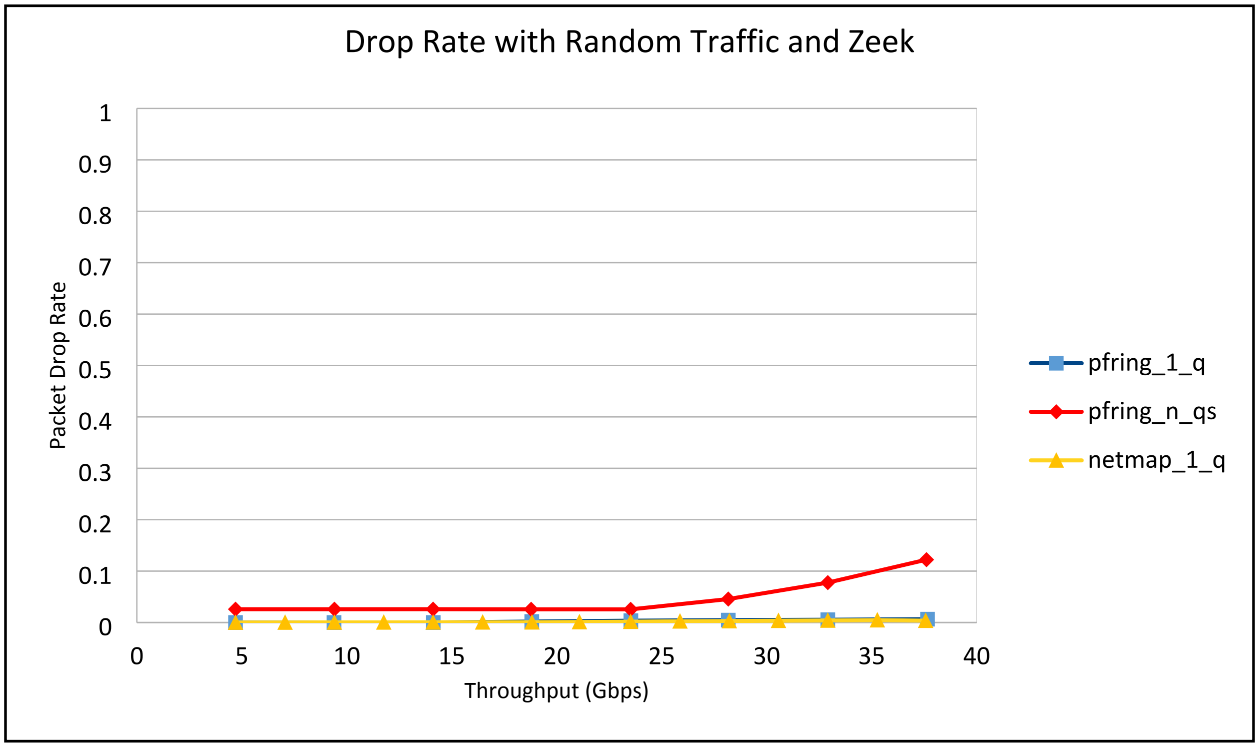

Figure 4. Drop Rate with Random Traffic and Zeek.

Realistic traffic:
Using realistic traffic, data for the same measurements as the random traffic was collected. In the case of realistic traffic, reaching a line-rate of 40 Gbps was possible because packets were replayed from a packet capture rather than dynamically generating packets.

With the light consumer, Netmap performed increasingly worse and dropped more packets. In contrast, PF_RING continued to perform well and maintained a packet loss of less than 1%. The packet drop rate is shown in Figure 5. This indicates that Netmap and its load-balancer, *lb,* did not handle realistic traffic well. The realistic traffic included some elephant flows (i.e., large SMB transfers, HTTP downloads, etc.) and varying packet sizes (i.e., small packets such as DNS, ARP, DHCP, etc.).

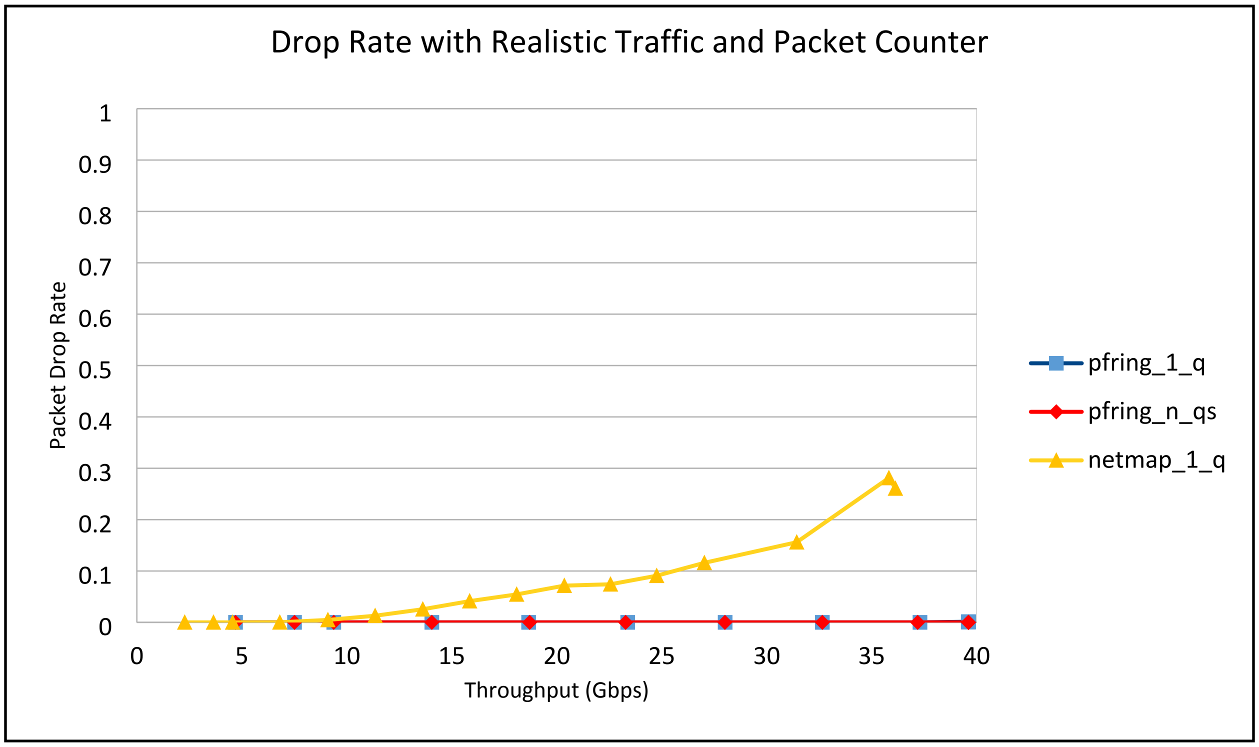

Figure 5. Drop Rate with Realistic Traffic and Packet Counter.

