Performance Analysis of Intel DPDK on Physical and Virtual Machines

Vidhya Sankaran –vs444

Divyesh Darde- dsd96

Agenda

- Scope of Project
- Overview of Intel DPDK
- Setup and Configuration
- Demo on Physical and Virtual Machine
- Performance Measurements
- Analysis and improvements suggested

Team and Advisor Details

- Professor-

  Prof. Hakim Weatherspoon
- Advisor

  Han Wang
- Project Team

  Vidhya Sankaran –vs444

  Divyesh Darde- dsd96

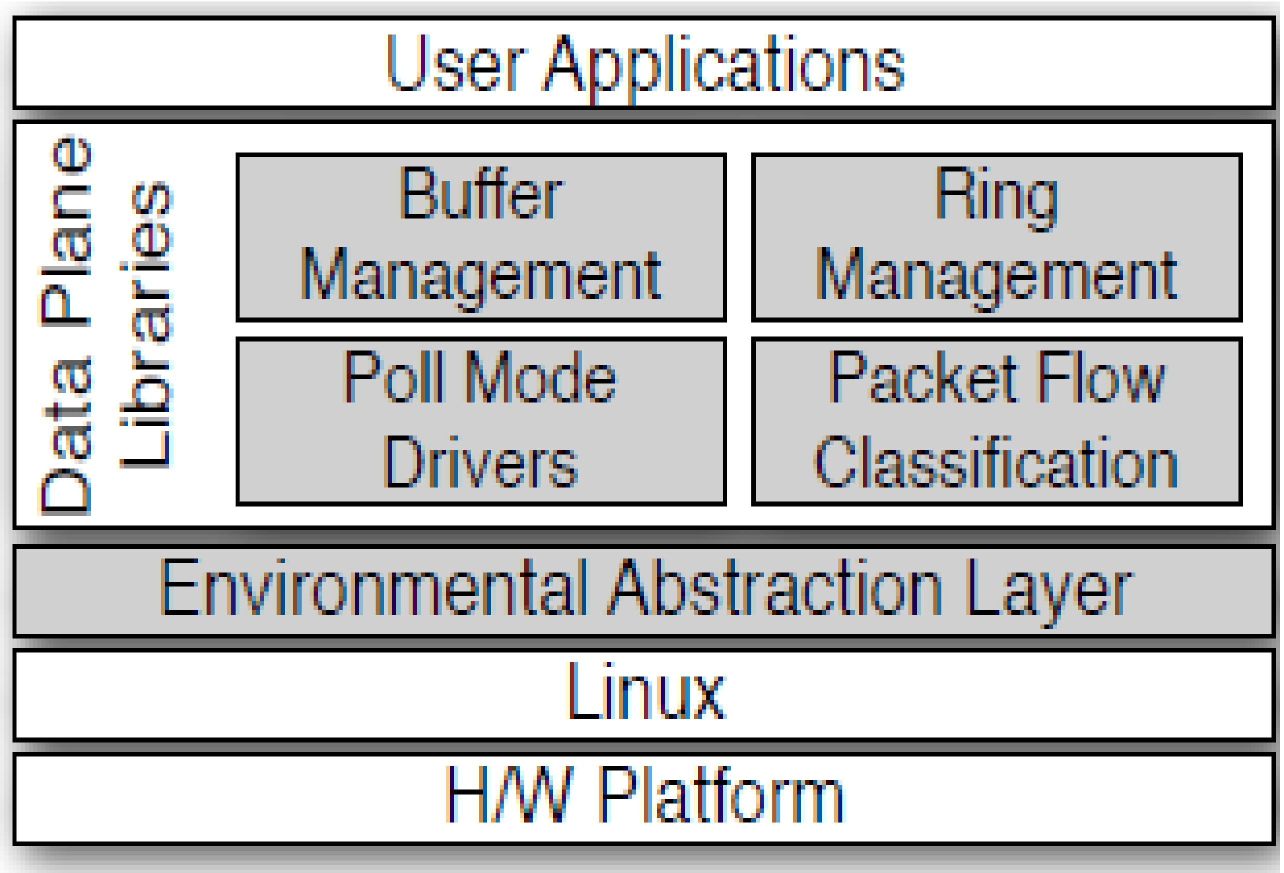
Scope

- Understanding the working of Intel DPDK
- Performance measurement of Intel DPDK with physical machines with and without Intel DPDK driver.
- Performance measurement of Intel DPDK with virtual machines
- Comparison and analysis of performance.

