논문 19-45-01-08 The Journal of Korean Institute of Communications and Information Sciences '20-01 Vol.45 No.01
https://doi.org/10.7840/kics.2020.45.1.42

데이타 플레인 가속화 기술들에 대한 기술 동향 및 전망

김 용 근[◆]

A Technical Trend and Prospect of Data Plane Acceleration Technologies

Yongkeun Kim[◆]

요 약

네트워크 트패픽은 폭발적으로 증가해왔고, 이의 증가는 계속될 것으로 예상된다. 이러한 네트워크 요구사항을 맞추기 위해, 10 Gbps, 40 Gbps 혹은 그 이상의 높은 대역폭을 갖는 NIC들이 개발되어 사용되고 있다. 그러나, 이러한 고대역폭 NIC들로부터의 트래픽이 범용 운영체제를 탑재한 시스템에서 적절히 처리된다는 보장은 없다. 이는 범용 운영체제가 네트워크에 특화된 환경이 아닌, 일반 컴퓨팅 환경을 위해 개발되었기 때문이다. 이런 고대역폭 요구와 운영체제의 처리능력 사이의 차이를 해소하기 위해, 범용 운영체제 기반에서 많은 고속 패킷 처리 기법들과 데이타 플레인 가속화 기술들이 제안되었고 구현되어 왔다. 본 논문은 범용 운영체제 (리눅스) 기반에서의 패킷처리와 문제점들을 체크한 후, 다양한 고속패킷처리 기법들에 대해 조사하고, 이 고속 패킷처리 기법들을 이용한 데이타 플레인 가속화 기술들에 대해 고찰한다.

Key Words : fast packet processing, data plane acceleration, kernel bypass

ABSTRACT

The network traffic has been exponentially grown and it is expected that the growth will continue. To meet the market requirements, high bandwidth NICs with 10 Gbps, 40 Gbps or even more have been developed and widely used. However, there is no guarantee that the traffic from the high bandwidth NICs can be properly processed in a system with general purpose operating systems, because the operating systems have been developed for general computing not for network centric. To fill the gap between high bandwidth requirement and operating system's processing capability, many fast packet processing techniques and data plane acceleration technologies have been proposed and implemented upon general purpose operating systems, especially Linux. In this article, after checking the packet processing of general purpose operating system, focusing on Linux, and its issues, then, various techniques for fast packet processing are investigated. It, then, reviews the different types of data plane acceleration technologies with the packet processing techniques.

Ⅰ. Introduction

The required network bandwidth has been increased drastically because more and more devices are connected to network and applications require more and more bandwidth. The network equipment suppliers need to support those bandwidth requirements by providing with higher performance equipment periodically. Traditionally, network equipment suppliers have used their own proprietary

◆ First Author : KoreaQuest, Inc., mkim@koreaquest.net, 정회원
논문번호 : 201910-236-B-RE, Received October 18, 2019; Revised November 3, 2019; Accepted November 7, 2019

hardware and software to perform specific network purposes, because it was likely a feasible way for them to meet market requirements, competing against other vendors, and there were little common hardware platforms to use. It can give them their own specialties beating competitors and dominating market, however, it also increases the development and manufacturing costs, inflexibilities in terms of new hardware and software development, and difficulties to meet time-to-market while the market requires quicker service launch.

However, this trend has been changed for more than last decade, as the general purpose processors have been getting more processing capabilities and more network features because networking became one of essential features even in pure computing systems. Moreover, general purpose operating systems, such as Linux, already have enough networking protocol stacks. More and more network vendors have started to develop their own equipment using the general purpose operating systems on top of the general purpose processors. It allows the vendors to do rapid developments in timely manner, reducing hardware development cost and schedule, letting them focus on their own software development with flexibilities, and supporting proper packet processing performance required from the market. Even in a data center or cloud, the network traffic processed by single server is getting higher with high bandwidth NICs (Network Interface Cards). 10 Gbps (Giga-bit per second) is getting popular and 25/40 Gbps even 100/200 Gbps NICs are available in the market.

The issue is the growth of NIC's processing bandwidth is becoming faster than the CPU's (Central Processing Unit), so, the question would be whether a general purpose operating system can process the traffic from those high-speed NICs or not, as its network stack is conceived for general purpose communications rather than high-speed networking applications. It turned out that the operating system network stack can hardly support multiple 10 Gbps interface packet processing at line rate, through many experimental studies so far[1,2,3]. To overcome the hurdle, various techniques and technologies have been proposed and implemented.

In this article, after checking the packet processing in a standard operating network stack focusing on Linux in Section II, the various techniques to reduce the performance bottlenecks are reviewed in Section III. The current status and activities of DPA (Data Plane Acceleration) technologies are reviewed and described in Section IV, with conclusion in Section V.

Ⅱ. General Packet Processing and Its Issues

This section describes the general packet processing in a standard operating system, Linux, and its issues in a performance perspective.

2.1 General packet processing in Linux

The most commonly used operating systems provide a network stack generally focusing on compatibility rather than performance. As Linux has been widely used for both of network vendors and general computing area, this article uses Linux as a reference of operating system unless others are specified.

Linux network stack follows an interrupt-driven basis as typically below and described in Fig. 1[4].

1) Each time a new packet arrives into the corresponding NIC, this packet is attached to a descriptor in a NIC's receiving (RX) queue, which is typically circular and referred as rings. This packet descriptor contains information regarding the memory region address where the incoming packet is copied via a DMA (Direct Memory Access) transfer.