With the Zeek consumer, we observed PF_RING with hardware load-balancing showing consistent overhead again, as is shown in Figure 6. However, Netmap performed far worse at higher speeds, with packet drop rates near 80%. PF_RING with software load-balancing was the only setup able to keep drop rates below 1% at the highest speeds. PF_RING with hardware load-balancing did relatively well, but with higher drop rates of 3.5% at 40 Gbps.

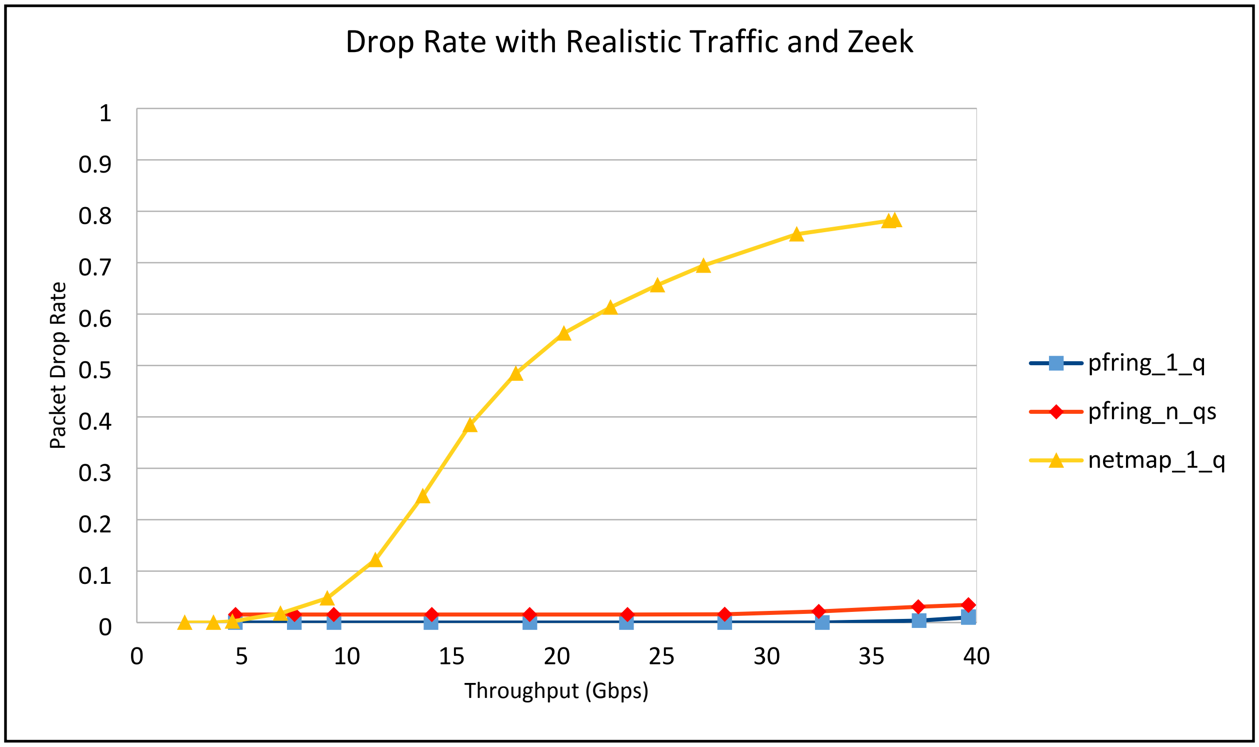

Figure 6. Drop Rate with Realistic Traffic and Zeek.

Imbalanced traffic:
With imbalanced traffic and the light consumer, the hardware load-balancing performed better than both Netmap and PF_RING software load-balancing, as is shown in Figure 7. The relative performance of PF_RING's hardware versus software load-balancer inverted when compared to the results with realistic traffic.

Also note that there was no longer the consistent minimum dropped packets that we saw previously with PF_RING with hardware load-balancing. We believe this was due to the nature of RSS functionality on the NIC. With imbalanced traffic, most packets were going to very few NIC queues, and thus most of the other queues were no longer generating as many hardware interrupts and consuming those resources.

Figure 7 shows Netmap with imbalanced traffic performing better and dropping fewer packets than Netmap with realistic traffic at the same speeds. This is because it failed to finish testing at higher speeds; thus the premature end of the data series for Netmap.

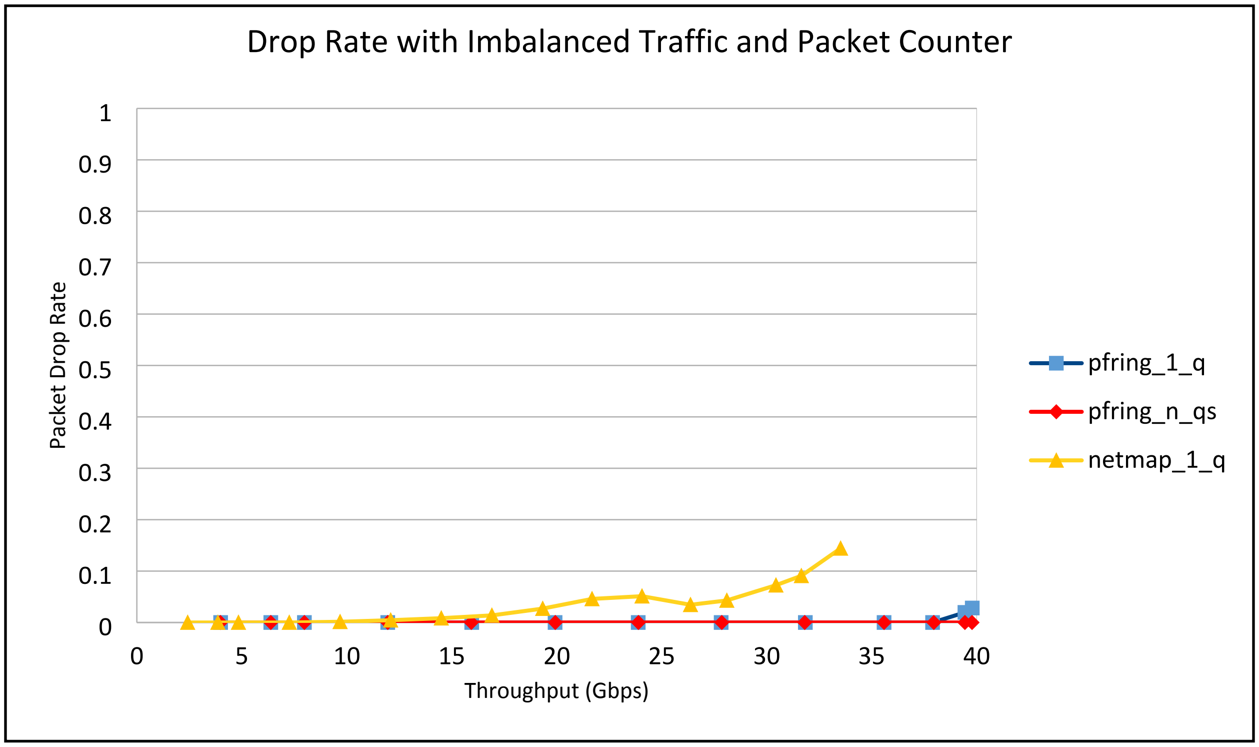

Figure 7. Drop Rate with Imbalanced Traffic and Packet Counter.