Intel DPDK Overview

- Is a complete framework for fast packet processing in data plane applications.
- Directly polls the data from the NIC.
- Does not use Interrupts- to prevent performance overheads.
- Another feature-Uses the huge pages to pre-allocate large regions of memory which allows the applications to DMA data directly into these pages
- DPDK also has its own buffer and ring management systems for handling sk_buffs efficiently.

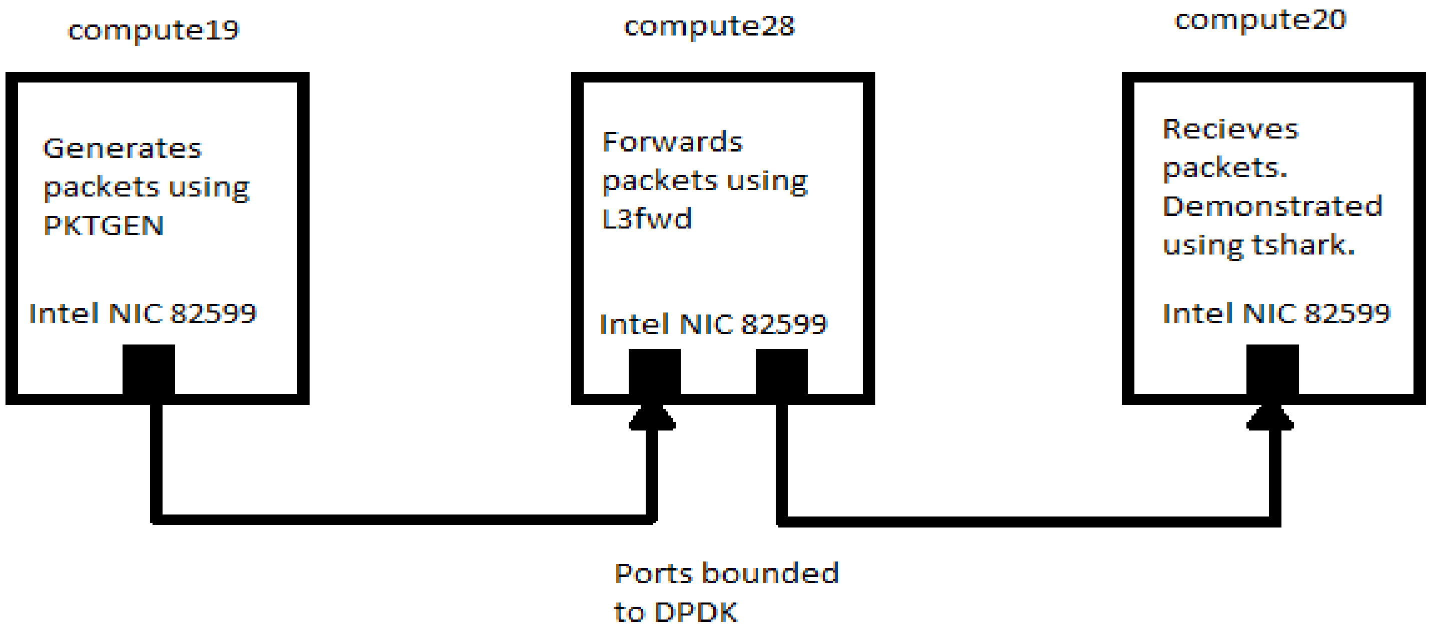
Core Components of DPDK



Core Components

- A queue manager implements lockless queues
- A buffer manager pre-allocates fixed size buffers
- A memory manager allocates pools of objects in memory and uses a ring to store free objects
- Poll modes drivers (PMD) avoid interrupt driven input, avoiding context switching