2) After DMA the packet from NIC to the DMA memory region, RX interrupt is raised and the corresponding interrupt software routine is launched and copies the packet from the DMA memory region into a local kernel packet buffer (referred as **sk_buff** structure in Linux). Once the copy is made, the packet descriptor is released so that it can be used for receiving new packets.

With NAPI (New Application Programming Interface supported by Linux kernel 2.6 or higher),

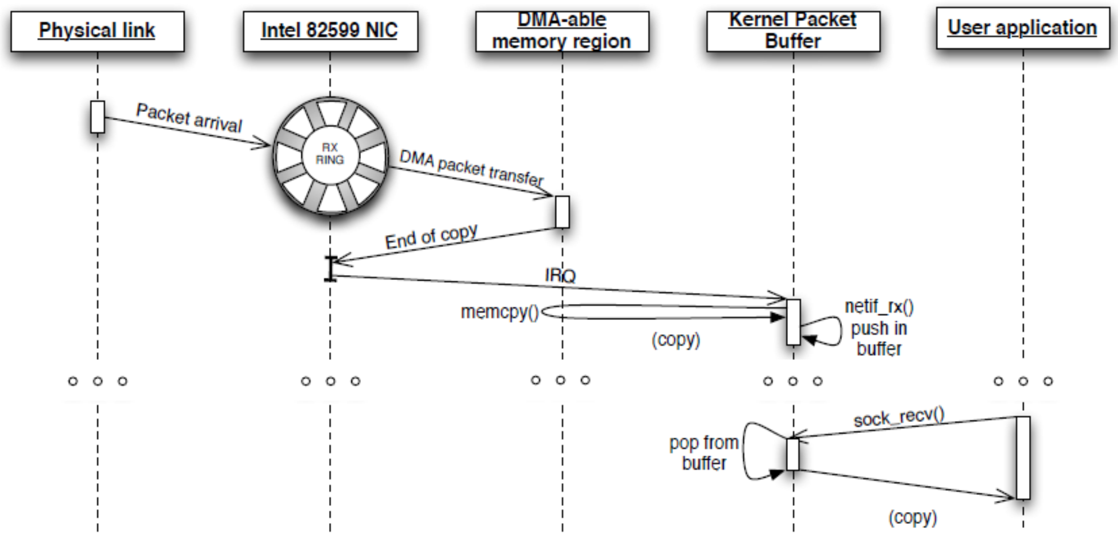

그림 1. 리눅스 NAPI RX 흐름[4]
Fig. 1. Linux NAPI RX scheme[4]

when the RX/TX (Receiving/Transmission) interrupt arrives, its NAPI aware interrupt routine schedules the execution of a **poll** function, disabling future similar interrupts. The **poll** function checks if there are any packets available, and copies and enqueues them into the network stack if ready, without waiting to an interrupt. The same **poll** function, then, will reschedule itself to be executed in a short future (i.e, without waiting to an interrupt) until no more packets are available. If there are no more packets to process, the corresponding interrupt is activated again. NAPI compliant drivers can drop traffic at NIC level, without unnecessary work (i.e, copying the packets into the kernel packet buffer as NAPI unaware drivers, previous to kernel 2.6, drop the packet in kernel level). These polling mode of NAPI is more CPU consumer than interrupt-driven when the network load is low, but, it shows less cost with heavy network load.

3) The kernel packet buffer structure with the just received packet data is pushed into the system network stack for protocol processing. The packet in the kernel packet buffer is, then, copied to user applications for further processing.

2.2 Issues of the general packet processing

[5] describes the factors affecting to packet processing mostly in hardware perspective, which are inadequate utilization of CPU capacity, interrupt overhead with possibility of interrupt livelock, limited bus bandwidth, RAM (Random Access Memory) latency, and I/O (Input/Output) latency. NAPI shows better interrupt handling for interrupt storm as it can happen when high-speed NICs get short packets at line rate. However, it is not enough to overcome the performance challenge in high-speed NICs because there are many other architecture problems in software perspective. More detailed factors in software perspective are described[4] as below.

Per-packet allocation and deallocation of resources. Every time a packet arrives to the system via an NIC, memory resource for a packet descriptor (and a kernel packet buffer) has been allocated and de-allocated. Its cost is significant especially for high performance packet processing. The kernel packet buffer (**sk_buff**) related operations consume 63.1% of CPU power in the reception process of a 64B packet[6]. The buffer allocation and de-allocation take near 1,200 and 1,100 cycles respectively[7].

Serialized access to traffic. Thanks to multiple hardware RSS (Receiver Side Scaling) queues that can distribute the traffic using a hardware-based hash function, the receiving process can be parallelized by assigning each RSS queue to a specific core. However, the kernel network stack merges the packets at a single point for layer 3/4 analysis. It causes a bottleneck for N:1 mapping and user space process can not get packets from a specific RSS queue. It affects to the performance degrade, loosing parallel processing benefits at the driver level.

Multiple data copies from driver to user space. There are multiple data copies in the packet processing, from DMA memory region to kernel packet buffer and from the kernel packet buffer to user space applications, understanding single data copy consumes about 500 to 2,000 CPU cycles per packet depending on the packet size[7].

Context switching between kernel and user space. The context switching is required to traverse between user space and kernel space, caused by system calls for packet processing. It consumes up to 1,000 cycles per packet[7].