Figure 8 shows the packet drop rate with Zeek and imbalanced traffic. The figure shows drop rates increasing as expected. But with this result all configurations failed to maintain drop rates below 1%. Although PF_RING with hardware load-balancing still performed the best.

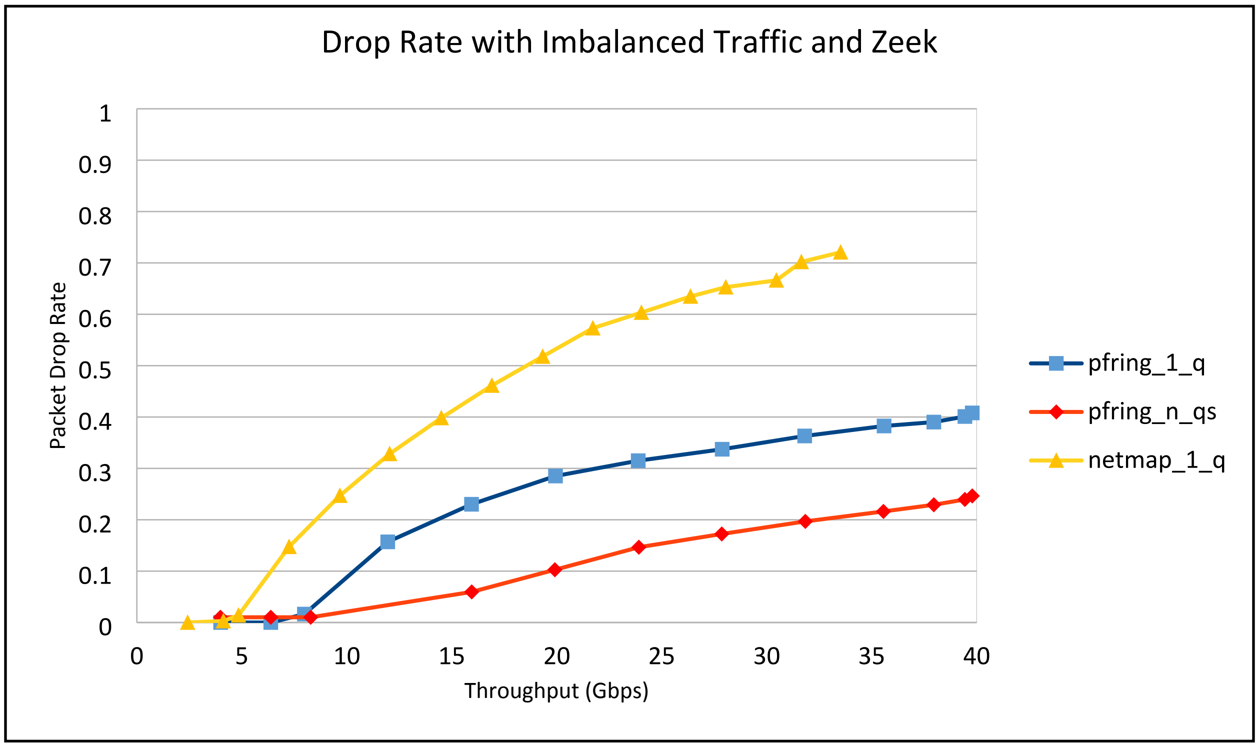

Figure 8. Drop Rate with Imbalanced Traffic and Zeek.

The results with imbalanced traffic was investigated further and two important observations were identified:

1) Upon inspecting the NIC hardware packet counters, we realized that the NIC *rx_dropped* counter increased rapidly at higher speeds. This indicates that the software load-balancer process was not servicing the NIC buffers fast enough when approaching 40 Gbps. Thus, the software load-balancer itself was failing to keep up with distributing imbalanced traffic. For realistic traffic or random traffic this was not the case. Note that *rx_dropped* was zero or near zero for random traffic and realistic traffic in the corresponding conditions.
2) Of the packets that did make it through to the software load-balancer process and the consumer processes, 62% of those packets were sent to just three queues and associated consumer processes. Additionally, packets were dropped from only the same three queues. The other 31 queues were relatively unloaded consumer processes and had zero packet drops. Thus, the increase in packet drops with Zeek versus the light consumer was predominantly from the computational load of the DPI analysis outstripping the computational capacity of those three cores.

In Figure 9 and Figure 10, the light consumer and Zeek consumer were used to show an illustrative example of the points above. The total end-to-end drop rate of PF_RING with software load-balancing was separated into two components:

1. *NIC Drop Rate:* This was due to the software load-balancer not keeping up with the service the NIC buffers.
2. *Load Balancer Drop Rate:* This was due to the consumer application not consuming packets from the load-balancer queues fast enough.

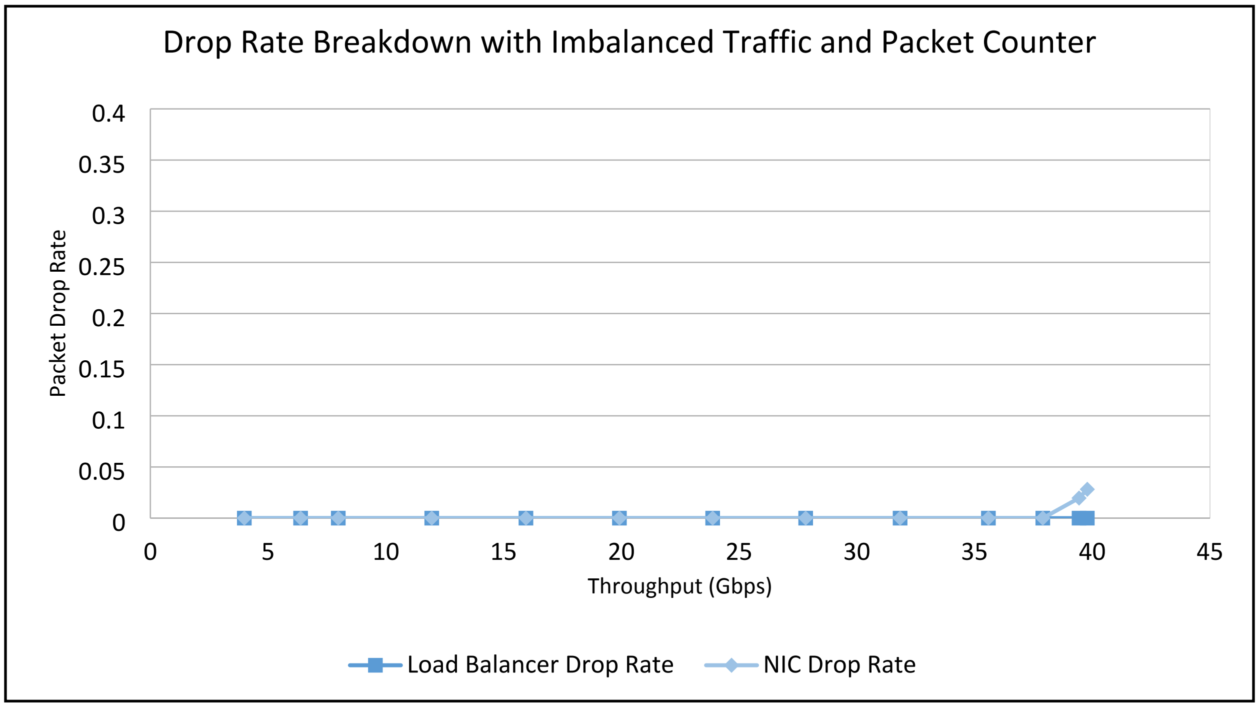

Figure 9. Drop Rate Breakdown with Imbalanced Traffic and Packet Counter.

