Vectorized Packet Processing

Thesis

Submitted in partial fulfillment of the requirements of
BITS F 423T/BITS F421T, Thesis

by

Addanki V L S S Vamsi Krishna
2013A8PS790G

Under the supervision of
Prof. Dario Rossi

BITS, PILANI –K K BIRLA GOA CAMPUS

Date: 10/05/2017

ACKNOWLEDGEMENT

I would like to thank Prof. Dario Rossi for giving me an opportunity to work on this project. I would also like to thank Leonardo Linguaglossa, postdoc at Telecom-Paristech for all the help. I've learnt a lot from him.

CERTIFICATE

This is to certify that the Thesis entitled, ________________________________

____

is submitted by ____________________________ ID No .________________

in partial fulfillment of the requirements of BITS F 423T/BITS F421T Thesis embodies the work done by him/ her under my supervision.

Signature of the supervisor

Name

Designation

Date____________

List of Symbols & Abbreviations used in this Report

VPP	- Vectorized Packet Processing (approach) / Vector Packet Processor (Engine)
DPDK	- Data plane Development Kit
Pktgen	- Packet Generator
FD.io	- Fast Data Input Output
NIC	- Network Interface Card
UCS	- Unified Computing System
CIMC	- Cisco Integrated Management Controller
DCE	- Data Center Ethernet
KVM	- Keyboard,Video,Mouse
IP	- Internet Protocol
XC	- Cross Connect
I/O	- Input/Output
DMA	- Dynamic Memory Access
vSwitch	- Virtual Switch
hw	- Hardware
sw	- Software

ABSTRACT

In the last few years, numerous frameworks have emerged which implement network packet processing in user-space using kernel bypass techniques, high-speed software data plane functionalities on commodity hardware. VPP [3] is one such framework which gained popularity recently for some of the interesting points like its flexibility, user-space implementation, kernel bypass techniques etc. VPP allows users to arrange or rearrange the network functions as a packet processing graph, providing a Full-blown stack of network functions. Unlike the other frameworks in which packet processing is done packet-by-packet, VPP performs packet processing, vector-by-vector i.e a batch of packets at once. This brings a significant performance benefits due to L1 cache hit.
In this report I discuss an introduction to Vector Packet Processor, a framework by Cisco which is now a part of the Fast Data Input Output (FD.io) project [2]. Later in this report I will discuss about the initial experiments on VPP, the test bed used for performing the experiments with VPP and how to set up the test bed. The variation of three values (Throughput, Average Vector size, CPU clock cycles) are observed while changing Maximum vector size. The result clearly show that cache-hit ratio increases from vector size of 4 to 256 where it is maximum and decreases.I begin with related work about packet processing to understand why VPP is important in the high speed networking world.

Keywords : VPP, kernel-bypass, processing graph, node, vector, cache.

Table of Contents

ACKNOWLEDGEMENT....2
List of Symbols & Abbreviations used in this Report....4
ABSTRACT....5
Chapter 1 Introduction....7
1. Kernel Bypass Networks....7
1.1 Literature....7
1.2 Kernel Bypass Solution....7
2. Kernel Bypass Techniques....7
3. Vector Packet Processor (VPP)....9
3.1 Introduction to Vector Packet Processor....9
3.2 VPP Graph....10
3.3 VPP Plugin....10
Chapter 2 Testbed....12
1. Introduction....12
2. Virtual Machine Testbed....12
2.1 Cisco Rack Mount Server....12
2.2 vSphere....13
3. Bare Metal Testbed....14
4. Setup....14
5. List of Experiments Performed....14
6. Observation....15
Chapter 3 Experimental Results....16
1. Throughput vs Maximum Vector Size....16
1.1 Bare Metal Configuration....16
1.2 Virtual Machine Configuration....17
2. Average Vector Size vs Maximum Vector size....18
2.1 Bare Metal Configuration....18
2.2 Virtual Machine Configuration....19
3. Average Clock Cycles per vector vs Maximum Vector Size....20
Conclusion....22
References....23

Chapter 1
Introduction

1. Kernel Bypass Networks

1.1 Literature

In the last few years, there was a shift from hardware network devices to software functionalities. Although hardware components are fast at processing, they lack flexibility to adopt new functions. They were only restricted to a specific set of functions. A complete change of hardware was required to add new functionalities. So researchers have put a lot of effort into optimizing performance of software processing stack which led to the development of several frameworks such as Netmap [6], DPDK [1], Click [7], Fast-Click etc. In the beginning of the emergence of the software packet processors, most of the frameworks were implemented in such a way that all the high speed functionalities used to be placed close to hardware as a separate kernel module. This added an over-head to packet processing since user applications need to execute system-calls to access the kernel module.

1.2 Kernel Bypass Solution

In the last decade, we observed emerging techniques implementing kernel bypass. DPDK is a good example for this model. Vector Packet Processor (VPP) is the latest framework which implements kernel bypass model for high speed packet processing. Kernel Bypass techniques are discussed in the next section.

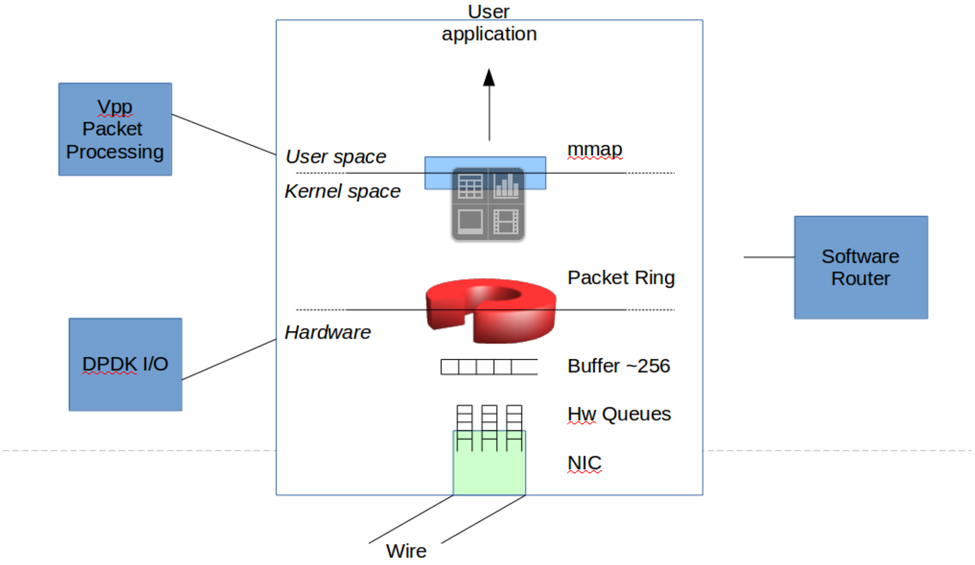
2. Kernel Bypass Techniques

In order to reduce the overhead of system calls and to allow the access to the network device hardware at the user-space level, a technique called Kernel bypass is used. This is a technique Some of the important kernel bypass techniques implemented by Kbnets (Kernel bypass networks) are i) I/O Batching ii) Zero Copy iii) Multi-Threading