No exploitation of memory locality. There are always possibilities for cache misses and their cost is significant (13.8% of CPU cycles in reception of 64B packet[6]). In NUMA (Non-Uniform Memory Access) architecture, which has become a reference for multi-processor architecture, the memory locality is more important as it affects to both cache misses and longer memory latencies (for example, when a processor accesses to a memory region which belongs to another processor).

Ⅲ. Techniques for Fast Packet Processing

There have been many proposed and implemented techniques to enhance the packet processing performance in considerations of the issues specified in Section II, along with operating system's kernel itself. In this section, these techniques are explained in software and hardware approaches.

3.1 Software based approaches

For fast packet processing, most of the techniques are software based because the general purpose operating systems have not been designed for fast packet processing and its enhancements are mostly related to the operating system's and software I/O processing. These have been researched and specified in many studies[1-7].

1) ***Kernel bypass***. One of the ways to achieve fast packet processing is to bypass the kernel and to let user space applications handle the packets directly from NICs as the kernel itself is the main bottleneck in architectural perspective even with certain enhancements. With kernel bypass, the user space is responsible for implementing the rest of the network stacks, meaning the user space must implement the TCP/IP (Transmission Control Protocol / Internet Protocol) protocols suite and provide interfaces for applications to access messages carried over by the protocols. This scheme became the main stream of data plane accelerations and there are many open societies and technologies to achieve it, which will be explained more in Section IV.

2) ***Zero-copy***. In typical Linux network packet processing by kernel, it shows at least two packet copies to deliver to user space applications (from ring buffer to kernel packet buffer, then, from the kernel packet buffer to the applications) unless the packets are dropped in some reasons or forwarded to other NIC interfaces (for routing application for example). Mapping the kernel packet memory region to user space can provide user space applications with the access to packets without the additional copy from kernel packet buffer to user applications. It is already implemented in the current Linux kernel as a raw socket with RX_RING/TX_RING socket option[4]. This same concept can be applied for DMA memory region to kernel. As most of data plane acceleration technologies use kernel bypassing, the zero copy technique from DMA memory region to user space applications has been used. In kernel bypass operation, it can be achieved by arranging for a buffer pool to reside in a shared region of memory visible to both NICs and user space software.

3) ***Pre-allocated packet buffers***. It consists of the pre-allocation of all memory resources required for packet processing (data and meta-data (descriptor)), to reduce the per-packet memory allocation overhead. Also, when a packet has been processed, its memory resources are not released to the system but re-used for new incoming packets.

4) ***Parallel direct paths***. Direct parallel paths between RSS queues and applications need to be processed to solve the serialized access to traffic for layer 3/4 analysis. It may achieve the best performance when a specific core is assigned both for taking packets from RSS queues and forwarding them to the user space. It requires certain modifications of the data exchange between kernel space and user space in Linux.

5) ***Batch I/O processing***. It groups packets into a buffer and processes them to kernel or user memory in groups called batches. It reduces the number of system calls and the consequent context switchings, and mitigates the number of copies. It amortizes per-packet processing overhead. However, it increases the latency and jitter, and may cause timestamp inaccuracy because packets have to wait until a batch is finished or a timer expires.

6) ***Forward pre-fetching***. To reduce cache misses, the driver may pre-fetch the next packet while the current packet being processed. The idea is to load the memory locations, which will be likely used in a near future, into processor's cache in advance for faster access when required.

7) ***Affinity***. Memory affinity is for a process to allocate/use memory assigned to the processor in which it is being executed, to exploit memory locality. CPU and interrupt affinity is also important as it is more likely to find packets in a local cache if previously these data have been received by an interrupt handler assigned to the same core.

8) ***Huge page support***. In modern CPU architectures, memory is managed as pages, which are virtually and physically contiguous blocks of memory with a standard page size of 4KB (Kilo Bytes). When an application is run, the page addresses for accessing memory locations need to be translated from the virtual to the physical. To improve performance, the most recently used for the translation is kept in a cache, called TLB (Translation Lookaside Buffer). As each page occupies an entry in the TLB, the bigger page size the less cache misses. Intel-64 architecture can support 1 GB (Giga Byte) page size in 64-bit addressing mode, along with 4 KB and 2 MB (Mega Byte) page sizes, so, supporting this huge page size takes an advantage of less cache misses, resulting in faster packet processing.

9) ***Lockless programming***. Locking is a conventional mechanism for programming in multi-process environment, to synchronize the access to a resource. Although it is one of the simplest ways to synchronize, it also shows many disadvantages, such as contention, locking overhead, lack of composability, priority inversion, convoying, etc. These issues directly impact on the performance of packet processing, so, for faster packet processing, programming and operations in lockless or in a way to reduce locking are necessary.

10) ***Hardware multi-queue support***. Most of modern NICs can process packets in multiple hardware queues, which prove very useful not only for load balancing and dispatching but also for better I/O performance especially in multicore systems. For example, RSS hashes some pre-determined packet fields to select a queue, while queues can be associated with different cores. In this way, traffic from a single NIC can be distributed among different cores. It is not pure software solution, but, as it is a part of commodity hardware typically used with the existing software techniques above, this article regards it as a part of software based.

3.2 Hardware based approaches

There are two main streams currently used for fast packet processing techniques in hardware perspective, which are GPU (Graphics Processing Unit) or FPGA (Field Programmable Gate Array) based.