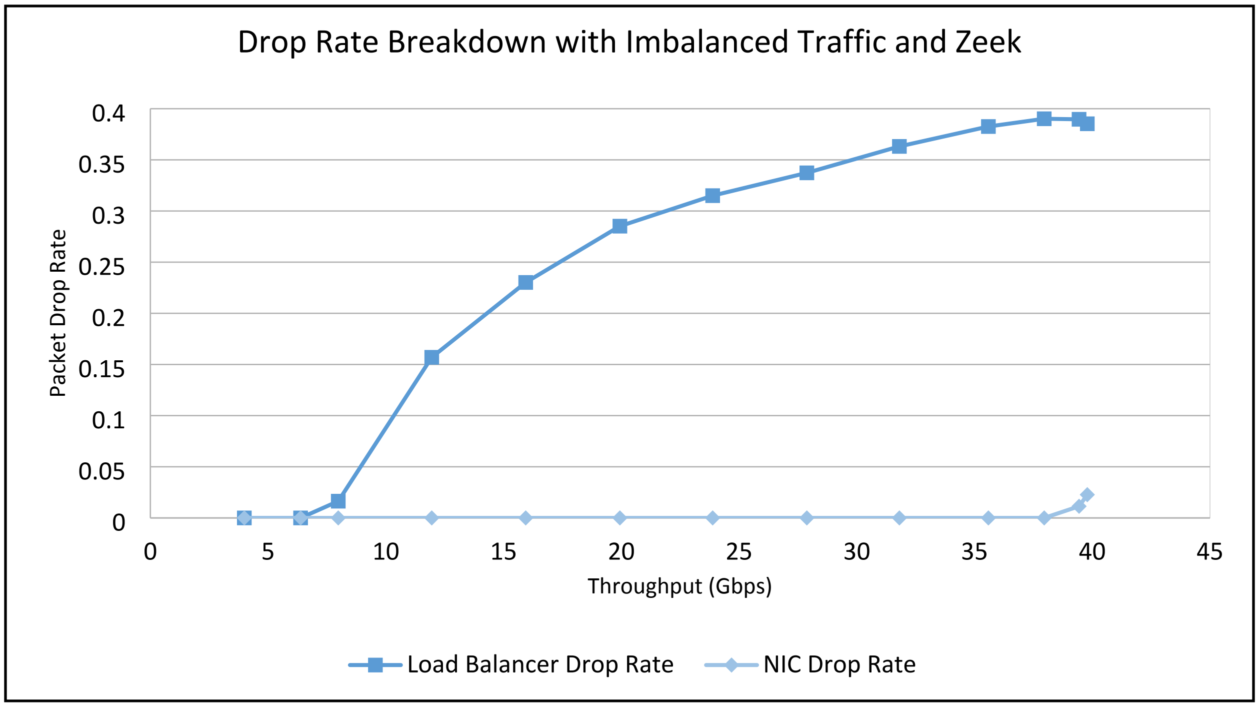

Figure 10. Drop Rate Breakdown with Imbalanced Traffic and Zeek.

In both cases, the NIC Drop Rate began to grow beyond 1% at data rate speeds above 37 Gbps. This increasing drop rate is cause for concern, especially if targeting speeds beyond 40 Gbps. From Figure 9, the plot shows that with the packet counter the Load Balancer Drop Rate remained low. From Figure 10, the plot Zeek drop rate increased rapidly even at relatively low speeds.

Effect of Packet Rate and Packet Size:

In addition to aggregate data throughput (measured in Gbps), the packet rate (measured in packets per second (pps)) offered to the packet processing system can limit the overall performance. The packet rate can limit performance because there is some overhead, both in hardware and software, associated with the processing of each packet regardless of packet size. Thus, at smaller packet sizes, the packet processing system may not be able to reach expected aggregate data throughputs (i.e., line rate; 40 Gbps). In contrast, with larger packet sizes it is possible to achieve expected aggregate data throughputs even at a lower packet rate. This effect is because the total throughput is the average packet size multiplied by the packet rate.

To measure the impact of packet size on the ability of the system to handle high packet rates while minimizing packet drop rate, we used the random traffic source. With the random traffic source, the packet sizes generated were varied and tested. The experiments used packet sizes:

1200, 1000, 900, 800, 700, 600, 500, 400, 300, and 200 bytes

The experiments were performed with the packet-counter (i.e., *pfcount* or *pkt-gen*) as the consumer. Experiments were performed with both:

- PF_RING + *zbalance_ipc* + one RSS queue, and

- Netmap + *lb* + one RSS queue.

Also tested was a configuration with no consumer: just PF_RING + *zbalance_ipc*, and Netmap + *lb*.

For each of the packet sizes, the packet rate was varied until the maximum packet rate was achieved in Packets Per Second (pps), while maintaining < 1% packet drop rate. The results are shown in Figure 11 and Figure 12.