I/O Batching : When the network interface of a router receives any packets, they are written to one of the hardware queues of the NIC. Hardware queues are implemented at NIC hardware level. As soon as a packet is ready to be scheduled for processing, an interrupt is issued to the operating system which stops all the running processes and runs the interrupt routine. I/O batching is a technique to group packets in a separate buffer and send an interrupt once the buffer is full. This reduces the overhead of interrupt for each packet by sending an interrupt only after a batch of packets are ready to be processed. This can speed-up the overall processing.

Zero Copy : When the network interface receives any packets , they are written to a shared memory region which is shared between network interface card and the operating system. In order to access this memory, the operating system needs to issue system calls for memory copy operation. Although this was the old-fashioned approach, nowadays software solutions tends to avoid a memory copy between the kernel and the user-space: this approach is called Zero-copy. In the Zero-copy approach, the NIC simply writes the packets, through Direct Memory Access (DMA) to a specific memory area, called the packet ring, which is a circular buffer of memory accessible by both the network device and the user- space application. Some locking mechanisms may be required to prevent multiple writes on the same portion of memory.

Multi-Threading : CPUs with multiple processors and cores are currently widely available. Multi-threading features allow to increase parallelism of network applications. This is tightly coupled with the availability of hardware queues but even when a single receive queue is available, the software scheduler can assign different packets to different cores, and then perform independent processing. The parallelism degree of a network application, which represents the number of ”threads” simultaneously running, is related to a speed-up in the performance of the system: the more threads available, the better the performance, up to a saturation point where increasing the number of threads does not affect the performance.



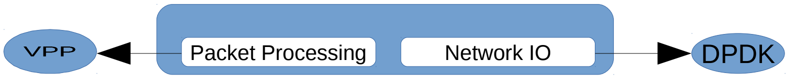
[Kernel bypass architecture]
Fig:1.2.1

3. Vector Packet Processor (VPP)

3.1 Introduction to Vector Packet Processor

VPP [3] is one amongst the latest framework for high speed packet processing. It runs completely in user space and implements kernel bypass techniques to access hardware. Its design is hardware, kernel and deployment agnostic. As opposed to the old packet by packet processing methods, VPP processes a vector of packets at once. So it is vector by vector processing. I will discuss more about vectors later in the report.

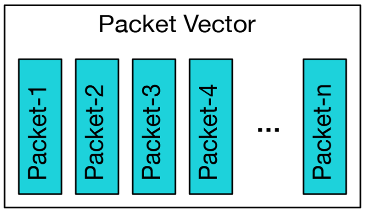
VPP uses the services of DPDK [1] for kernel bypass and accessing the hardware. DPDK package is an integrated part of VPP package. Network I/O is implemented by DPDK and the packet processing is implemented by VPP application.



[VPP router model]
Fig:1.3.1

VPP makes use of DPDK to fetch a vector of packet from the Network card, and then it processes all packets in a vector. Vector is a group of packets which are processed at once. Current Vector size is 256.

VPP only copies the pointers to the packets from the packet ring shown in the figure *[Kernel bypass architecture] Fig:1.2.1*. VPP creates its own buffers of 256 indexes which point to the packet ring and forwards this buffer of packet indexes in the VPP graph defined by the fib.