Setup and demo on Phy Machine



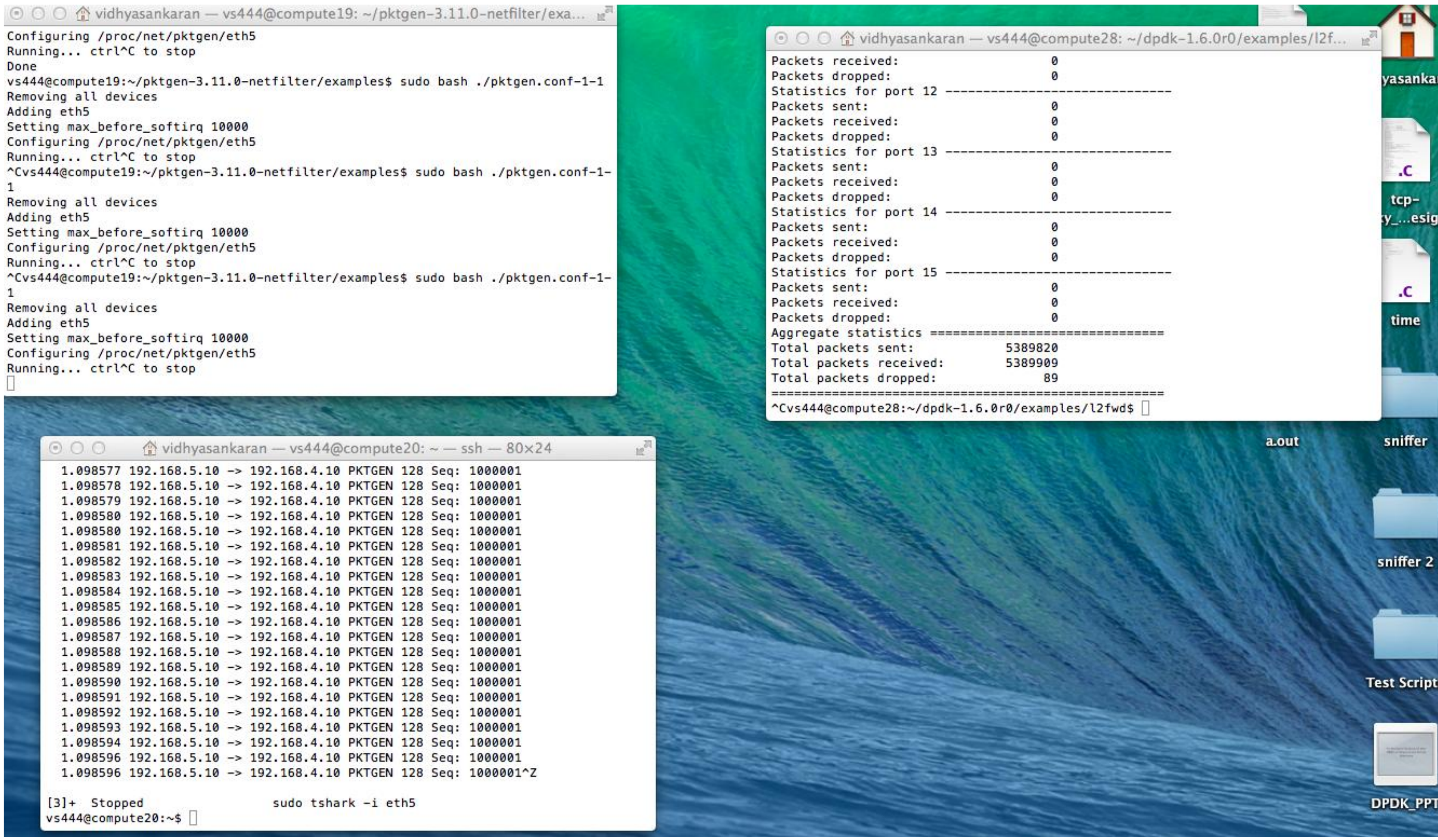
L3 Forwarder: We used L3 Fwder. The forwarding function uses a hash map for the flow classification. Hashing is used in combination with a flow table to map each input packet to its flow at runtime.

The hash lookup key is represented by a 5-tuple- Source IP, Dest IP, Source Port , Dest Port and Protocol