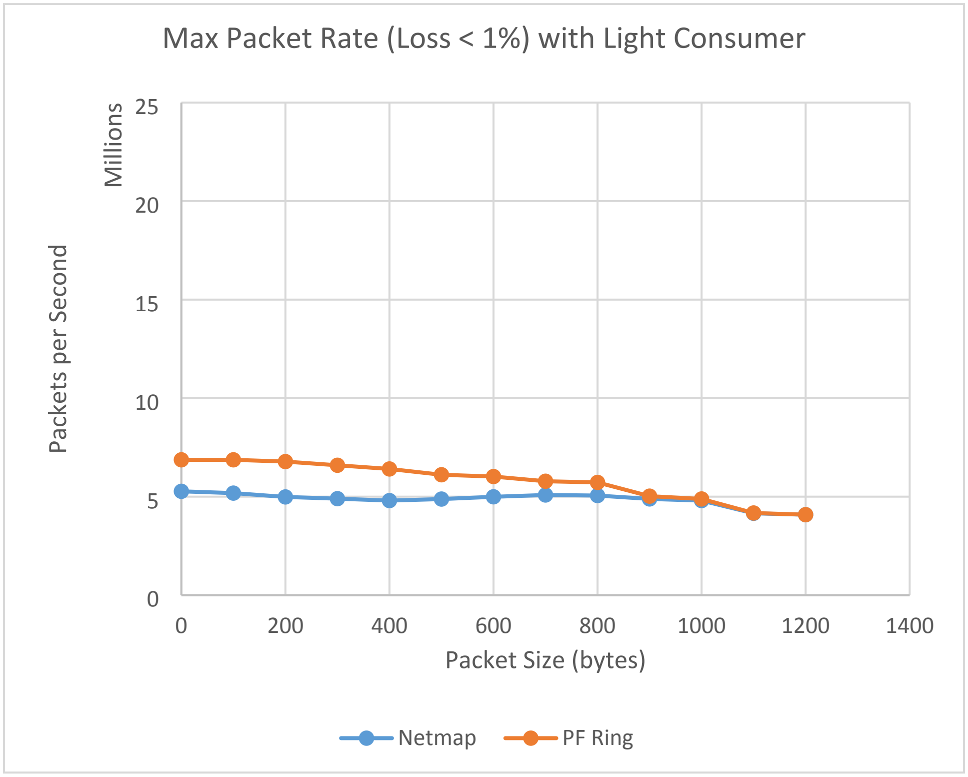

Figure 11. Maximum packet rate obtainable with loss < 1% with light consumer.

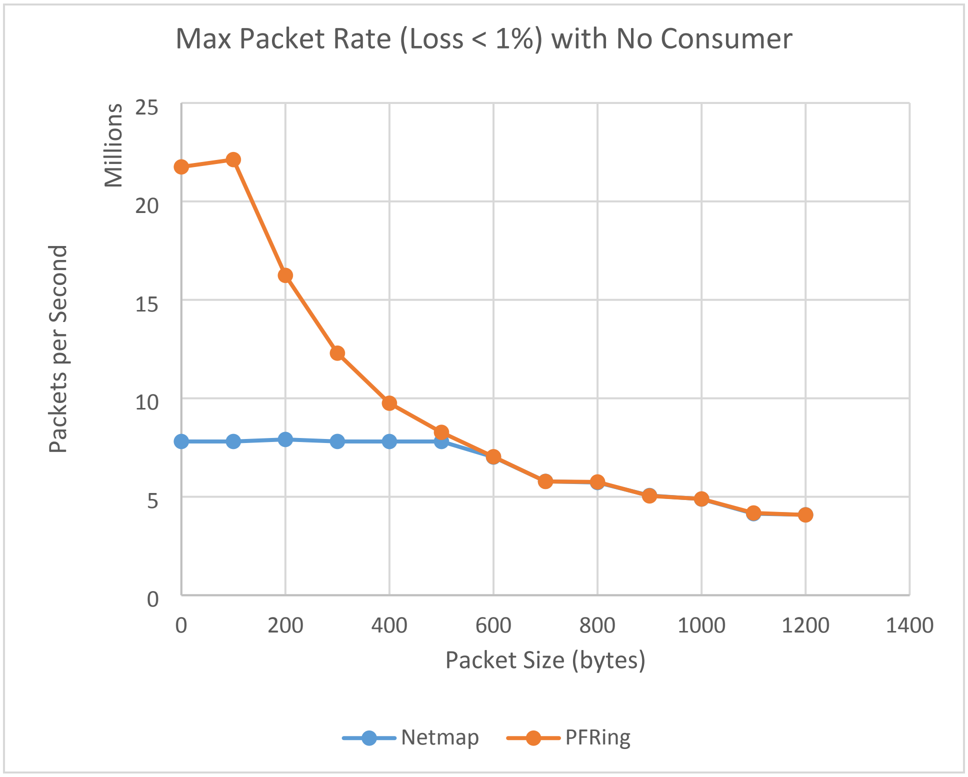

Figure 12. Maximum packet rate obtainable with loss < 1% with no consumer.

Figure 11 illustrates the performance of PF_RING and the packet counter consumer. The figure shows the system's maximum packet rate is 6.9 million packets per second and drops off at larger packet sizes. With Netmap and the packet counter consumer, the system's maximum packet rate is 5.3 Mpps. For all packet sizes, PF_RING outperforms Netmap. Note that we were not able to test Netmap with 1200-byte packets due to issues we encountered with Netmap's buffer allocation at larger packet sizes.

Figure 12 illustrates the performance without a consumer attached to the load balancer process. In these cases, both PF_RING and Netmap achieve higher packet rates to varying degrees. PF_RING was able to reach nearly 22 million packets per second, while Netmap was able to reach nearly 8 million packets per second. PF_RING's significantly higher performance without a consumer indicates that much of the bottleneck in system packet rate performance was at the interface between the load-balancer and the consumer. This was also supported by our inspection of the performance counters on the NIC during our end-to-end packet loss tests in the previous section, which indicated zero or minimal (<0.01%) packet loss at the interface to the NIC itself, and most of the packet loss occurring between the load-balancer and the consumers.

Effect of Shunting:

As noted earlier in this report, *elephant flows* are described as a few, high-throughput traffic flows that dominate the bulk of the traffic and thus cause a small number of traffic analysis workers to bear a disproportionate amount of the total work. This is because all the packets in an *elephant flow* are hashed to the same worker, even if other workers remain relatively underutilized.

One of techniques to mitigate the *elephant flow* phenomenon is called *shunting*. Shunting discards later packets in large flows with the result that the computational cost of packet analysis is only performed on the first *N* packets of a large flow.

The imbalanced traffic capture includes several elephant flows that dominate the majority of total traffic throughput in the capture. This causes a few CPU cores to be flooded with a disproportionately large portion of the packets such that they cannot keep up, resulting in a very high drop rate. In our experiments with the imbalanced traffic, we observed that 62% of all packets went to just three of the physical cores.

Shunting was implemented with PF_RING FT. PF_RING FT shunts remaining packets of a flow beyond 1000 packets. Experiments were executed, and performance was measured. With shunting, packet loss rates dropped considerably, as shown in Figure 13, which compares how many packets were assigned to each CPU core, both before and after shunting. In Figure 13, the maximum height of each bar shows the total number of packets assigned to each CPU core without shunting; note that the heights are highly imbalanced across the CPU cores. The blue portion of each bar is the number of packets processed by each CPU core with shunting enabled. The packets processed with shunting are much lower in total and more evenly distributed across CPU cores. The orange section of each bar is how many packets were shunted away and no longer assigned to each CPU core for processing.