1) ***GPU based processing***. The GPUs offer extreme thread level parallelism, while CPUs maximize instruction-level parallelism. In general, GPUs are very well suited for packet processing

applications as they offer data-parallel execution model. It can posses thousands of processing cores and they adopt single-instruction, multiple thread (SIMT) execution model where a group of threads execute concurrently[3,8].

2) ***FPGA based processing***. The FPGAs have a massive amount of parallelism built-in because they posses millions of LEs (Logic Elements) and thousands of DSP (Digital Signal Processing) blocks. However, they have an increased programming complexity which is the main challenge for using an FPGA as an accelerator. There are HLS (High-Level Synthesis) tools that try to overcome this problem by allowing to program FPGAs in high-level programming languages[3].

Ⅳ. Technologies for Data Plane Acceleration

There have been lots of activities and committees for acceleration of data plane performance since early 2000. This section reviews and summarizes most of well known approaches.

4.1 DPDK (Data Plane Development Kit)

One of the most widely used technologies is called DPDK, which was initiated by Intel Corp. in 2010 and became a fully open-source project. The open-source committee was established at DPDK.org in 2013 and has facilitated the continued expansion of the project. It consists of libraries and drivers, also known as PMDs (Poll Mode Drivers), to accelerate packet processing workloads running on the variety of CPU architectures, including x86, POWER (Performance Optimization With Enhanced RISC (Reduced Instruction Set Computer)) and ARM (Advanced RISC Machines) processors, mostly in Linux user space (packaged in Fedora, Ubuntu, Debian, RedHat, etc), with a FreeBSD port available for a subset of DPDK features. It is licensed under the Open Source BSD (Berkeley Software Distribution) License[9]. This libraries and drivers abstract away the low-level implementation details, providing flexibility as each vendor implements its own low-level layers. DPDK has been getting more popular in recent years, in collaboration with many open-source projects, such as OVS (Open vSwitch), ODP (Open Data Plane), OFP (Open Fast Path), vDPA (vhost Data Path Acceleration), OPNFV (Open Platform for Network Function Virtualization), etc.

The basic concept is to let user space applications process packets without involving Linux kernel network stack (which is the main bottleneck for packet processing performance), communicating directly with networking device described in Fig. 2[10]. In Linux network processing (left side of Fig. 2), it separates network packet processing routines from user applications, using the existing routines in kernel. When applications in user space send/receive packets, the packets are processed in kernel network stack along with NICs using interrupts. This approach can give some benefits for application developers as it makes the developers to be free from the network packet protocol processing, letting the kernel process them instead. However, the switching between kernel mode and user mode needs to be done for packet processing via system calls, and the cost of it is not trivial. Also, there are other costs for interrupt processing of incoming packets from NICs and data copying from the kernel to user space. These overheads are serious especially for applications requiring high packet processing

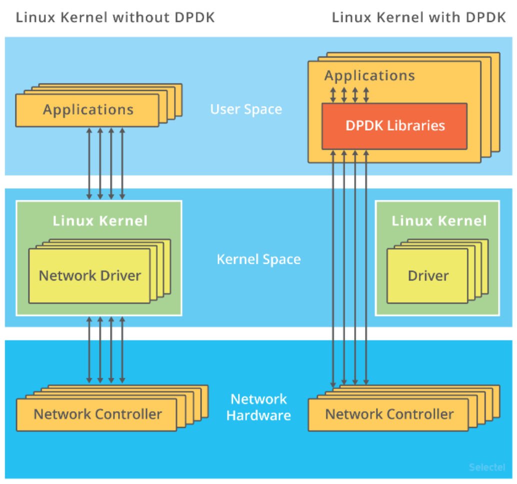

그림 2. DPDK 유무에 따른 리눅스커널[10]
Fig. 2. Linux kernel with / without DPDK[10]

performance at line rate. One of the ways to minimize this overheads is to let all the packet processing be done in user space instead of kernel space, by bypassing the kernel and it is where many data plane acceleration technologies come, including DPDK.

The basic architecture is described in Fig. 3, consisting of EAL (Environment Abstraction Layer) and data plane libraries. The data transfer is done by PMDs in direct communications with NICs by bypassing kernel and user space applications use the libraries instead of expense system calls. The EAL provides a generic interface that hides the environment specific from the applications and libraries, including the services, such as DPDK loading/launching, core affinity/assignment procedures, system memory reservation, trace and debug functions, spinlocks and atomic counters, CPU feature identification, interrupt handling, alarm functions, etc[11]. The PMDs are designed to work without interrupt-based processing mechanisms, consisting of APIs (Appllication Programming Interfaces) and provided through BSD drivers running in user space to configure the devices and their respective queues. It can access the RX and TX descriptors directly without any interrupts to quickly receive, process and deliver packets to user applications[11]. There are about 39 supported NIC drivers for both of native and virtual as of today, including 1 Gbps, 10 Gbps, 40 Gbps and paravirtualized driver called virtio. Also, many hardware accelerators for baseband, crypto, compression, etc., are supported.

The core libraries are ring manager (**librte_ring**) for providing a lockless multi-producer, multi-consumer FIFO (First In First Out) API in a finite size table, memory pool manager (**librte_mempool**) for allocating pools of objects in memory, buffer manager (**librte_mbuf**) for managing buffers that may be used by DPDK applications to store message buffers, timer manager (**librte_timer**) for timer service to DPDK execution units with the ability to execute a function asynchronously[11]. Additionally, the framework includes NUMA awareness to avoid expensive memory operations across memory nodes and huge pages to optimize physical-to-virtual page mappings within the CPU's TLB.

