The Core Cost of Doing Business

Don Wallwork, *Broadcom Corporation*
Andy Gospodarek, *Broadcom Corporation*

Abstract

Achieving line-rate reception or transmission of flows with the Linux kernel datapath often seems trivial with technologies like LRO, GRO, and TSO. However, when forwarding traffic using the Linux kernel datapath there is often a greater cost per packet to make sure it arrives at its intended destination -- whether that destination is a VM, container, or a remote system. This talk will cover in detail:

- the cost of the kernel datapath forwarding operation
- the typical cost of the destination lookup of a packet -- including how flow table state impacts lookup speed
- the impact of the size of the frames on link utilization

The goal will be to present a formula for estimating the minimum number of server cores that will be needed to service traffic at particular rates and frame sizes for IPv4 and IPv6 as well and VxLAN encapsulated/tunneled traffic with the goal of understanding how the utilization of these cores impacts planning and deployment of new systems.

1. The Rise of Open vSwitch

In the decade since the first release[1] Open vSwitch has become a popular tool used to help usher in Software Defined Networking (SDN) across datacenters. The network abstraction that it provides to datacenter operators makes it extremely attractive to those that want to easily define the location of services that may not always exist in the same physical location in a datacenter. Its inclusion in Linux Kernel version 3.3 was also a major milestone that would cause its popularity to grow.

In addition to adding support for the Linux kernel as a *datapath provider*, Open vSwitch added support for other two other significant datapaths to increate performance.

The DPDK framework was used to speed the processing of datapath frames and avoid the overhead of using the Linux kernel datapath when those datapath features (like NAT and Conntrack) were not needed. Later the DPDK datapath provider added support for those features in an effort to add feature parity between DPDK and kernel providers.

Another significant datapath provider added to Open vSwitch was the TC Flower datapath. This was intended to be an alternative to the Linux kernel datapath that could be easily offloaded to hardware on devices with such support.

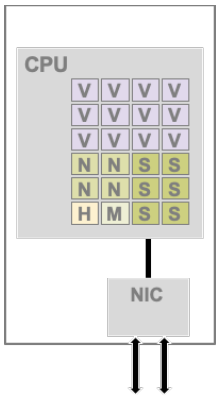
Another significant milestone for Open vSwitch its the inclusion into OpenStack. Inclusion in OpenStack which helped solidify Open vSwitch's place as a leader in the network virtualization space.

When setting up a server to use as a hypervisor an administrator often must dedicate cores to specific tasks. This assignment of resources (cores and memory) to specific tasks help ensure that there are enough resources available to service the users of that system. This typically comes in the form of processor, memory, or storage reservation. A typical server may have the following functions reserved or pinned to specific cores:

- VM Networking (N)
- VM Storage (S)
- Hypervisor Management (M)
- Host Networking (H)

with the remainder of the cores on the system could be used for VMs or Containers (V). A breakdown of what this could look like on a 24 core system appears in Figure 1:

***Figure 1**: Hypervisor Core Allocation*



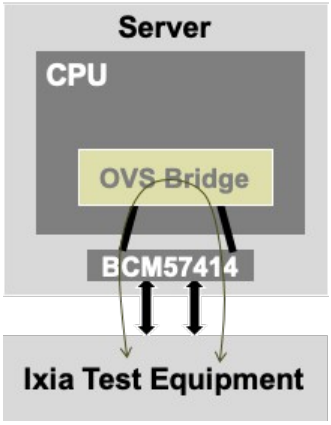
One can note that on this 24 core system only half of the cores are available to run workloads. This paper will examine the 4 cores being allocated to handle the network workload (N) generated by the applications (V) and whether or not there are cases where

2. Open vSwitch Performance

Quantifying the performance of Open vSwitch can be a difficult task. Not only are there multiple datapaths that can be used for traffic, but there are a host of different configuration options that can alter the performance in packets per second or in core utilization.

Based on customer deployment experience and reproducibility the traffic being forwarded in these initial tests was IPv4 UDP IMIX traffic (358 byte packets) that was expecting to be routed by the OvS bridge. There was no expectation of a firewall or NAT being used. The goal was to understand the best-case performance for routing traffic by OVS between two ports.

