International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 18 (2017) pp. 7833-7837
© Research India Publications. http://www.ripublication.com

An Experimental review on Intel DPDK L2 Forwarding

Dharmanshu Johar
Department of Computer Science and Engineering,
R.V. College of Engineering, Mysore Road,Bengaluru-560059, Karnataka, India.
Orcid Id: 0000-0001- 5733-7219

Dr. Minal Moharir
Department of Computer Science and Engineering,
R.V. College of Engineering, Mysore Road, Bengaluru-560059, Karnataka, India.
Orcid Id: 0000-0001- 8256-5440

Devendra Bhardwaj
Department of Computer Science and Engineering,
R.V. College of Engineering, Mysore Road,Bengaluru-560059, Karnataka, India.

Abstract

Packet processing using the traditional techniques can be very time-consuming and resource usage for the same is very inefficient. The issue of slow processing arises from the fact that the processing techniques need to interact with the Kernel layer quite intimately and this is a laborious task. Intel DPDK is an integrated set of libraries and network interface controller drivers which can lead to a paradigm shift from using discrete working components for each module of packet processing to a consolidated model, which is simplified and shows scalability. The DPDK has a set of data plane libraries that can improve packet processing performance by many times the current processing strength provided by a traditional OS. The idea is to allow users to integrate their application software with DPDK libraries to access and make use of packet transfer acceleration provided by DPDK. This paper explains the basic architecture and functioning of DPDK along with one of its simple applications-The L2 forwarding process.[1] The results of the experimental application show the stable increment to performance that DPDK has provided. The analysis done on L2 forwarding application deduces the boost that DPDK libraries give to packet processing.

Keywords: DPDK, EAL, netmap

INTRODUCTION

Multi-core networking designs are becoming more attractive as they can help offset the problems faced with managing high cost heterogeneous network architectures. Asymmetric multiprocessing software designs, that involve bypassing the Kernel space to directly access the purpose built user space are the most recent software development techniques and are slowly becoming the norm for faster architectural design in opposition to the traditional high-overhead symmetric software designs on multi-core environment.[1]

The Intel® DPDK (Data Plane Development Kit) is one such tool, used to accelerate packet processing. [1] The Intel® Data Plane Development Kit is a set of data plane libraries and network interface controller drivers for fast packet processing. It is a BSD-licensed optimized software library for Linux User Space applications. It allows for faster packet processing than what was previously achievable using the standard Linux Kernel network stack through optimized libraries. The optimized library gives user ability to address challenging data plane processing needs. The Intel DPDK library provides a set of building blocks that can be used to create or enhance data plane solutions. [2]

Linux, which is a general purpose operating system, has been designed with a view to support a large set of applications. Intel® DPDK makes use of Linux to improve the performance and throughput of packets on Intel architecture platforms.

There are numerous ways to use Intel® DPDK. It may be used as a standalone solution for integration with customer applications or as part of commercial data plane solutions from proved ecosystem partners. The latter is specifically suited for high speed networking and data I/O intensive applications.

The typical performance of Intel® DPDK will continue to improve as Intel® also improves its own architecture and process technology. Customers will enjoy these performance and feature increases without losing any of the benefits in backward compatibility, consistency in instruction sets and tool chains and a strong ecosystem support.[2]

OVERVIEW ON INTEL® DPDK

The main purpose of Intel® DPDK lies in fast packet processing for data plane applications. It provides a simple and complete framework for the same. Some key DPDK components include the Memory Pool Manager, Buffer

Manager, Queue Manager, Ring Manager, Flow Classification, and Poll Mode Drivers for 1 Gigabit Ethernet (GbE) and 10GbE controllers, the Environment Abstraction Layer (EAL), as well as a Bare Metal Execution layer. Memory Manager is responsible for allocating pools of objects in memory. A pool is created in huge page memory space and a ring is used to store free objects. It also provides an alignment helper to ensure that objects are padded to spread them equally on all DRAM channels. Whereas, the Buffer Manager reduces the time that the operating system spends allocating and de-allocating buffers by a significant amount. The Intel® DPDK pre-allocates fixed size buffers which are stored in memory pools. Queue Manager implements safe lockless queues, instead of using spinlocks that allow different software components to process packets, while avoiding unnecessary wait times. Flow Classification provides an efficient mechanism which incorporates Intel® Streaming SIMD Extensions (Intel® SSE) to produce a hash based on tuple information so that packets may be placed into flows quickly for processing, thus greatly improving throughput. Intel® DPDK also includes Poll Mode Drivers for 1 GbE and 10GbE Ethernet* controllers which are designed to work without asynchronous, interrupt-based signalling mechanisms, which greatly speeds up the packet pipeline.