[vector]
Fig:1.3.2

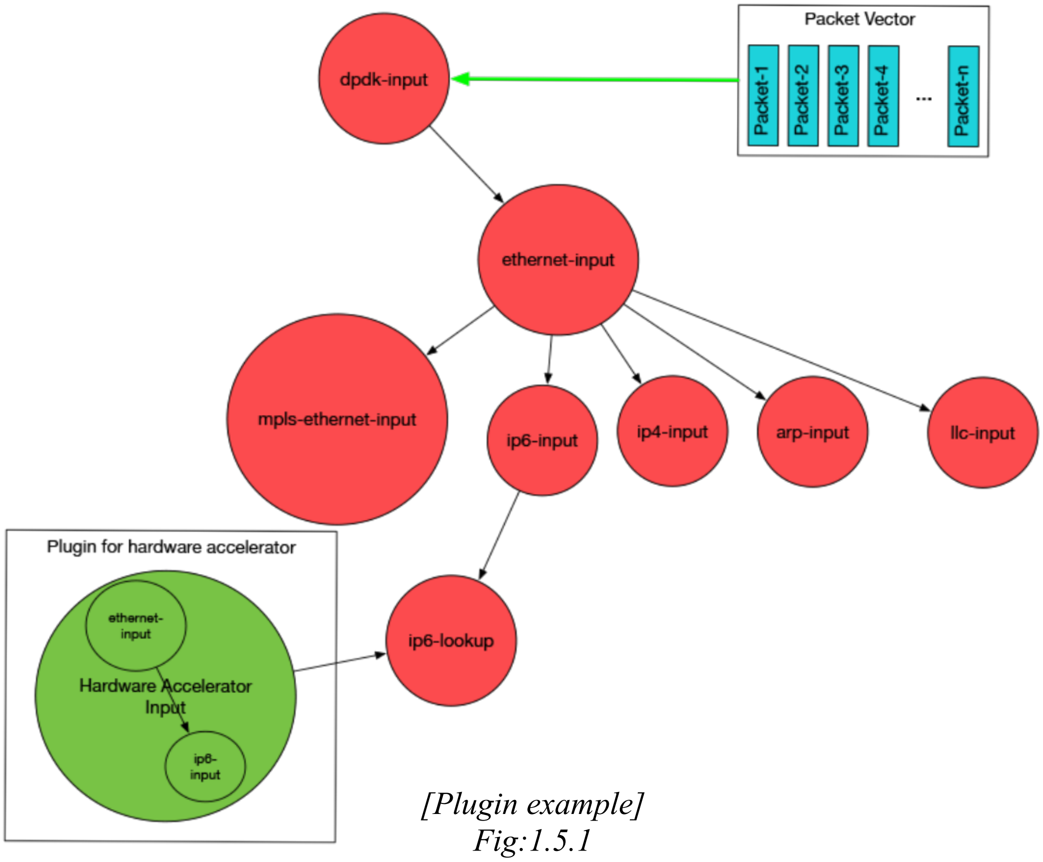
3.2 VPP Graph

Once the buffer of packets (one vector) is available, VPP starts processing all the packets in the vector. VPP processing is implemented as a forwarding graph. Each node represents a network function, and is implemented as a function in C language , for example ipv4-lookup, dpdk-input, interface-output etc. There are 4 types of nodes.
1) Pre-input   2) Input   3) Internal   4) Process

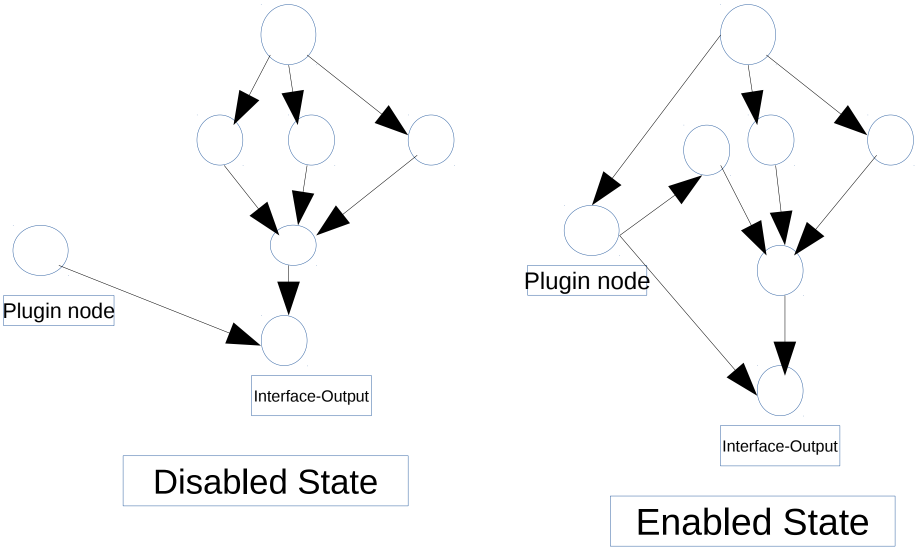
Pre-input nodes are the functions which handle the packet fetching from hardware and make the buffer of indexes available to VPP. Input node may be in two states. 1.Interrupt 2. Polling . The input nodes are in Interrupt mode when the number of packets available in the buffer are less than 5. When there are more than 10 packets available, the node changes from Interrupt mode to polling mode. In the polling mode, the node continuously keeps checking if there are any packets available to fill the vector and to process. Process nodes are separate nodes which are not connected to the graph. They run separately as threads.
Each node predicts the next node after processing the vector and sends the packets to different next nodes based on its decision. This is called ***Branch prediction.***

3.3 VPP Plugin

VPP allows user to modify the graph traversal. Custom nodes can be inserted in the graph and can be connected to desired nodes. Flexibility is a huge advantage of software routers like VPP. A simple example for plugin is hardware acceleration. Some of the graph nodes can be replaced by a hardware which implements the same functions. An input node and output node can be created for the hardware which connects to the graph. If the hardware is present, the previous node can be programmed to send packets to the input node of hardware and when the hardware is failed, the graph still performs normally in default mode.



[Plugin example]
Fig:1.5.1

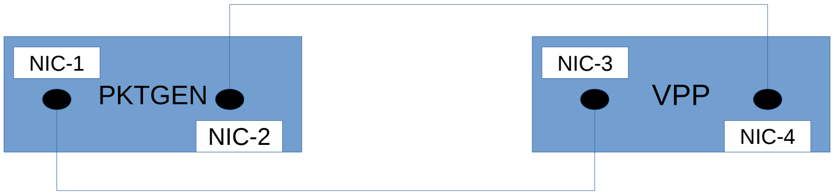


[Plugin Enable/Disable]
Fig:1.5.2

Chapter 2
Testbed

1. Introduction

There was never a proper explanation for why 256 is the number of packets contained by a vector for VPP. So, the initial set of experiments are to evaluate VPP performance by changing the vector size. The vector size is varied from 4 to 2048 in the steps of 2. We planned to perform these experiments in two types of environments. 1) Bare Metal 2) Virtual Machine. The basic idea is to have 4 network interfaces which are connected as shown in the figure below with VPP connected to two of them and a packet generator on the other two with one port transmitting and the other receiving.