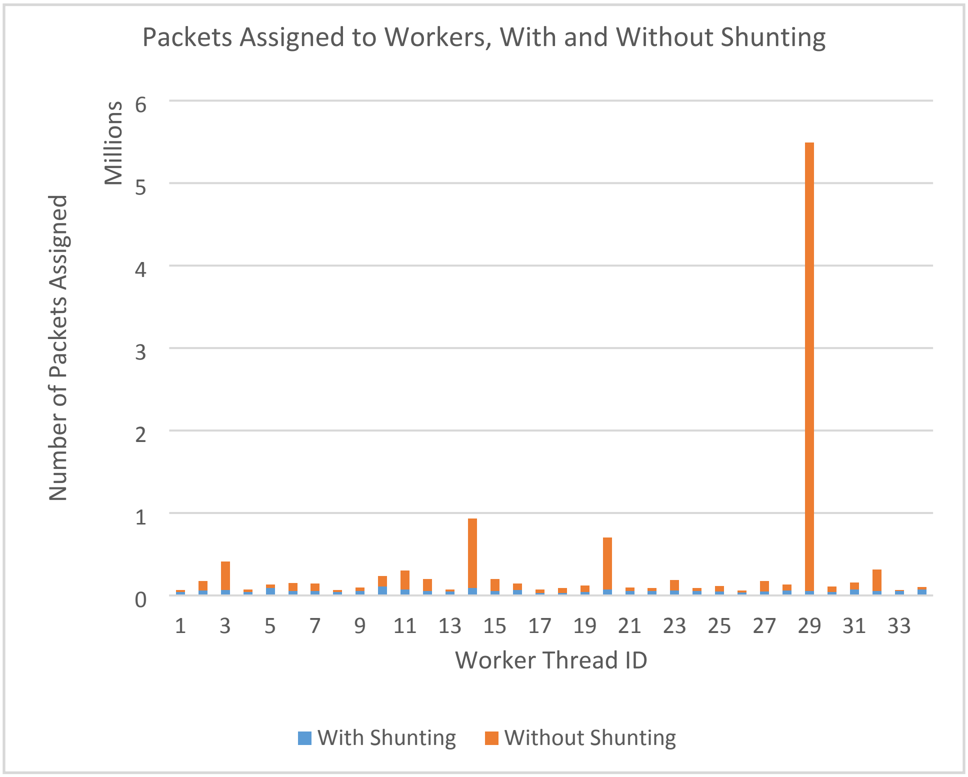

Figure 13. Packets Assigned to Workers, With and Without Shunting.

The drawback to shunting, of course, is that analysis is no longer performed on the entirety of the traffic. However, shunting has been widely considered to be effective, as the very beginning of a connection contains enough information for meaningful security analysis to happen [13]. Thus, while shunting significantly reduces the total number of packets analyzed, it does not significantly reduce the ability of DPI to detect threats and attacks.

Initial investigations into 100 Gbps x86 Testing

Our initial research objective included developing a DPI capability for 100 Gbps data rates. Ultimately, we were unable to perform experiments at 100 Gbps due to implementation issues with the software and hardware stack for 100 Gbps. The following issues prevented the 100 Gbps implementation:

- PF_RING does not currently support a commodity 100 Gbps NIC (in our case, we tested using a Mellanox ConnectX-5 100 Gbps NIC), and
- Netmap's source code repository contains patches for the Mellanox ConnectX-5 driver that correctly compiled and loaded. But when the drivers were switched into Netmap mode, packets failed to be received on the interface. We verified that the NIC did operate correctly when not using Netmap, and was instead operating through the Linux kernel network stack, but unsurprisingly, this resulted in unusably high packet loss.

Our experiments indicated that software and hardware compatibility for kernel bypass with 100 Gbps drivers was still relatively early and immature. Whereas on the 40 Gbps NIC, we encountered no significant issues with either PF_RING or Netmap. We also conducted some initial testing into newer kernel bypass technologies, such as the Linux kernel's built-in AF_XDP [14] interface, but AF_XDP support was even less mature and it did not have fully-accelerated kernel bypass in the Mellanox ConnectX-5 driver yet. The result was significant performance issues and limitations (i.e., restrictions on the size of packets). We anticipate that as time goes on, the support for 100 Gbps NICs will improve, and the continued development of standardized interfaces such as AF_XDP will ease implementation of future traffic analysis systems.

Conclusions of x86 Testing

Our implementation and testing of the various software and hardware configurations of kernel-bypass to the NIC, load-balancer, and packet consumer proved that the use of x86-based hardware and software platforms for high-throughput DPI is promising. Our experiments showed that potentially high packet rates can be achieved, but system performance is currently hampered by bottlenecks at the load-balancer and load-balancer to packet consumer / DPI worker thread interface.

With the use of shunting, a single system can handle the computational workload of DPI at 40 Gbps line-rate with reasonable packet loss. This points to the need for improved load-balancing performance, whether that is achieved through hardware and/or software optimizations.

With regards to ease of development and maintainability, compared to the OCTEON III platform, the x86-based platform was significantly easier to work with at a system level due to the wide availability of mature tooling and the broad used of x86-based system across computing. Furthermore, the availability of existing DPI frameworks such as Zeek operated well on x86-based systems. However, support for kernel-bypass drivers for NIC hardware beyond 40 Gbps was immature and lacking. Even for 40 Gbps NIC hardware, Netmap presented challenges in finding a workable combination of configuration parameters and was slightly less stable than PF_RING. Netmap had a major advantage over PF_RING in that it is fully open-source, while PF_RING is not (e.g., the PF_RING ZC kernel-bypass drivers are closed-source and require paid licenses to use).

We anticipate that the situation around configuration and compatibility concerns for both 40 Gbps and 100 Gbps NIC support will eventually be improved due to active interest and development in high-speed network processing on x86 in the literature and in industry. This is especially true around new cross-vendor kernel-bypass technologies such as AF_XDP and the wider adoption of 100 Gbps NIC hardware.

6. ENCRYPTED PACKETS AND DPI

Today's networks carry a large amount of encrypted packets, which creates difficulty when using DPI. For encrypted packets, it is not possible to do deep packet inspection. However, an architecture can support DPI of encrypted data flows if the device performing DPI is also capable of intercepting the encrypted packet and decrypting it, similar to a network man-in-the-middle attack. Another method to perform DPI on an encrypted packet would be to place an appliance in-path that brokers the key exchange between the sender and receiver. This way the appliance will see the plain-text content of the communication and DPI can obtain packet visibility and inspection. This approach is known as SSL interception (SSLi).

Initial testing with encrypted connections on OCTEON III

Initial testing was performed of the OCTEON III's ability to initiate and terminate encrypted connections and measure its possible performance if used in an SSLi scenario. In an SSLi scenario at scale, the SSLi appliance needs to act as both an SSL/TLS-enabled endpoint as well as an SSL/TLS-enabled client. This is necessary since it must accept encrypted connection requests from clients, open encrypted connections to eventual destinations on the Internet, and forward traffic between the two.

The most computationally expensive part of an SSL/TLS connection is the initial key exchange, which utilizes asymmetric cryptography. The bulk transfer of data uses symmetric encryption, which is much less computationally expensive. In this research we are primarily interested in benchmarking the performance of connection establishment, i.e., how many SSL/TLS transactions per second (TPS) can be achieved on the system.

To characterize the OCTEON III's SSL/TLS transaction performance, we performed two sets of tests:

1) An initial test to measure raw asymmetric cryptography performance. This test used a build of OpenSSL that leverages hardware acceleration on the OCTEON III in the OCTEON Linux environment. We ran the "openssl speed" command to collect the typical OpenSSL speed benchmarks for asymmetric cryptography performance for RSA and ECDH

operations. We ran as many parallel threads as there were processors on each respective machine (i.e., 48 for the OCTEON III, 36 for the x86 system).