***Figure 2**: Test Server Configuration*



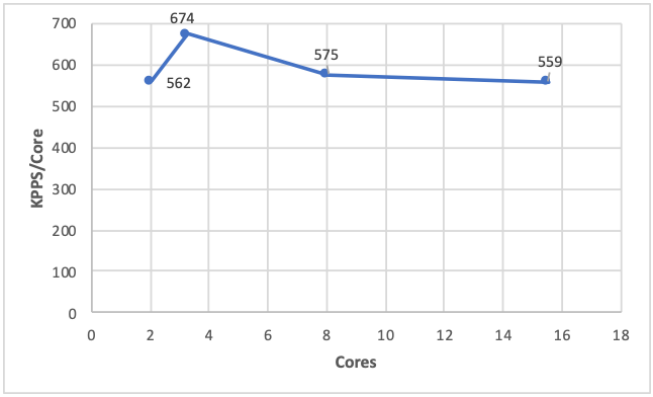
The server used was a Dell R730 with a dual socket Intel Xeon E5-2697v4@2.6GHz and a Broadcom BCM57414 (2x25Gbps) NIC. The system was tuned for maximum performance BIOS settings and IRQ affinity was set to the proper NUMA node for the NIC.

The software on the system was RHEL7.6 with 3.10.0-957.1.3.el7 and openvswitch-2.9.0-83.el7fdp.1.

2.1. Kernel Datapath

The first tests involved a testing the kernel datapath to determine how many packets per second a single core could handle sending and receiving. Once this was known the number of receive and transmit rings was altered to control how many different RSS queues would be handling the traffic received to understand how this scaled.

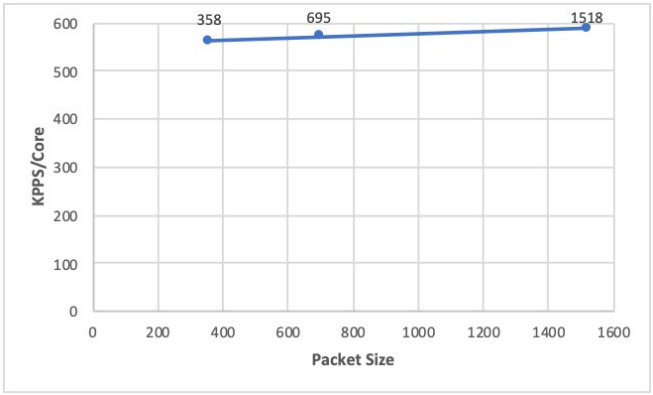
***Figure 3**: Received packets per second per core vs number of cores processing traffic*



The graph in Figure 3 shows a fairly flat line for the maximum number of packets per second a core can handle as the number of cores (and the traffic) is scaled up.

With a Xeon processor running at 2.5GHz, this graph tells us that we are able to handle 560kpps/core. This translates to 4600 cycles/packet or 2us. Based on this high value for cycles per packet there was an expectation that there would not be a difference in packet per second per core that could be processed as the packet size scaled towards MTU.

***Figure 4**: Received packets per second per core vs packet size of each packet*



Testing showed this hypothesis to be correct. This system can handle 560kpps/core across a variety of frame sizes and that means it consistently takes around 4600 cycles/packet.

These numbers provided a usable baseline for the performance in packets per second per core when using the OvS datapath available in the Linux kernel for known flows.