[Experiments Hardware Setup]
Fig:2.1.1

2. Virtual Machine Testbed

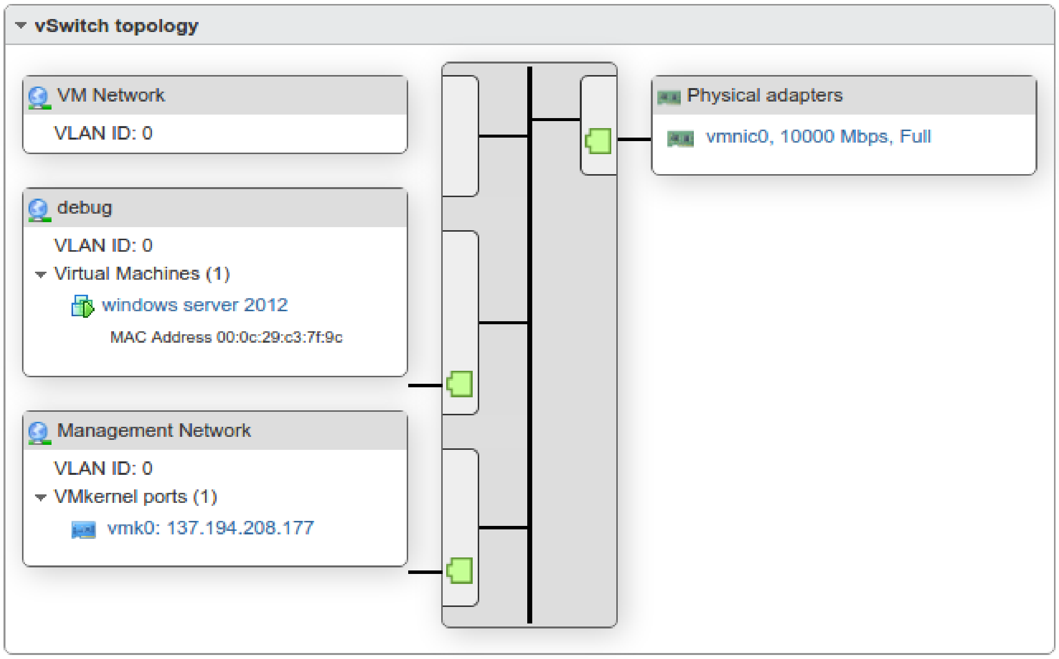
2.1 Cisco Rack Mount Server

Cisco rack mount blade servers [8] are equipped with 2× Intel Xeon Processor E52690, each with 12 physical cores running at 2.60 GHz in hyper-threading and 576KB (30M) L1 (L3) cache. 192 GB of Primary memory, 2.7 TB of secondary memory, 2 DCE interfaces which manage the traffic flowing to and fro from the server, 2 network interfaces Intel x520 with dual port full-duplex 10Gbps links. Each server port is connected to a fabric interconnect which has several network ports. Cisco UCS manger [9] is an application to manage these server. Once UCS manager is opened, each server needs to be associated with a service profile which defines the resources used by the server. DCE interfaces can be used only by creating virtual network interfaces under the DCE interface. Each vnic is configured with proper vlan id to access network. Once the service profile is associated with the server an IP address is assigned to both server and the service profile. KVM manger is used to view the server monitor, pass input from keyboard and mouse. In order to install a virtual machine on the server, VmWare ESXi

[10] has to be installed first. After installing ESXi, the ESXi host has to be assigned with a network interface. Once the host gets a valid IP address ESXi host can be managed by using the IP address from vSphere web-client or vSphere desktop application.

2.2 vSphere

VSphere [10] is used to manage the ESXi host and the virtual machines installed on the ESXi host. Using the IP address of the ESXi host a web client can be opened from a web browser. There are few steps to create a virtual machine with desired network interfaces. 1) Create a virtual machine 2) Edit setting to manage resources of the server by allocating required number of cpus, memory, network interfaces etc. 3) Configuring network ports and virtual switches. Each physical interface of the server visible in UCS manager is connected to vswitch as an uplink and virtual network interfaces are attached to the vswitch on the other side which can be used by a virtual machine. An example of the vswitch topology is shown below. In the figure Vmkernel ports are those which can be used to access the ESXi host for management purposes.



vSwitch topology

VM Network
VLAN ID: 0

debug
VLAN ID: 0
Virtual Machines (1)
windows server 2012
MAC Address 00:0c:29:c3:7f:9c

Management Network
VLAN ID: 0
VMkernel ports (1)
vmk0: 137.194.208.177

Physical adapters
vmnic0, 10000 Mbps, Full

[vSphere vswitch topology]
Fig:2.2.2.1

In this way, 4 network interfaces, two each are on the same network port group connected to two vswitches, are made available to the Ubuntu virtual machine.

3. Bare Metal Testbed

Bare Metal setup is almost the same as explained in the previous section. After creating a service profile in UCS Manager, Ubuntu can be installed directly on the server using an iso image of Ubuntu and VPP can be installed on the operating system. Four NICS are cross-connected by physical wires. Cisco Rack Mount server has 2 network interfaces Intel x520 with dual port full-duplex 10Gbps links. The network interfaces are connected by SFP+ interfaces.

4. Setup

Dpdk Pktgen [4] is used for generating packets. Packet generator is run on the NIC-1 and NIC-2 (fig : 3.0) VPP is connected to NIC-3 and NIC-4 (fig : 3.0) Packets are sent out of NIC-1 and received at NIC-3 connected to VPP. A static ARP entry is added to VPP for NIC-4 to transmit packets to NIC-2. Different configurations of VPP can be used for performance evaluations. 1) xconnect mode : Interfaces are in layer 2 mode and all the packets received by one interface is sent out of other interface which is xconnected to it. 2) Layer 3 mode : Interfaces are in layer 3 mode. IP address, IP route entries needs to be configured to forward packets

5. List of Experiments Performed

1. Variation of throughput with variation in maximum vector size. [VLIB_FRAME_SIZE]
2. Variation of average vector size per node with variation in VLIB_FRAME_SIZE.
3. Variation of average CPU clock cycles per node with variation in VLIB_FRAME_SIZE.