It supports two operational modes, run-to-completion and pipeline, described in Fig. 4. In run-to-completion model, all resources must be allocated prior to calling data plane applications, running as execution units on logical processing cores. In Fig. 4, a specific port's RX descriptor ring is polled for packets through an API. Packets are then processed on the same core and placed on a

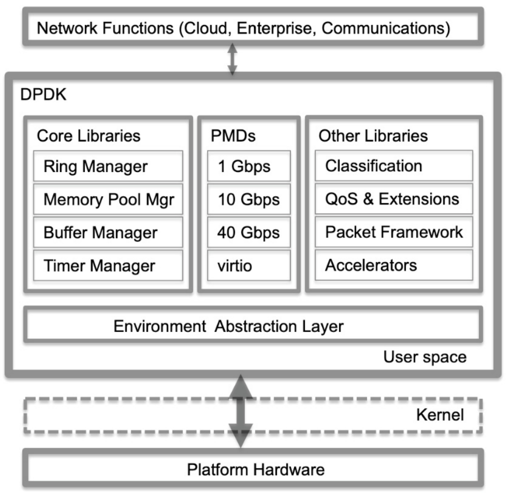

그림 3. DPDK 구조
Fig. 3. DPDK architecture

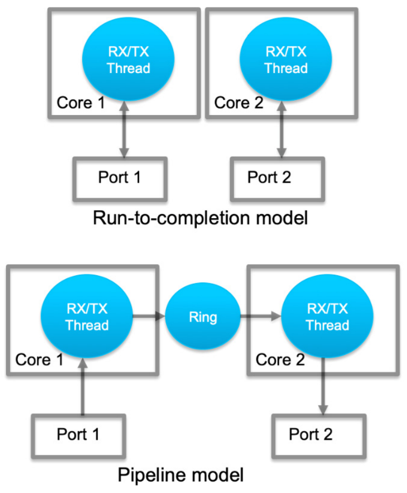

그림 4. DPDK 동작 모드
Fig. 4. DPDK operational modes

port's TX descriptor ring through an API for transmission. Each logical core assigned to the DPDK executes a packet processing loop with retrieving incoming packets, processing each received packet one at a time, and sending pending outgoing packets[11]. Pipeline model may also be used by passing packets or messages between cores via rings. This allows work to be performed in stages and may allow more efficient use of code on cores. One core polls one or more ports' RX descriptor rings through an API. Packets are received and passed to another core via a ring. The other core continues to process the packets which then may be placed on a port's TX descriptor ring through an API for transmission. Some logical cores may be dedicated to the retrieval of incoming packets and other logical cores to the processing of previously received packets[11].

DPDK has considered lots of things to enhance packet processing, not only for reducing the kernel bottlenecks mentioned earlier, but also for handling memory and I/O operations in a deliberate manner. It supports 1) huge page (to use bigger page such as 2MB or 1 GB) to reduce TLB misses, 2) NUMA awareness DPDK APIs are structured around for every operation, 3) simple way for DMA address translation, 4) awareness of underlying physical I/O memory area with IOMMU (I/O Memory Management Unit), 5) shared memory implementation for multiple processes not to require any address translations, and 6) optimized memory pools for high performance[12].

DPDK itself doesn't include Layer 2 to 4 protocol stack but a framework with layer 1 implementation (i.e, device driver level), which means TCP/IP stack should be implemented on top of it to process network packets and the way of protocol stack implementation affects to the performance significantly. The network vendors who want to use DPDK for their product development have two options, to develop the protocol stack by themselves (or outsourced) with or without open source protocol stacks or to license commercial protocol stack from protocol stack vendors.

There are some open issues for DPDK. Though DPDK supports NUMA awareness, if the packets are forwarded to ports assigned to other cores, memory access by the other cores to the receiving core's affects to the performance (lower than the ports assigned to the receiving core). Even layer 4 stack (TCP) is implemented on top of DPDK, the applications, who want to use it with socket interface, may need to be modified as its behavior is likely different from that of typical Linux socket interface. There have been tries to have the same socket interface behavior in DPDK user space, but, so far, its performance looks not good enough.

Initially, it focused on Intel x86 processors, but, after becoming open source, its supporting scope is getting wider and many other open committees use it as one of their architectures. Thanks to its high performance, many network vendors have developed their products based on DPDK and they are already in commercial phase, so, it can be regarded as a mature one.

DPDK is one of the most active open source projects, releasing new version every three month with thousands patches from hundreds people and adding more libraries. Its use is becoming a de-facto standard in the high performance packet processing area.

4.2 ODP (Open Data Plane)

ODP is to provide a common set of APIs for application portability across diverse range of networking platforms (SoCs (System on Chips) and servers) that offer various types of hardware acceleration[13], letting hardware vendors develop its actual implementation for what and how the APIs are realized, pictured in Fig. 5. After project launching in 2013, ODP implementations exist for

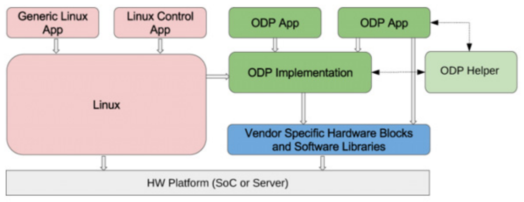
Generic Linux App
Linux Control App
ODP App
ODP App
Linux
ODP Implementation
ODP Helper
Vendor Specific Hardware Blocks and Software Libraries
HW Platform (SoC or Server)