ARCHITECTURE AND OPERATION

DPDK is a complete user space implementation. The onus is on the user to understand and take advantage of the vast potential stored in this handy tool. There are various optimizations that can be achieved by using the complete instruction set of the hardware and the OS. In essence, this implies that the developers and the programmers now have a toolset powerful enough to handle their workload and also help them make their network based applications run even more efficiently over cloud and virtualized environment, all via freely available tools from the DPDK website. The complete DPDK framework is as shown in Fig. 1.1.

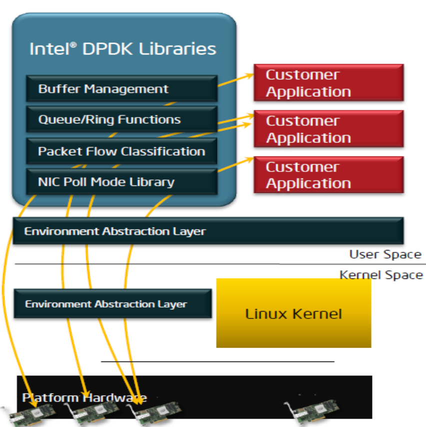

Figure 1: DPDK Architecture[1]

DPDK provides a framework through which the programmers can skip the Linux Kernel altogether and communicate directly with the Customer application in the user space. This creates a powerful tool that can massively accelerate the packet processing, as it successfully dumps the 'middleman' Linux Kernel. It makes use of the various DPDK libraries available in the DPDK framework and the Environment Abstraction Layer (EAL).[2] The EAL is used for hiding the environment specifics from the application and libraries and also to gain access to the hardware and memory space and other low-level resources. Initialization routine decides the allocation of these resources (that is, memory space, PCI devices, timers, consoles and so on). Thus, the framework creates a set of libraries for application-specific environments through the creation of an Environment Abstraction Layer, which may be specific to a mode of the Intel® architecture, Linux user space compilers or a specific platform. Make files and configuration files are used to create these environments and once these EAL libraries are created, they can be linked them to various applications. Some other libraries, outside of EAL, including the Hash, Longest Prefix Match and rings libraries are also provided.[3]

DPDK provides a scheme to overcome performance issues commonly associated with interrupt handler penalty, context switching, data copying and Linux scheduler. These areas may be acceptable for general purpose transactions but can be an issue for data I/O intensive workload when we start to talk about 10G/40G workload.

RUN TO COMPLETION MODEL

In this, all the resources are allocated prior to the call to the data plane applications that are running on logical processing cores. [3] All the devices are accessed via a polling mechanism as the model does not support a scheduler. However, interrupts are not used and hence there is no overhead causing drop in performance, as shown in Fig. 1.2.

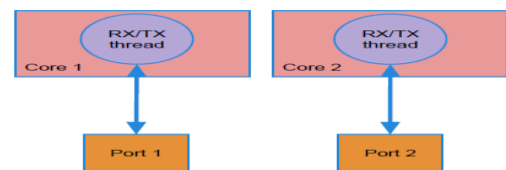

Figure 2: Run to Completion Model

PIPELINE MODEL

In addition to the Run to Completion model, the pipeline model shown in Fig. 1.3, can also be used by passing packets or messages between cores via the rings. This allows work to be performed in stages and can allow more efficient use of code on cores.

International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 18 (2017) pp. 7833-7837
© Research India Publications. http://www.ripublication.com

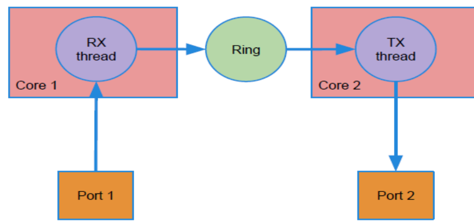

Figure 3: Pipeline Model

Layer 2 Forwarding Application

Layer 2 Forwarding is a tunnelling protocol as shown in Fig.1.4, used in order to establish a virtual private network connection over the internet, but one that does not provide encryption or confidentiality by itself. The idea behind this application, which uses Intel® DPDK, is to capture traffic on one port, modify its source and destination MAC addresses and then send it to another port. Each core is designated to receive traffic on the only one port. [4]