Types of modes in which VPP is tested

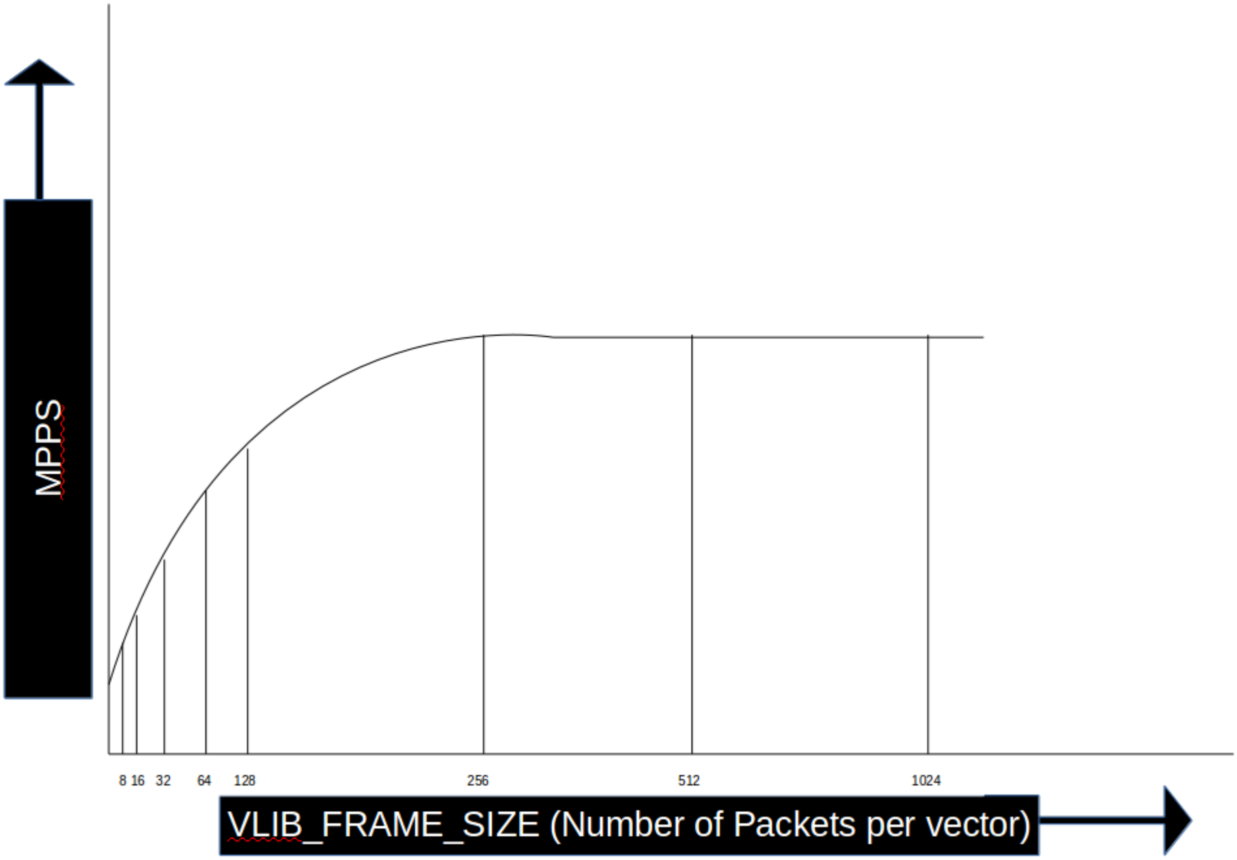
1. X-Connect
2. IP forwarding

Types of traffic generated

1. Static
2. Round-Robin
3. Uniform

6. Observation

At lower vector size the I/O is faster than the processing and packets are dropped by VPP. When the frame size is around 256, there is a cache hit and hence there is a huge gain in packet per sec processed by VPP. After 256, when the frame size increases a lot, cache is full and large amounts of primary memory is used. So there is a drop in the packets per sec processes by VPP. So it is reasonable to predict a graph as shown in the figure below.

