VPP for Data Plane Developers – An Introduction

by **Ajith Krishnan** | February 21, 2021

There are multiple challenges to a Data plane system developer. One major challenge is - how to maintain high-speed packet processing performance with ever increasing line speed to hundreds of Gbps or Tbps. Another challenge is in the handling of rapidly growing routing table size, e.g., more than 500,000 entries. In this blog we are providing an overview of a concept-Vector Packet Processing (VPP) technology along with insights into what goes on under the hood. This will enable the developers to get a basic understanding to get streamlined into actual development using VPP. In depth technical details are omitted and instead a high level overview of basic underlying technology is provided. We have included few very useful links to the latest topics on development of systems and applications using VPP.

Driven by media-rich and bandwidth-intensive Internet applications, line packet speeds have gone from 10G/40G to 100Gigabit Ethernet and beyond. Programming of Network switches (or data planes) has become increasingly important with increasing network virtualization in the Internet infrastructure and large-scale data centers.

Traditionally handling the extreme speeds in data plane packet processing was achieved with custom application-specific integrated circuit (ASIC) or field programmable gate array (FPGA) designed specifically for such purposes. The evolution of FPGA and ASIC technology had limitations concerning speed, density, power, and pin interfacing. While clock frequencies

cannot keep up with the higher interface speeds, massive parallelism and deep pipelining was needed to scale the throughput. While using vanilla Linux kernel for high speed packet processing takes large amount of CPU cores and needs high end processors. These factors are known limitations in the evolution of high-speed packet processing ecosystem.

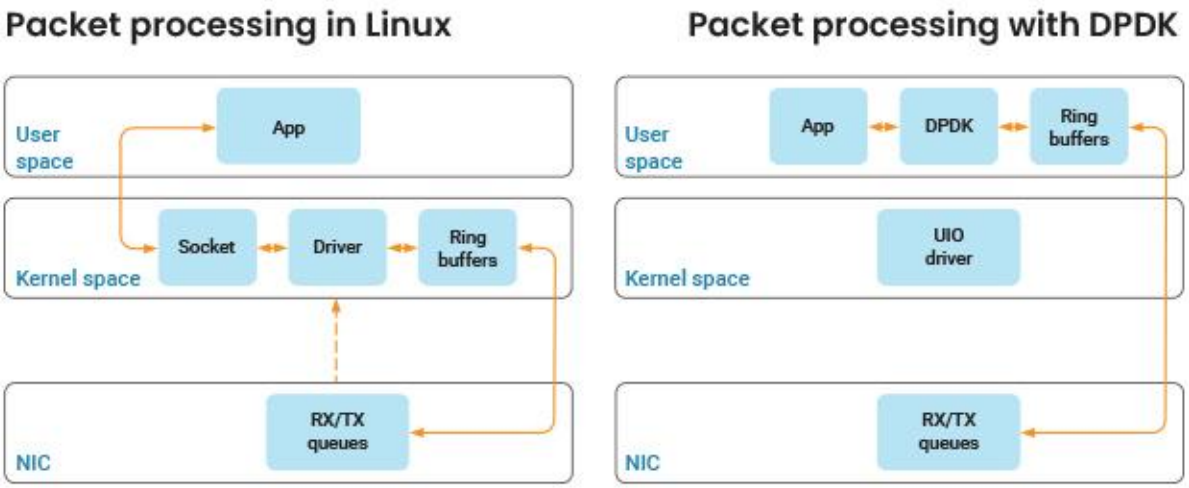
To overcome these limitations, the concept of fast data plane (FD.io) was introduced. FD.io is a secure universal data plane which is a cross platform network stack supported by multiple hardware vendors. VPP is the technology used in FD.io to achieve high speeds at low computing power.

Now let's examine what exactly VPP or Vector Packet Processing is. To understand the concepts of VPP, we first need to understand how packets were processed before VPP. The basic components of packet processing in a Linux system are a NIC or network interface card, the kernel, and the application which processes the packets. The following diagram represents a network packet at wire level.

This contains information like the headers which help in routing the packet to the destination as well as the payload intended for the application. The headers contain information like the MAC address of the NIC to which this packet is intended. The NIC has a set of transmission and receive queues per CPU. The flow of packets is as follows.

The packet arrives at the NIC, packet is copied to kernel memory, kernel generates an interrupt to start the packet processing, a set of routines are called to pass the packet to the protocol stack, the IP filter routines for firewall handling are invoked, it is determined by the kernel whether the packet is intended for local delivery or forwarding, packet is transferred to the TCP stack and finally packets are passed on to the application via the socket read system call.