그림 5. ODP 구조[13]
Fig. 5. ODP architecture[13]

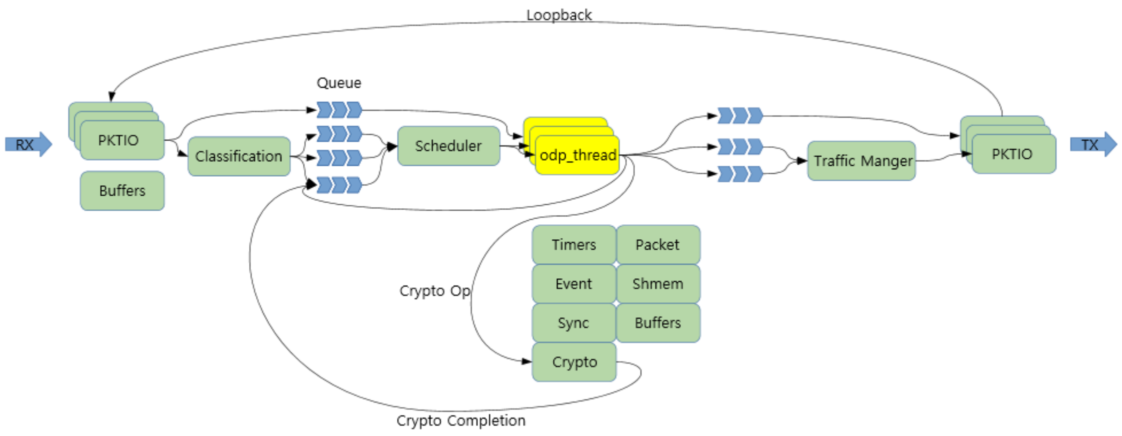

그림 6. ODP 패킷 흐름 개요[14]
Fig. 6. ODP packet flow overview[14]

ARM, MIPS (Microprocessor without Interlocked Pipeline Stages), POWER, x86 and proprietary SoCs.

ODP application packet flow is described in Fig. 6. After arriving and being received (**RX**) from a network interface represented by a logical port (called **PKTIO**), packets go either directly to **Queues** that are polled by ODP threads (**odp_thread**), or can pass through **Classification** to parse packet and apply pattern matching rules then are sorted into **Queues** that represent individual flows. These queues can then be dispatched to application threads (**odp_thread** in Fig. 6) via **Scheduler**, who is responsible for selecting and dispatching one or more events to a requesting thread. Threads, in turn can invoke various ODP APIs to manipulate packet contents prior to disposing of them. For output processing, packets make by directly queued to a **PKTIO** output queue or else they may be handed to **Traffic Manager** for controlling traffic shaping and processing programmatic quality of service before winding up being transmitted (**TX**). Output interfaces may operate in loopback mode, in which case packets sent to them are re-routed back to the input lines for "second pass" processing. For example, an incoming IPSec (Internet Protocol Security) packet cannot be properly classified (beyond being IPSec traffic) until it is decrypted. Once decrypted and its actual contents made visible, it can then be classified into its real flow[14]. Only the **odp_threads** have application logics and everything else is provided by the ODP framework and available for use by any ODP application.

ODP has released production-ready versions since 2016 and ODP-Linux and ODP-DPDK have been implemented as references. Both of DPDK and ODP work at device driver level and don't have protocol stacks, so, some commercial software vendors offer their protocol stacks on top of ODP layer. ODP is a member of OFP (Open Fast Path) who is also an open source committee for high performance TCP/IP stack. As it provides a common set of APIs for application portability, there should be optimized protocol stacks on top of it in order to achieve high performance communications.

4.3 OFP (Open Fast Path)

OFP[15] is to create and develop an open source fast path TCP/IP stack, designed to run in Linux user space (called "**fastpath**"). OFP operates on top of ODP as a protocol stack, in collaboration with Linux kernel network stack (i.e, if the packets OFP can not handle probably due to not-implemented yet for example, the packets go to Linux kernel and let its network stack (called "**slowpath**") handle it), running on ARM, x86 and MIPS and mainly contributed by Nokia, ARM and Enea. DPDK support can be made with ODP-DPDK integration layer. Its first version v1.0 was released in

December, 2015.

Its architecture is pictured in Fig. 7, consisting of network interface, ODP implementation, OFP libraries, user applications and Linux host system[15]. At least one core need to be allocated for Linux system calls for **slowpath** processing using **TUN/TAP** (network TUNnel/network TAP) interface and more cores may be allocated for Linux host if there are lots of **slowpath** traffic. Other cores can be allocated to ODP for **fastpath** processing. All the packets in incoming or outgoing are processed by ODP, then, the packets are delivered to OFP or Linux host depending on the decision whether these can be processed in OFP or not. **User Conf Code** is for a management of ODP and OFP and **User/Default Dispatcher** is a dispatcher implementation that reads packets through the ODP APIs. The routing and ARP (Address Resolution Protocol) tables are synchronized between Linux host (**slowpath**) and OFP (**fastpath**) using **Netlink** API and system commands.