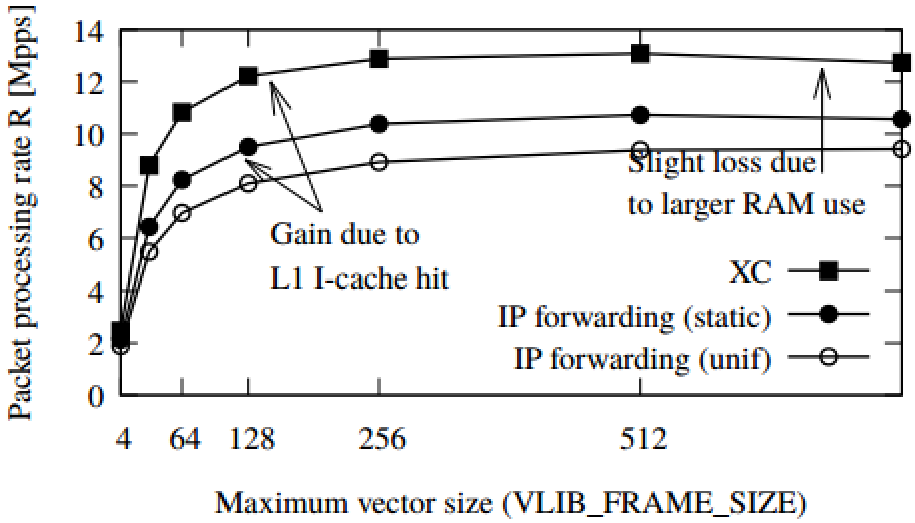


[expected graph mpps vs vlib_frame_size]
Fig:2.6.1

Chapter 3
Experimental Results

1. Throughput vs Maximum Vector Size

1.1 Bare Metal Configuration

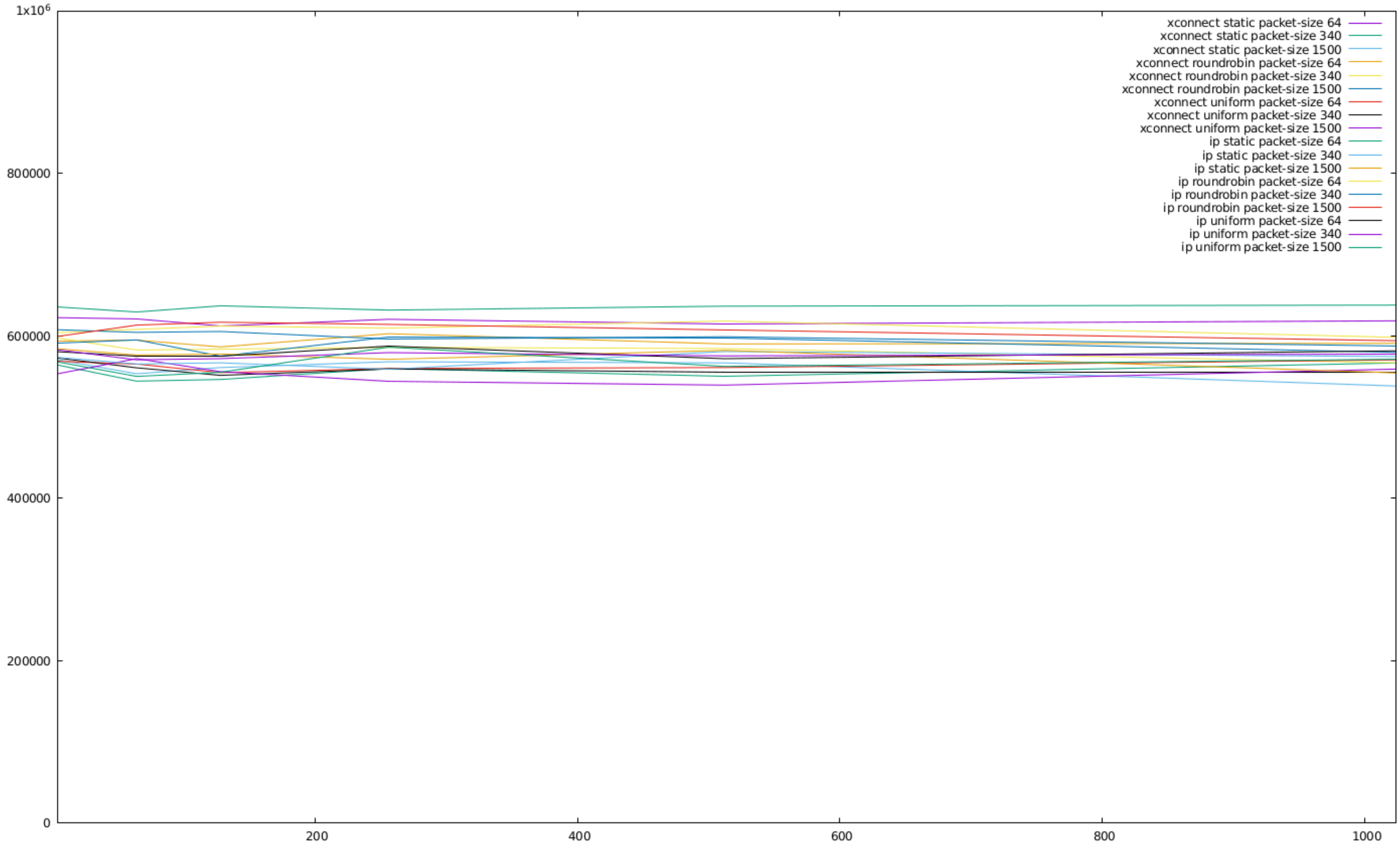


[Mpps vs maximum vector size in Bare Metal]
Fig:3.1.1.1

This behavior is expected. There is a gain when maximum vector size is increased in the beginning due to cache hit. After 256, the vector size becomes big enough and there are cache misses. As the vector size increases more, excessive RAM is used and the performance drops slightly.

In the x-connect mode all the packets received from one interface are sent out of the cross connected interface and hence there are only 2 active nodes in the processing graph. Therefore, we observe the highest throughput in xconnect mode. Where as in the IP forwarding case, all the network functions related to IP forwarding (ex: ip-lookup) are active in the processing graph. So the throughput drops to approximately 10mpps maximum in the IP forwarding mode.

1.2 Virtual Machine Configuration



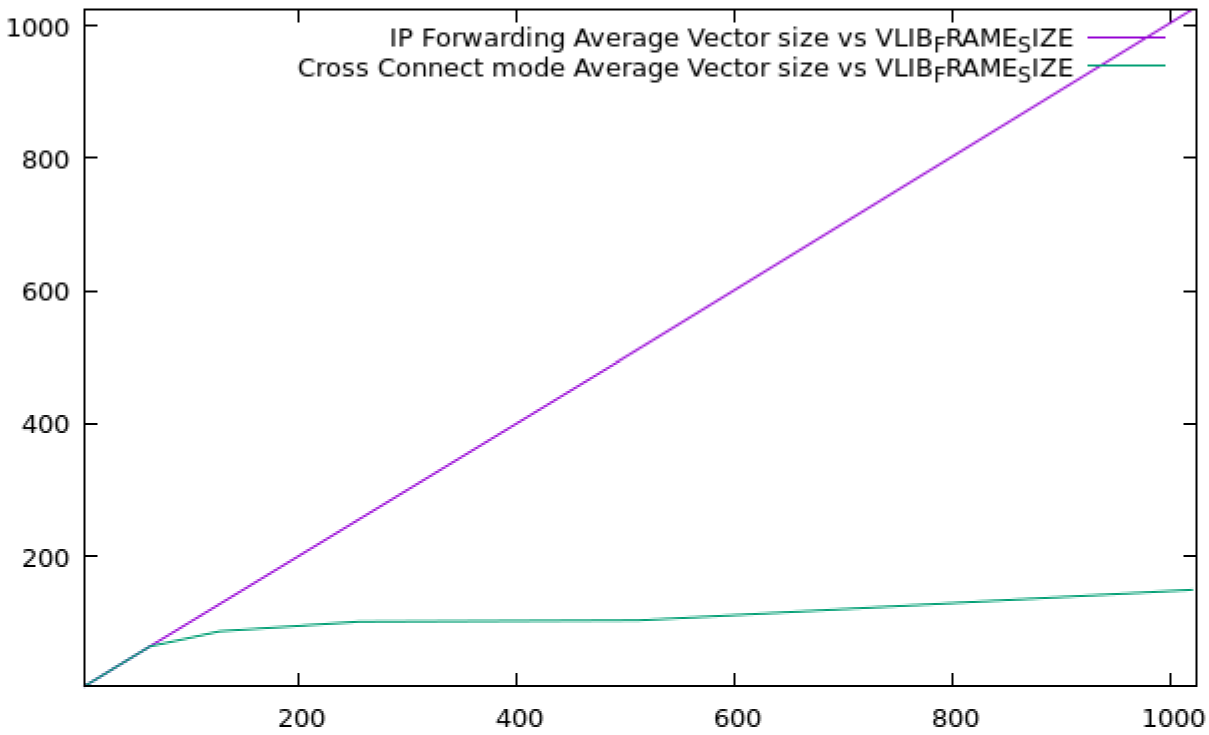
[Mpps vs maximum vector size in Virtual Machine]
[y-axis vs x-axis]

Fig:3.1.1.2

In this case the throughput remained constant while changing maximum vector size and the throughput is centered around 0.6 mpps in all the cases. The reason is that, in the virtual machine case the packet generator is only able to generate packets at a rate around 0.6 mpps. VPP can handle this rate with any vector size, as we saw in the bare metal configuration where the throughput at vector size of 4 is around 2mpps. So in this case all the packets transmitted by packet generator are processed by VPP and we observe a constant throughput curve centered around 0.6 mpps. Of course there is an overhead due to virtualization effect.