There are a whole lot of procedures to get the packet to the application layer. Natural question coming to our mind will be – can this flow be optimized further? Precisely this optimization is done by a framework used in VPP called DPDK***. DPDK stands for Data Plane Development Kit. DPDK optimizes the above flow as depicted in the in the below diagram:



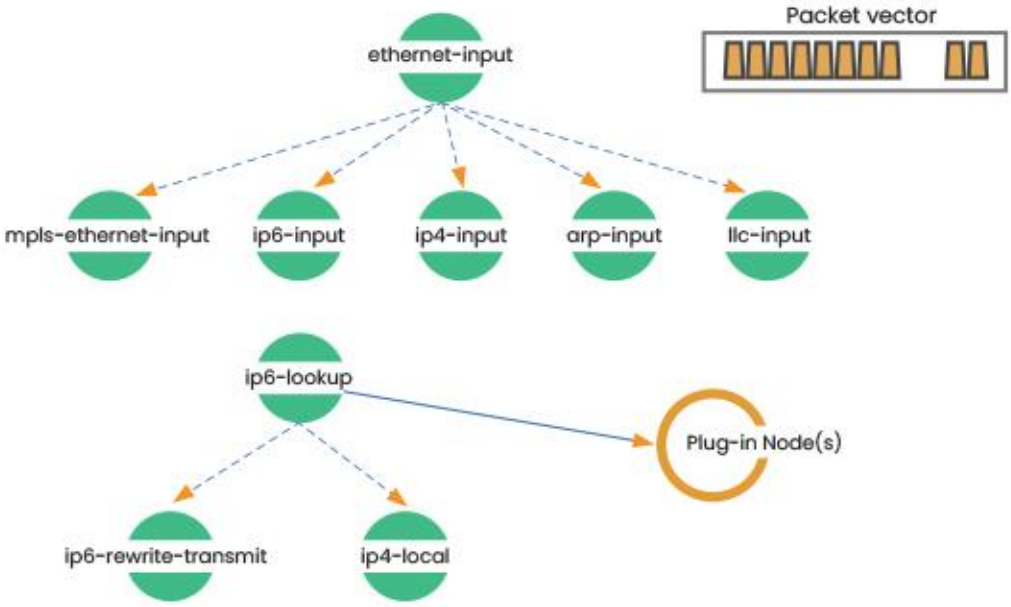
***DPDK requires a supporting CPU as well as a supported NIC. The list of supporting CPUs and NICs can be found in the following link:

https://core.dpdk.org/supported/

How does DPDK achieve this optimization? It provides a set of libraries and NIC poll mode drivers, which offload TCP packet processing from kernel to the application process in the User Space. Couple of techniques is used by DPDK namely FastPath (Kernel bypass) and Poll Mode Driver (PMD). As earlier mentioned FastPath eliminates the context switching when moving between kernel and user space, and PMD helps in polling for the packet instead of the interrupt mechanism to process the packet. DPDK also contains other techniques for aiding the packet processing which are not discussed in this blog.

Coming to VPP, DPDK is an important component. Once the packets are received at user application via DPDK, the VPP engine kicks in and forms a packet processing graph. It grabs all available packets coming from the RX queue in the NIC, and forms a vector of packets. A packet processing graph is applied node by node as per the following diagram:

Custom Application/Custom Packet Processing Graph



New graph nodes can be introduced to the above hierarchy to add custom processing. VPP provides an extensible framework to introduce new graph nodes.

To explain a bit more about the traditional method of packet processing, Scalar packet processing refers to processing one packet at a time. As mentioned earlier, this involves processing an interrupt, calling a set of routines and then deciding on the destiny of the packet. This leads to inefficient processing, since each packet has to be processed by some set of instructions loaded into the instruction cache (i-cache). This involves reading the packet from the queue, loading the instruction set, performing the checks on the packet, unloading the instruction set, routing the packet, and then repeating the same procedure for the entire set of packets received. When the code path length exceeds the size of the CPU instruction cache, thrashing occurs as the CPU is continually loading new instructions. In this method, each packet incurs an identical set of i-cache misses, increasing the time taken for processing each packet.

Vector packet processing resolves this inefficiency by processing more than one packet at the same time, since all packets have to go through the same instruction set in sequence. This avoids the continuous loading and unloading of the instruction set to and from the cache, thereby reducing the cache misses as well as read latency. As the size of the vector increases, the processing cost per packet reduces.

A rough everyday analogy to this would be to consider the problem of a stack of lumber where each piece of lumber needs to be cut, sanded, and have holes drilled in it. You can only have one tool in your hand at a time (analogous to the instruction cache). You are going to finish

cutting, sanding, and drilling the lumber faster if you first pick up the saw and do all of your cutting, then pick up the sander and do all of your sanding, and then pick up the drill and do all of your drilling. Picking up each tool in order for each piece of lumber is going to be much slower.

The main advantage of VPP is that it achieves high speeds without requiring multiple processing cores that are needed to achieve the same speeds in traditional Linux processing. Currently FD.io and VPP are being used to build high end virtual Load Balancers, Firewalls, Switches and Routers.

For getting started with VPP, there are prerequisites like specific hardware as mentioned in below link: https://core.dpdk.org/supported/

A guide to getting started is available at the following link :

https://my-vpp-docs.readthedocs.io/en/events-section/gettingstarted/index.html

There are both developer versions as well as commercial versions of VPP, latter can be used out of the box to create applications, and former can be used to make contributions to the VPP open source.

VPP can be used in a multitude of use cases, details of which are available at the following link:

https://fd.io/docs/vpp/master/usecases/index.html

And finally a practical use case of VPP is development of a next generation firewall (NGFW) with all advanced features like Deep Packet Inspection. First attempts at a NGFW firewall have been implemented using VPP and Snort by Netgate, the open source networking platform is called TNSR.

https://www.netgate.com/blog/beyond-next-generation-firewall-take-me-there.html

To summarize, VPP is a path breaking technology which will be used for many practical applications in the packet processing ecosystem and it will have widespread uses.