Its **fastpath** protocols consist of Layer 4 (UDP/TCP (User Datagram Protocol/Transmission Control Protocol) termination, ICMP (Internet Control Message Protocol)), Layer 3 (ARP/NDP (Neighbor Discovery Protocol), IPv4/v6 (IP version 4 / version 6) forwarding & routing, IPv4 fragmentation and reassembly, VRF (Virtual Routing and Forwarding) for IPv4, IGMP (Internet Group Management Protocol) and multicast, basic IPSec),

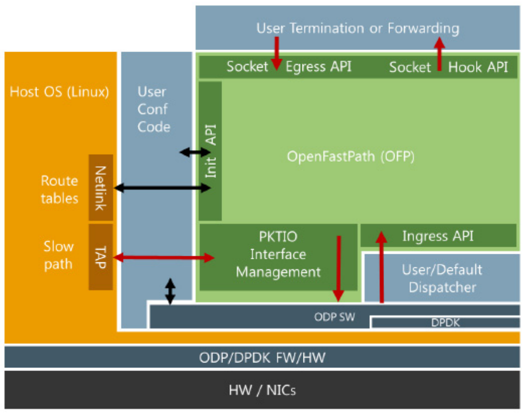

그림 7. OFP 시스템 구조[15]
Fig. 7. OFP system view[15]

Layer 2 (Ethernet, VLAN (Virtual Local Area Network)) and VxLAN (Virtual extensible LAN) and GRE (Generic Routing Encapsulation) tunneling. It also supports CLI (Command Line Interface) and configuration file.

The packet processing is handled through a series of self-contained processing functions (layer 2/3/4 protocols) as single thread run-to-completion environment. Thanks to its architecture (assigning cores for OFP dedicatedly in run-to-completion mode), its performance can be scaled up linearly by adding processing cores, showing much more performance than Linux kernel. In facts, it architecture, concept and operation behavior are very similar with 6WINDGate[16], the commercial version of network stack on top of DPDK.

It is still in incubation phase, meaning it may require certain customizations and optimizations to use it for commercial purpose. One of key contributors, Enea, does professional services for vendors who want to use it commercially. There are competitions against commercial stack vendors who have similar architecture and more mature products.

4.4 FD.io

FD.io (Fast Data input/output), as a Linux Foundation project, is a community with multiple projects in software-based packet processing towards the creation of high-throughput, low-latency and resource-efficient IO services suitable to many processor architectures (x86, ARM and PowerPC (Performance Optimization With Enhanced RISC - Performance Computing) and development environments (bare metal, VM (Virtual Machine), container)[17], founded in 2016. It uses DPDK for device driver layer to get packets to and from (v)NICs ((virtual) NICs) and threads/cores. Vector Packet Processing (VPP) library, donated by Cisco, is a key as the code in VPP is already running in commercial products and is modular, allowing easy plug-in without major changes to the underlying code basis and running in user space of Linux.

VPP is a data plane, consisting of a set of forwarding nodes arranged in a directed graph and a supporting framework. The framework has all the basic data structures, timers, drivers (and interfaces to driver kits like DPDK), a scheduler which allocates the CPU time between the graph nodes, and performance and debugging tools, like counters and built-in packet trace. The input node polls (or interrupt driven) an interface's RX queue for a burst of packets. It assembles those packets into a vector or a frame per next node, e.g. it sorts all IPv4 packets and passes those to the ip4-input node, the IPv6 packets into the ip6-input node and so on. When the ip6-input node is scheduled, it takes its frame of packets and processes them in a tight dual loop (or quad-loop) with prefetching to the CPU cache to achieve optimal performance. This makes more efficient use of the CPU cache by reducing misses, and scales efficiently for larger CPU caches. The ip6-input node pushes the various packets onto another set of next-nodes, e.g. error-drop if validation checks failed, or most typically ip6-lookup. The frame of packets moves like a train through the system until they hit the interface-output node and are shipped onto the wire, described in Fig. 8[18].

FD.io is in collaboration with other open source projects, such as DPDK for network I/O layer, OPNFV FastDataStacks project using VPP as a data plane forwarding component. Also, apart from the main VPP project, there are several adjacent projects under the FD.io umbrella: Honeycomb (ODL (OpeDayLight) integration), CSIT (Continuous System Integration and Testing), NSH SFC (Network Service Header, Service Function Chaining), ONE (Overlay Network Engine), VPP Sandbox, TLDK (Transport Layer Development Kit), package management, TRex (low-cost high speed stateful traffic generator), hICN (hybrid Information-Centric Networking)[18].

4.5 Netmap

Netmap has been suggested and implemented by Luigi Rizzo[19] in 2012, as a framework of fast and efficient packet I/O for both user space and kernel clients (programs, processes) without requiring custom hardware or changes to applications, showing 14.88 Mpps (Million packet per second) (i.e, the peak packet rate on 10Gbps) with one core (900 MHz). To achieve the high performance, it uses 1) preallocated linear and fixed size packet buffers when a device is opened, 2) removal of data copy costs (i.e, zero-copy transfer) by granting applications direct and protected access to packet buffers, 3) a lightweight metadata representation and 4) support of useful hardware features. User space applications can dynamically switch NICs into netmap mode, described in Fig. 9, and send and receive raw packets through shared memory buffers. It has similar scheme with that of DPDK as user space applications can talk to NICs directly, however, [20] insisted that it has less performance

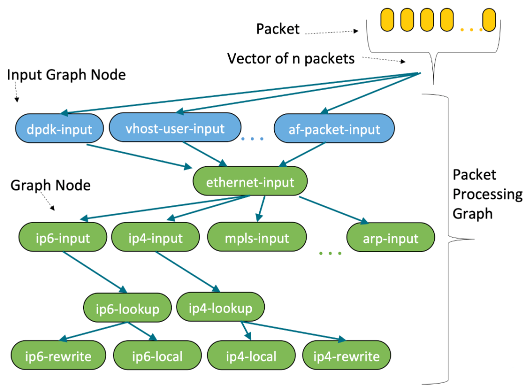

그림 8. VPP 구조:패킷처리 그래프[18]
Fig. 8. VPP architecture: packet processing graph of nodes[18]