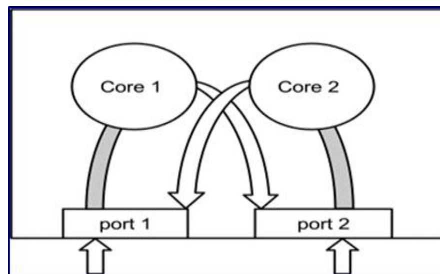

Figure 4: Pipeline Model

The DPDK API groups used in the app could be identified according to its purpose such as first, The Global initialization where initialization occurs using the *rte_eal_init* library provided in DPDK distribution, where the user must specify the number of CPU cores to be used and number of sockets among other things. Second, the setup of memory pools and port queues, where *rte_mempool_create* is used to create TX and RX pools.[5] It is then configured with buffer size, ring depth, cache size and optional NUMA socket parameter. In port initialization RX and TX queues use *rte_eth_rx_queue_setup* and *rte_eth_tx_queue_setup* for configuration respectively.[6] Based on the number of RX descriptors, RX queue is provided with a pointer for an appropriate depth and the pointer also points to an appropriate memory pool for buffers for the received packets. Both queues are configured with ring threshold registers. And third, enumeration of DPDK enabled ports, called devices, by probing over PCI bus using *rte_eal_pci_probe* command. The command *rte_eth_dev_count* returns the number of discovered devices. Ports are configured and then started using *rte_eth_dev_configure* and *rte_eth_dev_star* commands. Start threads executing infinite loops responsible for forwarding traffic are started using *rte_eal_remote_launch* command on every slave core but the master is only responsible for gathering statistics. RX/TX and modify the packets here the packets are captured in groups, called bursts, using *rte_eth_rx_burst* command, rather than individually. [7] Those packets are modified and combined in other bursts waiting for transmission. As soon as a TX burst of packets is full it is sent out of the port using *rte_eth_rx_burst*. The releasing packet buffer is not required after send whereas unsent packets have to be released to a memory pool. [8]

L2Fwd application configuration and mapping is shown in Fig 1.5. The L2 Forwarding application is responsible for packet processing using the DPDK

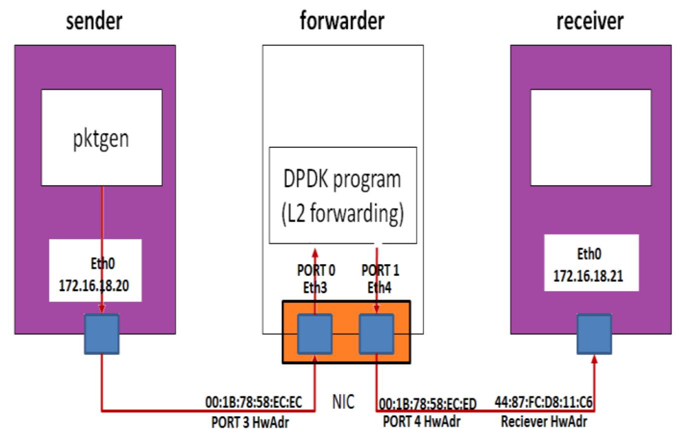

Figure 5: L2Fwd application Configuration and Mapping

The L2 forwarding application makes use of the layer 2 architecture to send and receive packets. It performs L2 forwarding for each packet that it receives on the RX_PORT and sends it to the destination port. The destination port is often the adjacent port from the enabled portmask. The portmask is defined such that if the first four ports are enabled (0xf), ports 1 and 2 forward packets to each other and ports 3 and 4 forward packets to each other. [9]

The packets sent are created using the pktgen tool which works on DPDK

RESULT AND ANALYSIS

In the paper L2Fwd application is implemented and tested. The implementation has used Port 0 and Port 1 for packet transfer. In L2Fwd, the packets are received on the RX_PORT and are then sent to the destination port which is the adjacent port on the enabled portmask. [6]So, in this case, packets are being sent from Port 0 and received at Port 1 and vice versa. The screenshots of executed code are as follows in Fig.1.6.

The packet transfer statistics are shown in Fig. 1.6 and one can see that the exact number of packets being sent from Port

International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 18 (2017) pp. 7833-7837
© Research India Publications. http://www.ripublication.com

0 is being received at Port 1 and vice versa. There is no loss in terms of packet transfer. A total of 3264 packets were sent from Port 0 to Port 1 and all 3264 were duly received at Port 1.

The statistics at the receiver end are observed using the command *vnstat –i eth0 –l*. This is shown in Fig. 1.7.