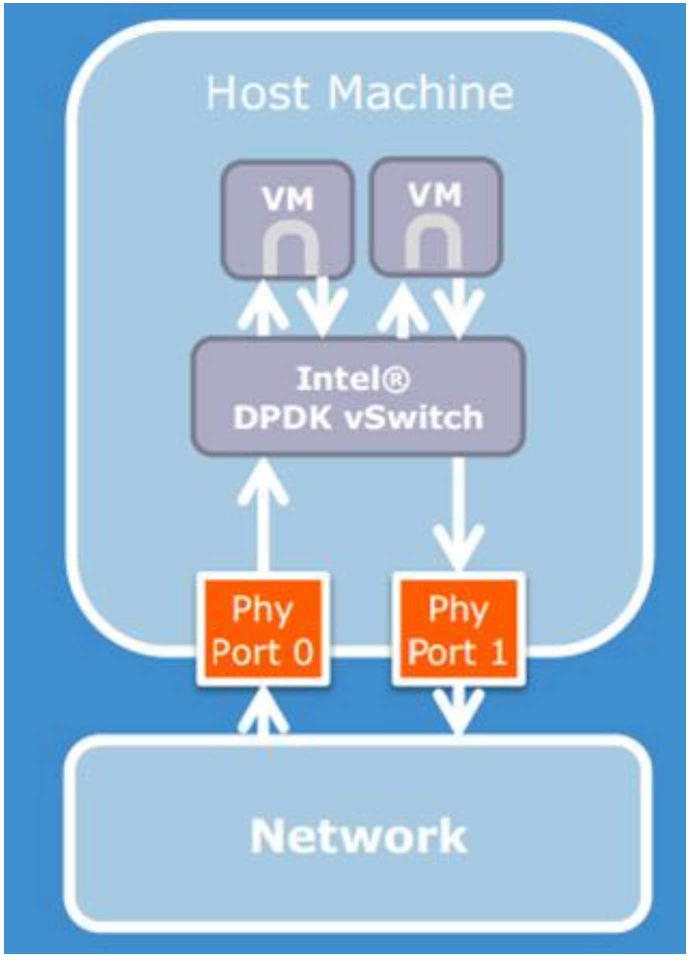
The ID of the output interface for the input packet is read from the identified flow table entry.

For the demo, the set of flows used by the application is statically configured and loaded into the hash at initialization time.