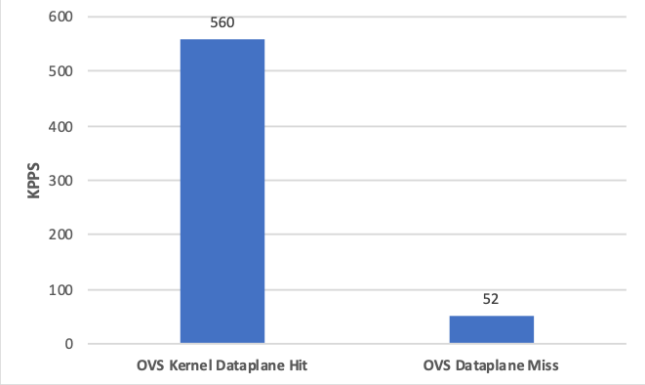
2.2. Userspace Datapath

The life of a packet when there is a miss in the kernel datapath is significanty different. An *upcall* is made to `ovs-`

`vswitchd` and flow tables stored there are queried to determine if a matching entry exists for that flow. The and the packet is then send to the proper outgoing interface by `ovs-vswitchd` using at matching rule or the default rule. The new rule is then written to the kernel datapath if needd.

This forwarding process is significantly less efficient than the kernel datapath largely due to the cost of the transition of the packet from kernel to userspace.

***Figure 5**: Received packets per second per core for kernel dataplane hit and miss cases*



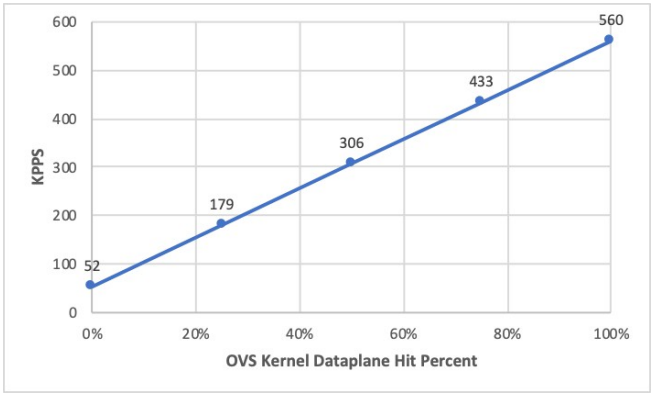
Testing showed that a kernel datapath miss would result in only being able to process 52kpps. This was a 10.8x decrease in performance over the kernel datapath hit performance. This also showed that a kernel miss would cost roughly 50,000 cyles per packet or 20us on the system used for testing.

2.2. Mix of Kernel and Userspace Datapath

Current and potential customers regularly come with expectations about flow setup and teardown rates as well as packet rates when forwarding traffic when building high performance datacenter networks. To better quantify what could be expected for new and existing flows, data was gathered to compute performance of the system when handling flows that have a hit in the kernel dataplane (or active flows) and close that miss the kernel dataplane (new flows).

Armed with the knowledge that a kernel datapath hit takes 4600 cycles and a kernel datapath miss takes 50,000 cycles, this graph was created to chart the relationship between the percentage of traffic that was a hit in the kernel datapath vs a miss in the kernel datapath.

***Figure 6**: Received packets per second per core based on OVS kernel dataplane hit percentage.*



If the last chart did not put into perspective the massive cost of misses to the kernel datapath this certainly does.

Putting this another way, at 75% hit rate for the kernel datapath per core there are only 13,000 new flows that are learned per second per core.

3. Options to Increase Performance

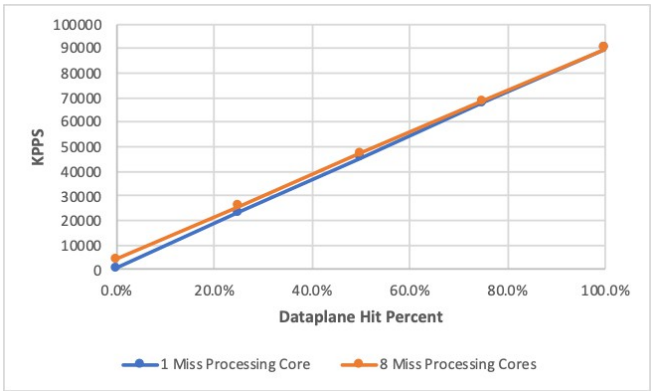
The first section of this document covered the fact that there is more than just a single datapath provider available for use with OvS. Both hardware acceleration via TC Flower offload and a PMD-based datapath were used to discover what gains were available there.