2) An end-to-end test that initiated SSL/TLS connections with the OCTEON III. We ran a Simple Executive SSL server on the OCTEON III and established and closed connections as quickly as possible with it from a cluster of x86 hosts running ApacheBench [15]. We chose SSL/TLS configuration parameters for the connection that were reasonable for modern encrypted connections, but not necessarily the strongest or most modern available. We chose this configuration since in a real enterprise environment many clients and servers support older cipher suites. These measurements were conducted with a variety of cipher suites focusing on RSA and Elliptic Curve key exchange and authentication algorithms.

Results and Conclusions of SSLi Testing on OCTEON III

The raw asymmetric cryptography performance of the OCTEON III and the 36-core x86 system, as reported by OpenSSL, is shown in Figure 14. Across the board, the x86 system shows about an order-of-magnitude higher performance than the OCTEON III system (as noted by the vertical axis being logarithmic).

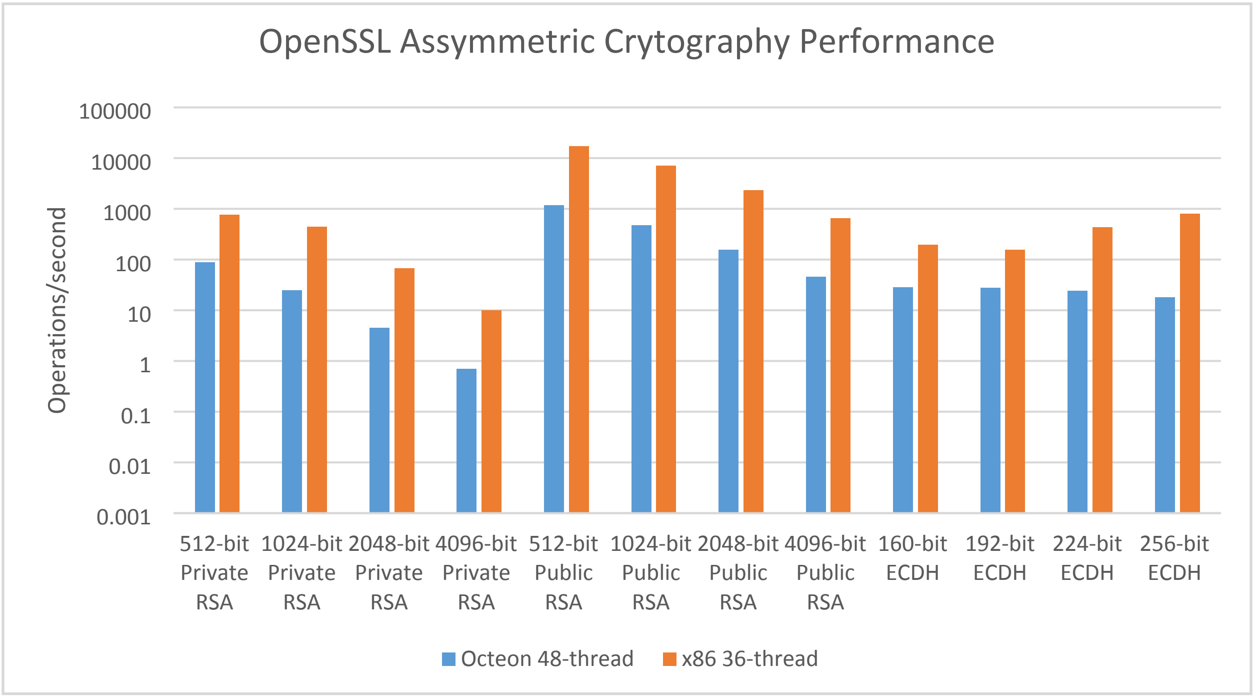

Figure 14. OpenSSL asymmetric cryptography comparison between the 48-core OCTEON III and the 36-core x86 systems.

In the end-to-end testing with ApacheBench, the OCTEON III's performance was most sensitive to the choice of digital signature algorithm, which is indicated by the second field of the cipher suite string. This divided the results into two clear classes of performance, as shown in Figure 15. The results show the cipher suites utilizing the Elliptic Curve Digital Signature Algorithm (ECDSA) operated at between 3500-5000 transactions per second, and those utilizing RSA operated at between 85000-90000 transactions per second. Within the cipher suites utilizing

ECDSA, the factor with the second largest impact was the choice between the Elliptic Curve Diffie-Hellman (ECDH) vs. Elliptic-Curve Diffie-Hellman Ephemeral (ECDHE) key agreement algorithm. Our testing showed performance differences of about 15% in favor of ECDH.