2. Average Vector Size vs Maximum Vector size

2.1 Bare Metal Configuration



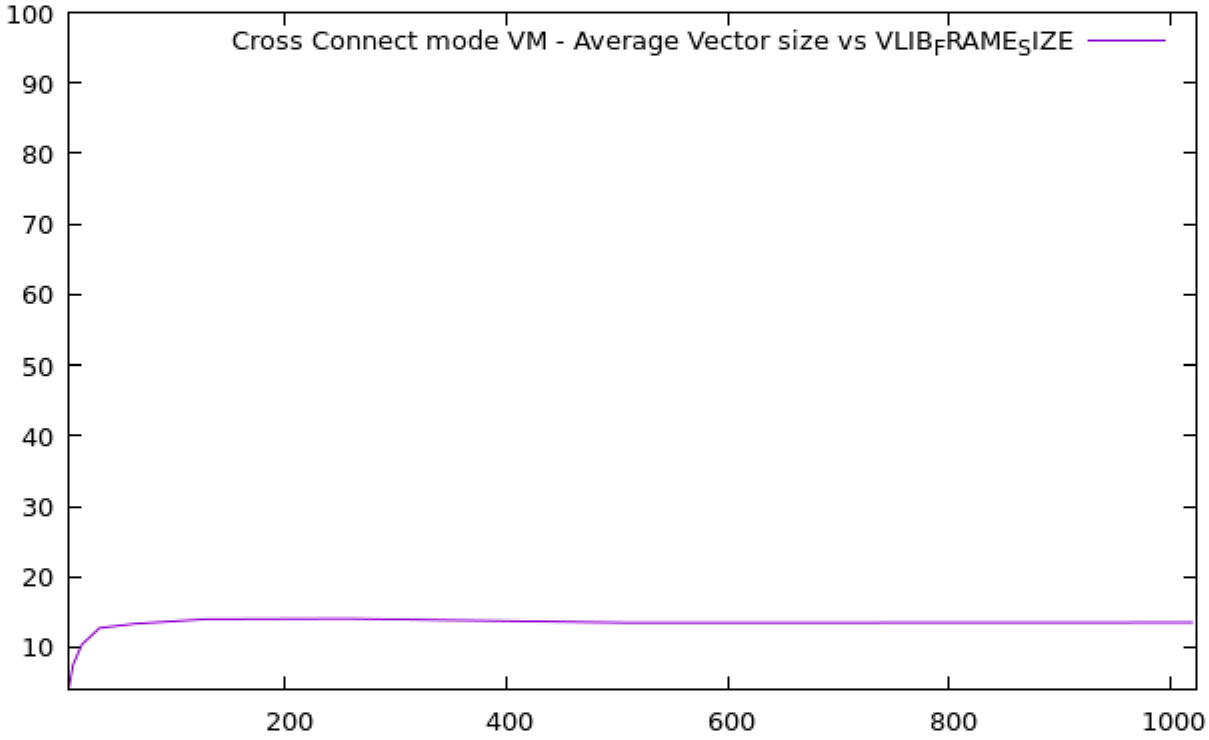
[Average Vector Size vs VLIB_FRAME_SIZE]
[y-axis vs x-axis]

Fig:3.2.1.1

In the case of IP forwarding, the average vector size per node is almost equal to the VLIB_FRAME_SIZE. It follows a straight line with approximately 45 degree slope.

In the Xconnect mode, the behavior is same till 64 and then the average vector size saturates.

2.2 Virtual Machine Configuration



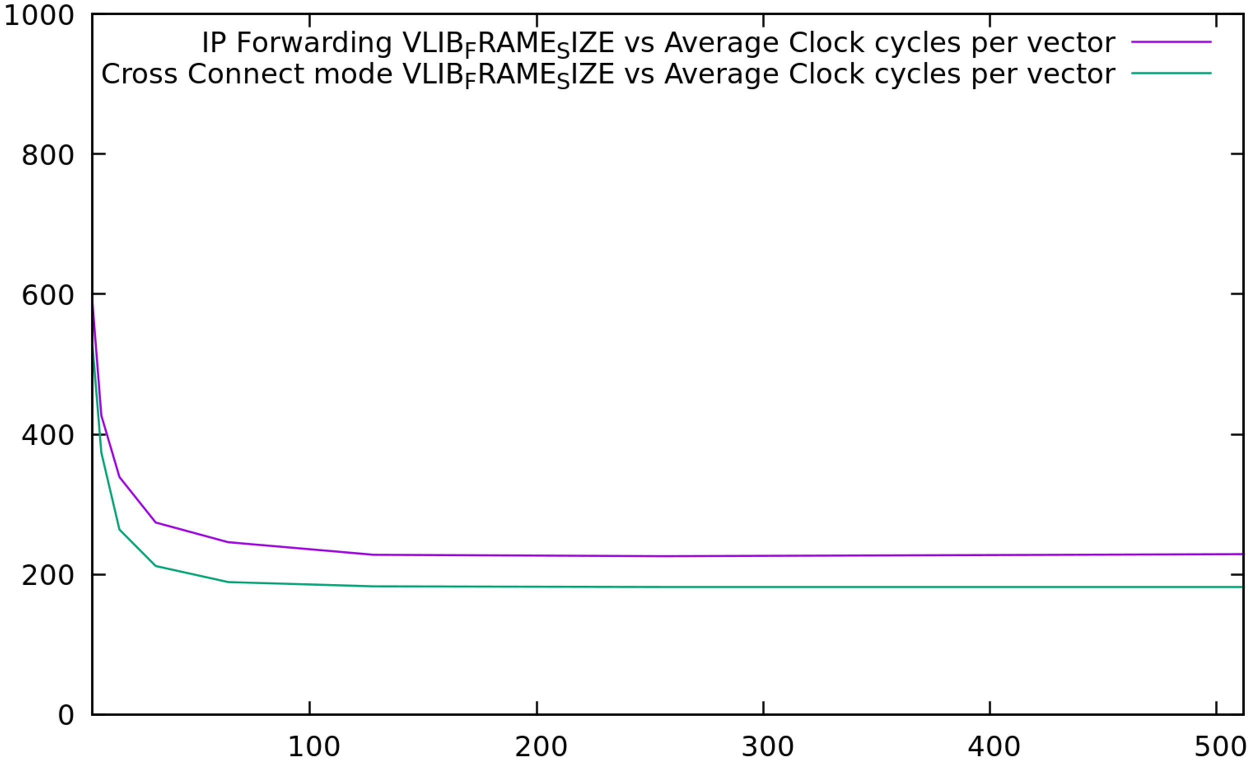
[Average Vector Size vs VLIB_FRAME_SIZE]
[y-axis vs x-axis]

Fig:3.2.2.1

The average vector rate is saturated around 13-14 in this case. This behavior is partly due to the virtual switch overhead and partly due to the virtual NIC overhead.