3.1. OvS with Hardware Acceleration

The hardware acceleration via TC Flower available from a variety of NIC vendors provides near-zero Xeon utilization when flows are completely offloaded to hardware. Unfortunately the *upcalls* to userspace that happen when trafffic does not match the hardware datapath are just as expensive as they are when the kernel datapath is missed.

This means that a graph of the throughput in packets per second has an identical slope to the kernel case.

***Figure 7**: Received packets per second per core based on hardware dataplane hit percentage.*

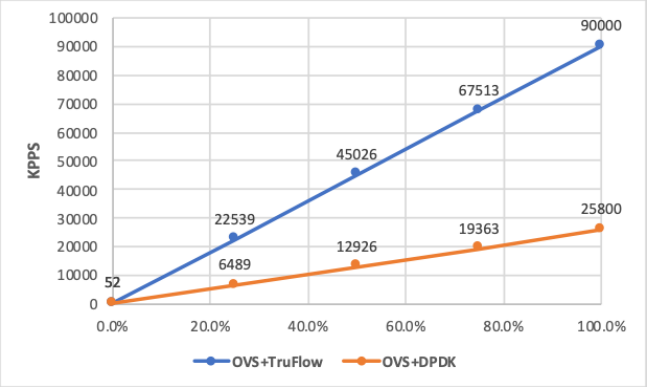


One would hope that using more cores (as many as 8) for processing the misses to the hardware datapath would make a demonstrable difference but it was not until we simulated a 10x performance boost in userspace processing did the line representing the *8 Miss Processing Cores* gain any separation from the line demonstrating 1 core performing miss processing of hardware offload. Miss rate is still the dominant factor in determining throughput – even when hardware offload is capable of handing 90Mpps with 100% hit rate.

3.2. OvS with DPDK

Though there was not much hope that OvS with DPDK as the datapath provider would provide any increased performance since the miss processing was still being performed by the same userspace code, tests were run to compare it to OvS with hardware acceleration.

***Figure 8**: Received packets per second per core based on DPDK dataplane hit percentage.*



As expected OvS + DPDK did not perform as well as the hardware acclerated dataplane. Performance was still *significnatly* better than the kernel datapath, but the cost of userspace handling of unknown flows still requires that an administrator allocate multiple Xeon cores to handle processing of packets.

Without a way to offload the processing of misses to the kernel datapath, hardware datapath, or DPDK datapath, there is not going to be a way to demonstrably reduce the cost of running Open vSwitch on a server.

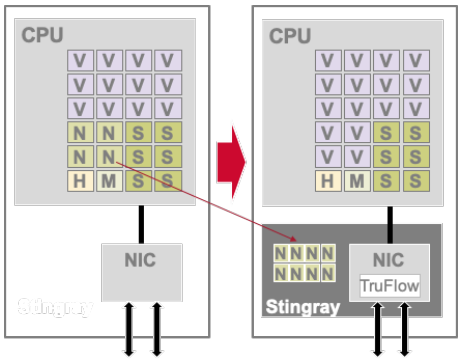
3.3. Full OvS Offload to Broadcom Stingray SmartNIC

An emerging trend in larger datacenters is to not only offload dataplane traffic to a NIC, but to offload control-plane (including dataplane misses) to the NIC itself. This is possible as new class of NIC have emerged that contain the same network acceleration capabilities as those NICs that can offload flows via the TC Flower datapath, but also have embedded processors capable of running a general purpose operating systems and applications.

In the case of the Broadcom Stingray SmartNIC this means the NIC itself has eight ARMv8 (A72) cores as well as storage for an operating system and memory that operate independently from the server. This *server in your server* runs Linux and has already demonstrated the ability to perform full offload of the OvS datapath and ovs-vswitchd at zero cost to the Xeon cores where the cards are installed.

This allows the transition of the network workload (N) from the Xeon cores to the ARMv8 cores. A visualization of this change appears in Figure 9.