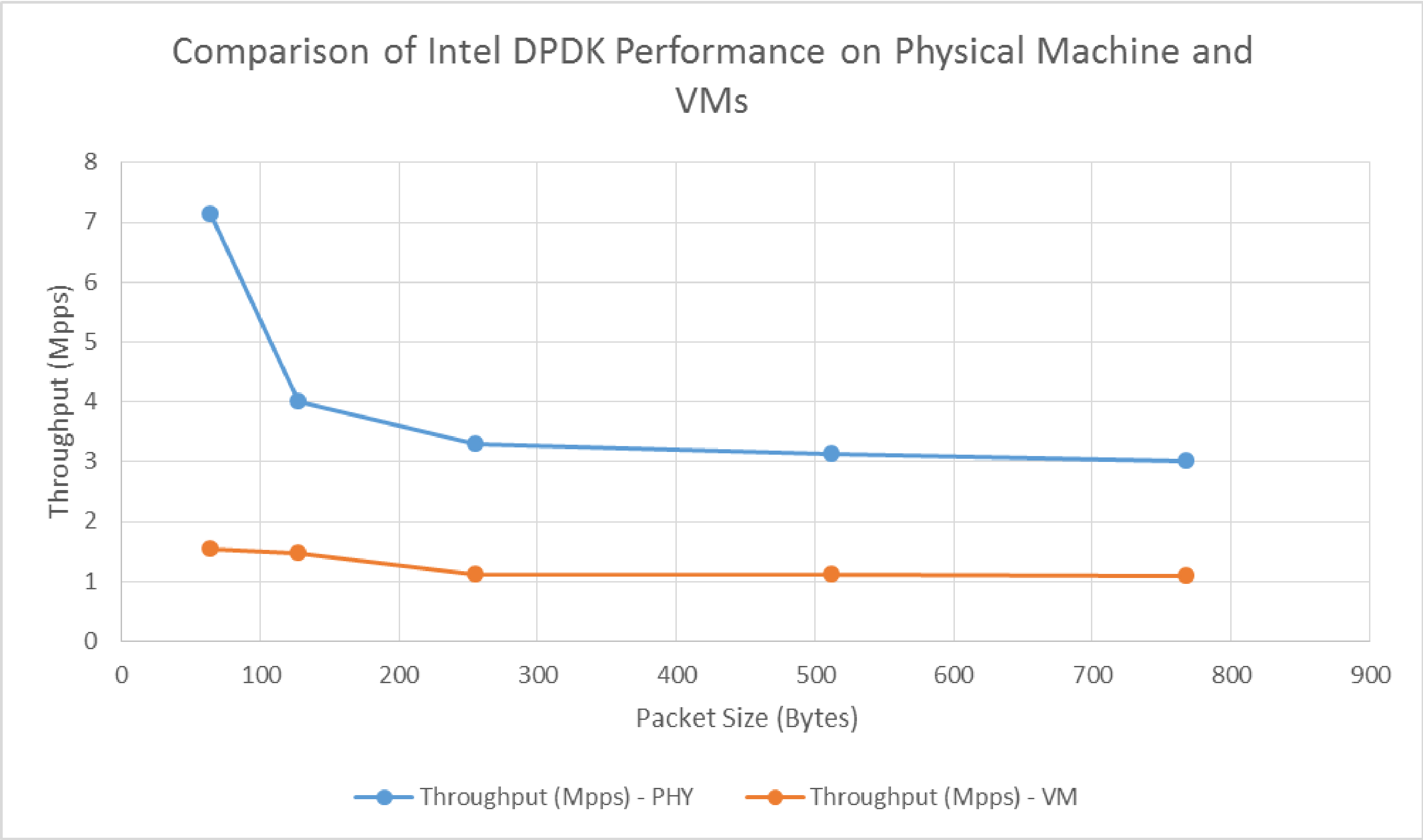
Demo


vidhyasankaran — vs444@compute19: ~/pktgen-3.11.0-netfilter/exa...
Configuring /proc/net/pktgen/eth5
Running... ctrl^C to stop
Done
vs444@compute19:~/pktgen-3.11.0-netfilter/examples$ sudo bash ./pktgen.conf-1-1
Removing all devices
Adding eth5
Setting max_before_softirq 10000
Configuring /proc/net/pktgen/eth5
Running... ctrl^C to stop
^Cvs444@compute19:~/pktgen-3.11.0-netfilter/examples$ sudo bash ./pktgen.conf-1-1
Removing all devices
Adding eth5
Setting max_before_softirq 10000
Configuring /proc/net/pktgen/eth5
Running... ctrl^C to stop
^Cvs444@compute19:~/pktgen-3.11.0-netfilter/examples$ sudo bash ./pktgen.conf-1-1
Removing all devices
Adding eth5
Setting max_before_softirq 10000
Configuring /proc/net/pktgen/eth5
Running... ctrl^C to stop
vidhyasankaran — vs444@compute28: ~/dpdk-1.6.0r0/examples/l2f...
Packets received: 0
Packets dropped: 0
Statistics for port 12 ------------------------------
Packets sent: 0
Packets received: 0
Packets dropped: 0
Statistics for port 13 ------------------------------
Packets sent: 0
Packets received: 0
Packets dropped: 0
Statistics for port 14 ------------------------------
Packets sent: 0
Packets received: 0
Packets dropped: 0
Statistics for port 15 ------------------------------
Packets sent: 0
Packets received: 0
Packets dropped: 0
Aggregate statistics ===============================
Total packets sent: 5389820
Total packets received: 5389909
Total packets dropped: 89
==
^Cvs444@compute28:~/dpdk-1.6.0r0/examples/l2fwd$
vidhyasankaran — vs444@compute20: ~ — ssh — 80×24
1.098577 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001
1.098578 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001
1.098579 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001
1.098580 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001
1.098580 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001
1.098581 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001
1.098582 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001
1.098583 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001
1.098584 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001
1.098585 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001
1.098586 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001
1.098587 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001
1.098588 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001
1.098589 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001
1.098590 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001
1.098591 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001
1.098592 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001
1.098593 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001
1.098594 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001
1.098596 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001
1.098596 192.168.5.10 -> 192.168.4.10 PKTGEN 128 Seq: 1000001^Z
[3]+ Stopped sudo tshark -i eth5
vs444@compute20:~$
a.out
sniffer
sniffer 2
Test Scripts
DPDK_PPT
time

Setup and Demo on VM Machine



Performance Comparison(With Intel DPDK using PKTGEN and L3 fwder in PHY and VM)



Why this performance gain?

- ✓ Put/get data directly to/from dma area to avoid data copies.
- ✓ Work in poll mode. The hardware do not need to generate interrupt for packet reception. Interrupt also causes context switch.
- ✓ Uses huge pages to hold the packet data to avoid the minor page faults.

Problem with VMs and Improvements suggested

- Inter VM communication will still require additional copy of data between the VMs.
- Overheads caused by inter-core communication and context switching.

Improvements suggested-

- Can use a shared memory framework that provides zero copy delivery to VMs and between VMs.[NetVM paper uses this]
- Support for high-speed inter-VM communication through shared huge pages

Questions