Fig. 1.8 shows steady throughput even with increased packet size. Thus, we can infer the stability that DPDK has provided from the statistical and graphical analysis.

```
Packets sent:                     0
Packets received:                 0
Packets dropped:                  0
Statistics for port 8 ------------------------------
Packets sent:                     0
Packets received:                 0
Packets dropped:                  0
Statistics for port 9 ------------------------------
Packets sent:                     0
Packets received:                 0
Packets dropped:                  0
Statistics for port 10 ------------------------------
Packets sent:                     0
Packets received:                 0
Packets dropped:                  0
Statistics for port 11 ------------------------------
Packets sent:                     0
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:            3264
Total packets received:        3264
Total packets dropped:            0
==================================================
```

Figure 6 : Packet Statistics in L2 Fwd experiment

```
eth0  /  traffic statistics

                               rx        |        tx
---------------------------------------+------------------
bytes                         23.01 MiB  |       543 KiB
---------------------------------------+------------------
        max                 1.36 Mbit/s  |     96 kbit/s
    average               281.36 kbit/s  |   6.48 kbit/s
        min                  152 kbit/s  |      0 kbit/s
---------------------------------------+------------------
packets                          290508  |          4734
---------------------------------------+------------------
        max                     915 p/s  |        85 p/s
    average                     433 p/s  |         7 p/s
        min                     262 p/s  |         2 p/s
---------------------------------------+------------------
time                       11.17 minutes
```

Figure 7 :lPacket Statistics observed at receiver end

Table 1: Statistics obtained from the experiment

Throughput	Packet Size (Bytes)
7.2	80
3.9	120
3.3	260
3.1	510
3	770

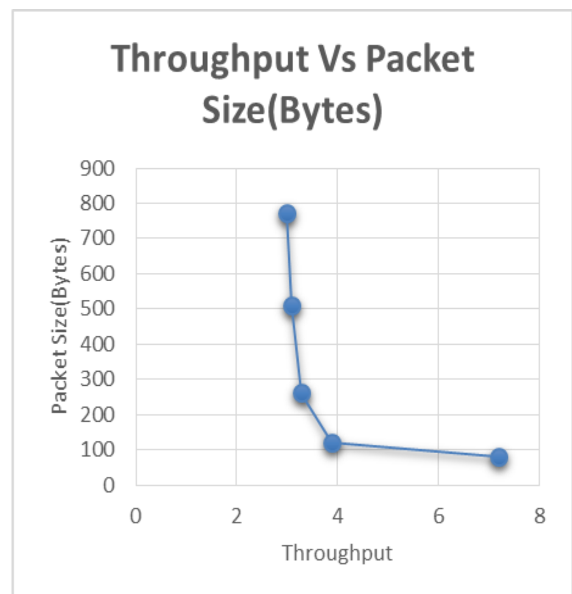

Figure 8: Graph plotted using statistics

CONCLUSION

Although the vast potential of DPDK libraries is yet to be made use of, the simple analysis done above shows that the packet throughput steadily increases with decrease in packet size when L2 packet forwarding is done using DPDK. The results of the analysis imply that DPDK can make a tremendous difference in acceleration of packet transfer and in our understanding and perusal of packet flow analysis.

ACKNOWLEDGMENT

The Authors would like to extend their gratitude to Intel for providing the reference manual and to the DPDK developers for regularly providing updated information on DPDK.

References

[1] Intel dpdk programmer's guide. intel/dpdk-prog-guide-1.7.0.pdf.

[2] Intel, "Data Plane Development Kit", online (retrieved June 2014). [Online]. Available: http://dpdk.org/.

[3] L. Rizzo, “netmap: A Novel Framework for Fast Packet I/O”, in Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12). Boston, MA: USENIX, 2012, pp. 101–112.

[4] “Intel 64 and IA-32 Architectures Optimization Reference Manual.” Intel Corporation, 2012.

[5] “PF_RING ZC,” http://www.ntop.org/products/pf_ring/pf_ring-zc-zero-copy.

[6] “Intel 82599 10 GbE Controller Datasheet Rev. 2.76.” Intel Corporation, 2012.

[7] “Impressive Packet Processing Performance Enables Greater Workload Consolidation,” Intel Corporation, 2013.

[8] “Impressive Packet Processing Performance Enables Greater Workload Consolidation,” in Intel Solution Brief. Intel Corporation, 2013, Whitepaper.

[9] “Intel I/O Acceleration Technology,” http://www.intel.com/content/www/us/en/wireless-network/accel-technology.html, Intel.

[10] Soumya Mahalakshmi A; Amulya B S; Minal Moharir, A study of tools to develop a traffic generator

for L4 – L7 layers 2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)Year: 2016, Pages: 114 - 118, DOI: 10.1109/WiSPNET.2016.7566102