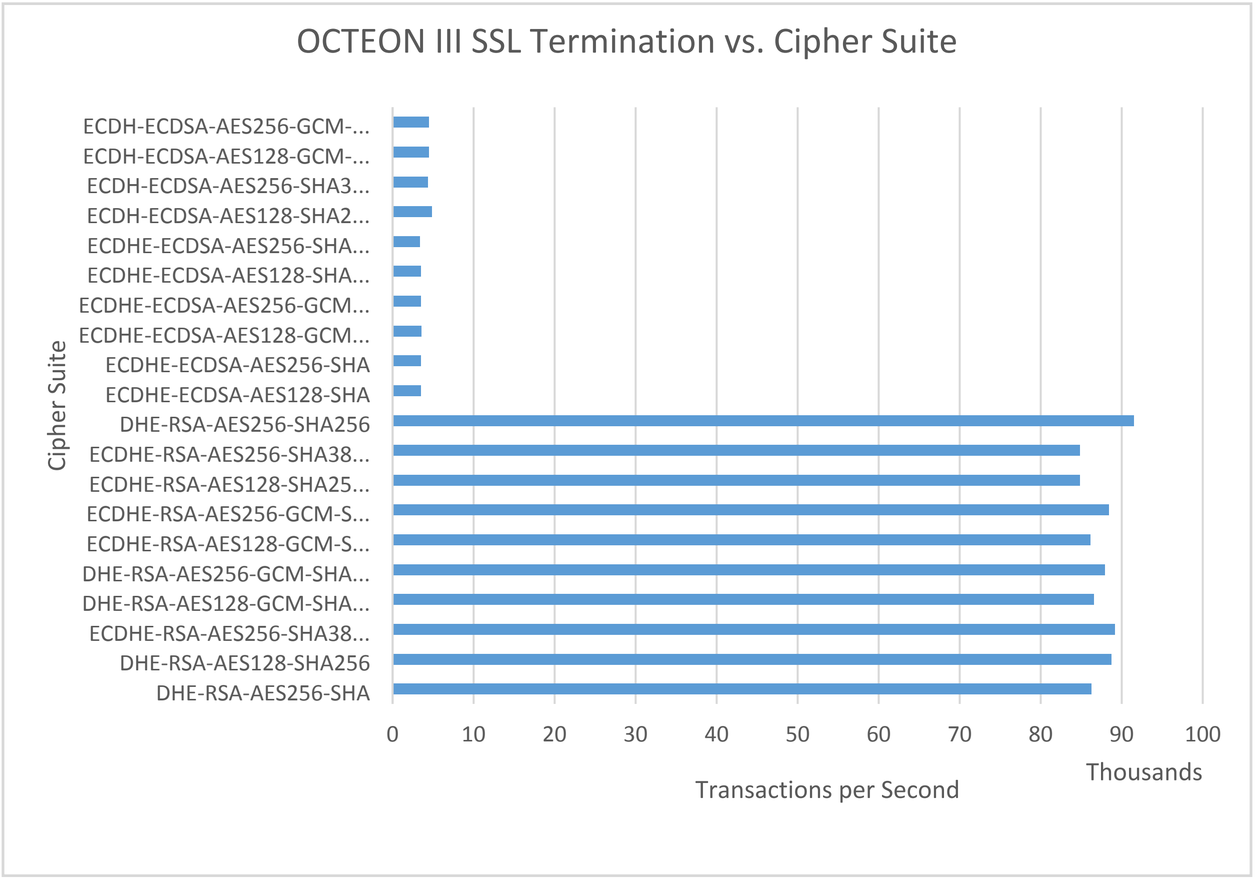

Figure 15. OCTEON III SSL Transactions per second with Simple Executive SSL Server for various cipher suites.

7. CONCLUSIONS

From the OCTEON III results, we conclude that the traffic analysis functions necessary for DPI are compute-bound. Our experiments and measurements showed the OCTEON III platform, which is designed for packet-switching and routing, was unable to perform DPI at line rate. Although the OCTEON III performs well at shuttling packets through the system at 40 Gbps, it did not have the necessary general-purpose compute performance to perform analysis on that traffic at line rate. A future platform utilizing the overall OCTEON III architecture, but with better performing general-purpose compute cores, may be sufficient and worth investigating in the future. However, it was determined that the hypothetical best-case DPI performance with Zeek was 15 Gbps, the cores would need to have its performance improve by several-fold.

From the x86-based hardware results, we conclude that a single x86 system can now provide enough compute density to enable 40 Gbps, and potentially 100 Gbps, line rate DPI traffic analysis. This conclusion is based on experiments using realistic enterprise traffic in a multi-socket system. However, our research showed the overall throughput was bottlenecked by the load-balancing throughput in certain scenarios (i.e., highly-imbalanced traffic and with high packet rates and small packets). These discoveries showed that, to achieve 100 Gbps DPI traffic analysis on x86-based systems, potential research and development should focus on more efficient design and implementation of load-balancing techniques. Candidate approaches could be through a mix of leveraging NIC hardware features and software, a hybrid software-hardware approach, or purely through more efficient and balanced load-balancing in software. This could include new hashing, queuing, and scheduling algorithms and more efficient synchronization mechanisms for interaction with the consumer threads. Our results motivate the need for novel architectural, algorithmic, and implementation work for network traffic load-balancing.

Additionally, advances in load-balancing at 100 Gbps on commodity hardware is applicable beyond DPI and traffic analysis to other domains that require high-speed load-balancing. Advances in load-balancers are applicable to other network security appliances, content distribution networks (CDN), cloud services, web application servers, software-defined networking, and other cost, power, and space-sensitive applications that depend on high-throughput packet processing on commodity hardware.

BIBLIOGRAPHY

[1] L. D. Carli, R. Sommer and S. Jha, "Beyond Pattern Matching: A Concurrency Model for Stateful Deep Packet Inspection," in *ACM Computer and Communications Security (CCS)*, 2014.

[2] Vincent Stoffer, Aashish Sharma and Jay Krous, "100G Intrusion Detection," Lawrence Berkeley National Laboratory, 2015.

[3] R. Sommer, V. Paxson and N. Weaver, "An Architecture for Exploiting Multi-core Processors to Parallelize Network Intrusion Prevention," *Concurrency and Computation: Practice & Experience - Multi-core Supported Network and System Security,* vol. 21, no. 10, pp. 1255-1279, 2009.

[4] B. Haagdorens, T. Vermeiren and M. Goossens, "Improving the Performance of Signature-based Network Intrusion Detection Sensors by Multi-threading," in *Proceedings of the 5th International Conference on Information Security Applications*, 2005.

[5] P. Enberg, A. Rao and S. Tarkoma, "I/O Is Faster Than the CPU: Let's Partition Resources and Eliminate (Most) OS Abstractions," in *Proceedings of the Workshop on Hot Topics in Operating Systems*, 2019.

[6] Mellanox Technologies Ltd., "Understanding PCIe Configuration for Maximum Performance," 20 February 2019. [Online]. Available: https://community.mellanox.com/s/article/understanding-pcie-configuration-for-maximum-performance. [Accessed 19 September 2019].

[7] Microway, "Performance Characteristics of Common Transports and Buses," [Online]. Available: https://www.microway.com/knowledge-center-articles/performance-characteristics-of-common-transports-buses/. [Accessed 19 September 2019].

[8] J. D. Gelas and I. Cutress, "Sizing Up Servers: Intel's Skylake-SP Xeon versus AMD's EPYC 7000 - The Server CPU Battle of the Decade? - Memory Subsystem: Bandwidth," AnandTech, 11 July 2017. [Online]. Available: https://www.anandtech.com/show/11544/intel-skylake-ep-vs-amd-epyc-7000-cpu-battle-of-the-decade/12. [Accessed 19 September 2019].

[9] Marvell Technology Group, Ltd., "OCTEON III CN7XXX Family of Multi-Core MIPS64 Processors," [Online]. Available: https://www.marvell.com/embedded-processors/infrastructure-processors/octeon-multi-core-mips64-processors/octeon-iii-cn7xxx/. [Accessed 19 September 2019].

[10] V. Paxson, "Bro: A System for Detecting Network Intruders in Real-time," *Computer Networks: The International Journal of Computer and Telecommunications Networking,* vol. 31, no. 23-24, pp. 2435-2463, 1999.

[11] L. Deri, "Improving Passive Packet Capture: Beyond Device Polling," in *Proceedings of the Fourth International System Administration and Network Engineering Conference (SANE 2004)*, 2004.

[12] L. Rizzo, "Netmap: A Novel Framework for Fast Packet I/O," in *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, 2012.

[13] J. M. Gonzalez, V. Paxson and N. Weaver, "Shunting: A Hardware/Software Architecture for Flexible, High-performance Network Intrusion Prevention," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007.

[14] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern and D. Miller, "The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, 2018.

[15] The Apache Software Foundation, "ab - Apache HTTP server benchmarking tool," [Online]. Available: https://httpd.apache.org/docs/2.4/programs/ab.html. [Accessed 19 September 2019].

DISTRIBUTION

1	MS9158	Jason Gao	8766
1	MS0813	Han Wei Lin	9315
1	MS0813	Vincent Urias	9315
1	MS0813	Brian Van Leeuwen	9315
1	MS0899	Technical Library	9536 (electronic copy)
1	MS0359	D. Chavez, LDRD Office	1171