3. Average Clock Cycles per vector vs Maximum Vector Size

3.1 Bare Metal Configuration



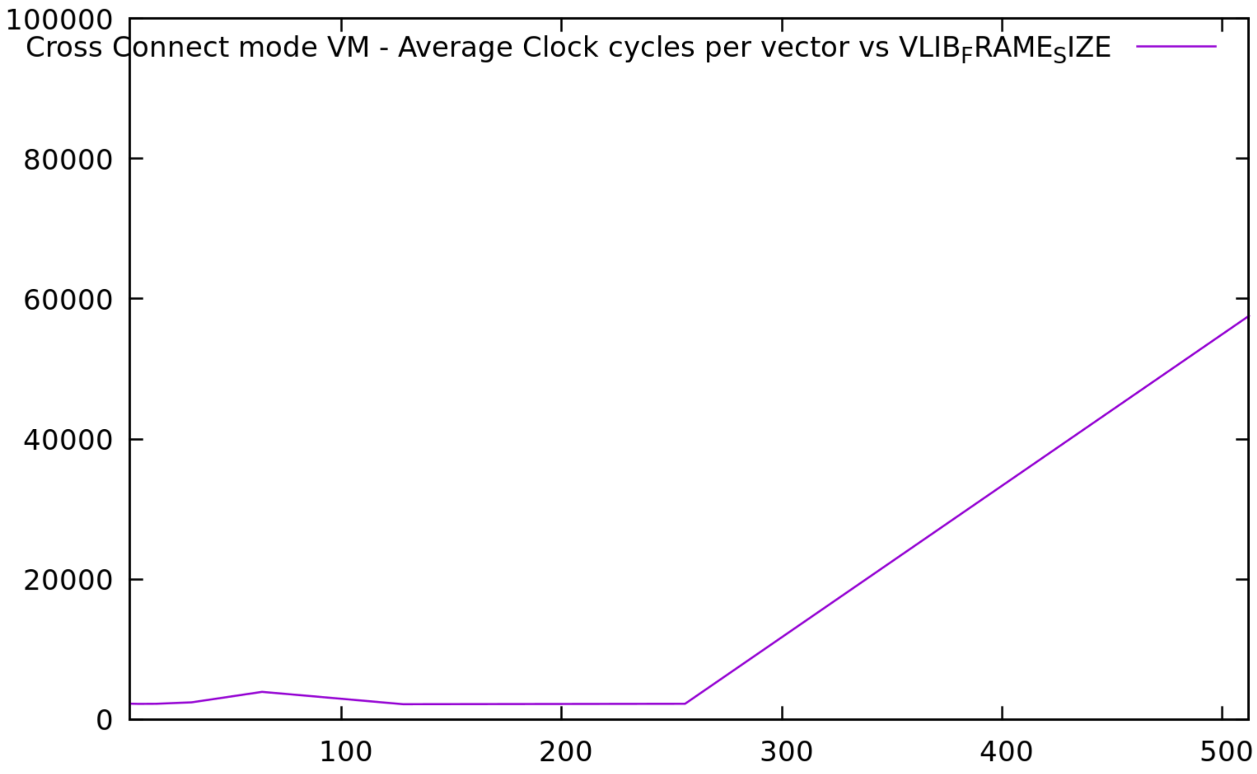
[Average Clock Cycles per vector vs VLIB_FRAME_SIZE]

[y-axis vs x-axis]

Fig:3.3.1.1

This is an expected behavior. The cache hit ratio increases as the framesize increases, until a certain point after which the ram usage increases. There is an exponential decrease from 4 to 256 and then it saturates for higher framesizes.

3.2 Virtual Machine Configuration



[Average Clock Cycles per vector vs VLIB_FRAME_SIZE]

[y-axis vs x-axis]

Fig:3.3.2.1

There are few disturbances in the graph from 4 to 64 due to virtualization. From 128 to 256 it slightly decreases and increases rapidly from 256.

All the experiments clearly show the increasing cache hit ratio from 4 to 256

Conclusion

This report shows the architecture of Cisco's Vector Packet Processor. The importance of kernel bypass techniques for performance benefits has been discussed. Some of the kernel bypass techniques used by VPP have also been discussed in the beginning of the report. Later in the report, I show the experimental results to prove vector size of 256 is roughly a convergence point between performance,resources and latency. The experiments on cpu clock cycles proves that Cache-hit ratio increases till vector size of 256 and then decreases.

Future Work

1. Flow Classification in VPP [13].

2. Implementing fair-drop and virtual queues in VPP, a stateful approach.

3. Introducing a process node for VPP, which can be used to collect any kind of statistics from any node, so that clock cycles utilized for statistics computation can be isolated from main core.

References

[1] Intel, “Data plane development kit,” URL http://dpdk.org.
[2] fd.io, “Fast Data Input Output framework” URL https://fd.io/technology
[3] VPP, “Vector Packet Processing Wiki” URL https://wiki.fd.io/view/VPP
[4] DPDK, “dpdk-pktgen” URL http://pktgen.readthedocs.io/en/latest/
[5] David Richard Barach,Eliot Dresselhaus, "Vectorized software packet forwarding" U.S Patent:7,961,636 B1 issued date June 14,2011
[6] L. Rizzo, “netmap: a novel framework for fast packet I/O,” Proceedings of the 2012 USENIX Security Symposium, no. June, 2012.
[7] Eddie kohler, Robert morris, Benjie chen, John jannotti, M. Frans kaashoek , “The Click Modular Router” Laboratory for Computer Science, MIT
[8] Cisco “ Rack mount server c-series” URL http://www.cisco.com/c/en/us/products/collateral/servers-unified-computing/ucs-c220-m4-rack-server/datasheet-c78-732386.html
[9] Cisco “UCS manager” URL http://www.cisco.com/c/en/us/products/collateral/servers-unified-computing/ucs-manager/data_sheet_c78-520522.pdf
[10] VmWare “ Vsphere and web-client” URL http://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/products/vsphere/vmw-vsphr-datasheet-6-0.pdf
[11] “VPP repository in development” https://github.com/TeamRossi/vpp-bench
[12] T. Barbette, C. Soldani, et al. Fast userspace packet processing. In ACM/IEEE ANCS. 2015.
[13] “Flow Classification” https://github.com/vamsiDT/VPP_flow_table