***Figure 9**: Hypervisor core allocation before and after moving OvS from Xeon cores to SmartNIC ARM cores.*



Not only are 4 cores now freed from the need to process OvS traffic, but moving the workload to as many as 8 ARMv8 cores actually provides an increase in performance over 4 Xeon cores running a pure software datapath (OvS + DPDK) on the Xeon cores.

Since the Broadcom Stingray SmartNIC also runs a standard Linux kernel and tools, the ability to accelerate hardware datapath to the Broadcom TruFlow hardware from the Stingray control plane (ovs-switchd running on the ARMv8 cores) also exists.

Figure 11: Hypervisor core allocation before and after moving OvS from Xeon cores to SmartNIC ARM cores and then accelerating OvS dataplane to TruFlow hardware.

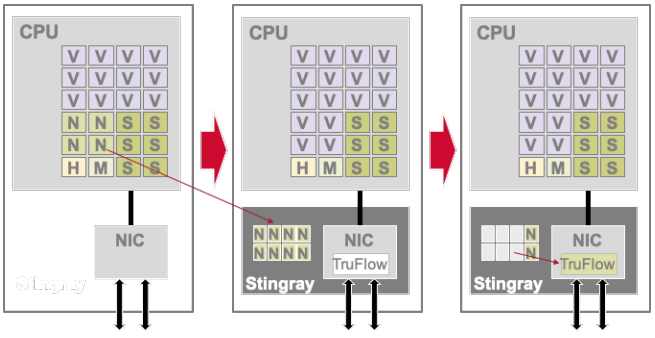
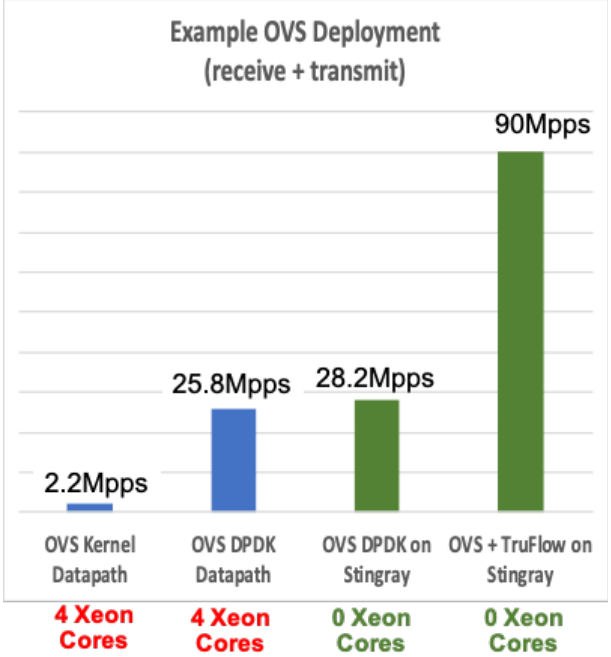


Figure 11 shows how this would free up the 8 ARM cores on the Stingray SmartNIC for doing nothing but processing datapath misses. The performance benefit of this case is the same as the benefit when accelerating the kernel datapath via TC flower on the Xeon cores, but now the Xeon cores are free from the burden of processing datapath misses and can be used for their intended workloads.

Figure 11: Comparison of OvS performance on Xeon Server and Stingray SmartNIC



The chart in Figure 10 shows (in blue) data presented in Figure 5 and Figure 8. This represents the throughput in PPS of OvS kernel datapath and OvS DPDK datapath in the 100% hit case. In green the chart shows the performance when the Stingray SmartNIC is used for running OvS with the DPDK datapath and OvS with acceleration via TruFlow in the 100% datapath hit rate.

3. Conclusions

The performance per watt of the ARMv8 processors combined with the high throughput between the ARM CPU subsystem and the networking interfaces has made this an attractive solution for full offload of OvS workloads from Xeon cores to the cores on a Broadcom Stingray SmartNIC. While SmartNICs may not be the solution of choice for everyone today this trend across large-scale datecenters is too important to ignore.

Acknowledgments

Thanks to Karen Schramm, Cedell Alexander, and John Carney for support during this project.

References

[TBD]