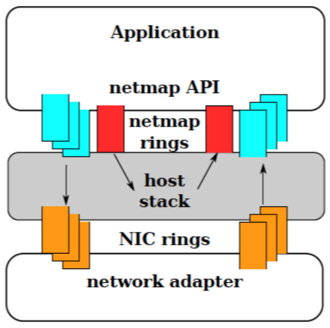

그림 9. Netmap 모드에서, NIC ring들은 호스트스택과 연결이 단절되며, netmap API를 통해 패킷을 교환된다. 호스트스택과의 통신은 두 개의 추가적인 ring들이 사용된다[19].
Fig. 9. In netmap mode, the NIC rings are disconnected from the host network stack and exchange packets through the netmap API. Two additional netmap rings let the application talk to the host stack[19].

than DPDK and requires more maintenance cost for NIC supports.

Netmap framework has been extended to virtualization supporting as ptnetmap[21] and its successor ptnet[22]. Ptnetmap is a netmap based virtual passthrough solution, which is similar with hardware passthrough concept, but allowing complete independence from the hardware and regarding the networking port exported to virtual machines as a software port not a piece of hardware. This concept can give benefits for live migration as the virtual machines are not limited to a specific hardware. Ptnet is a paravirtualized device model, like virtio[23]. It is based on netmap and its performance is comparable with that of virtio for traditional socket applications, having more portability[22].

4.6 PF_RING

PF_RING is a new type of network socket and user space framework that allows packet processing at high rates while providing consistent APIs for packet processing applications. It was developed by Luca Deri, a founder of ntop (www.ntop.org) with his paper[24] and its enhancement[25] called PF_RING DNA (Direct NIC Access). PF_RING polls packets from NICs using Linux NAPI which copies packets to PF_RING circular buffer ("ring"), then, the user space applications poll the ring to read the packets. It can distribute the incoming packets to multiple rings simultaneously for multiple applications, described in Fig. 10[26]. It consists of 1) kernel

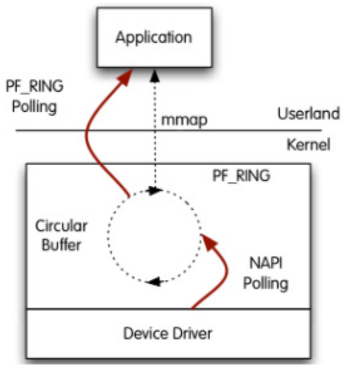

그림 10. 바닐라 PF_RING[26]
Fig. 10. Vanilla PF_RING[26]

module for low-level packet copying to the PF_RING circular buffer, 2) user space SDK (Software Development Kit) for user space applications, and 3) PF_RING aware device driver for additional improvements in packet capturing. It supports zero copy operation (called PF_RING ZC which is a commercial version of PF_RING), supporting full 10Gbps line rate processing (14.8 Mpps) with Xeon 2.5GHz. For the zero copy operation, packets are read directly from the network interface by bypassing the Linux kernel and the PF_RING kernel module. For this, NIC memory and registers are mapped into the user space, so the packet copy from NIC to DMA ring is done by NIC network process unit instead of NAPI. It is in similar position with that of DPDK at device driver level (i.e, doesn't have protocol stacks), however, as DPDK has much bigger name at the moment, it may focus on vertical applications specialized in packet capture and traffic analysis.

4.7 XDP (eXpress Data Path)

XDP, driven by an open source project called IO Visor[27] of Linux Foundation projects, is to accelerate the packet process inside kernel not bypassing it. The basic idea is not to replace the kernel stack but to provide simpler and faster alternative way in kernel, by kernel hooks using eBPF (extended Berkley Packet Filter: a highly flexible and efficient one in Linux kernel allowing to execute bytecode at various hook points in a safe manner. It can be executed at the lowest point of software stack) in collaboration with the existing kernel network stack.

The XDP packet process includes a kernel component that processes RX packet-pages directly out of driver via a functional interface without early allocation of kernel packet buffer (sk_buff) or software queues. Normally, one CPU is assigned to each RX queue, so, there is no locking RX queue, and the CPU can be dedicated to busy poll or interrupt model. BPF programs perform processing such as packet parsing, table lookups, creating/managing stateful filters, packet manipulation, etc[27,28]. The packets may be dropped, forwarded possible with packet modification (for NAT (Network Address Translation) for example), or locally received by Linux kernel, described in Fig. 11.

XDP's goal is to close the performance gap to kernel-bypass solutions (not intending to be faster than kernel-bypasses'), working in concert with Linux kernel stack along with all the benefits of BPF. It may not be used for most of network use cases, but for pre-stack processing (like filtering to support DDoS (Distributed Denial of Service) mitigation), forwarding and load balancing, batching techniques such as in software-based offloading like GRO (Generic Receive Offload), flow sampling / monitoring, etc.

Its performance has been evaluated in [29], which showed 24 Mpps with single core (while DPDK was 43.5 Mpps), however, comparing to kernel bypass technologies, it insisted XDP has certain benefits such as keeping kernel security and management compatibility, utilizing existing kernel stack features

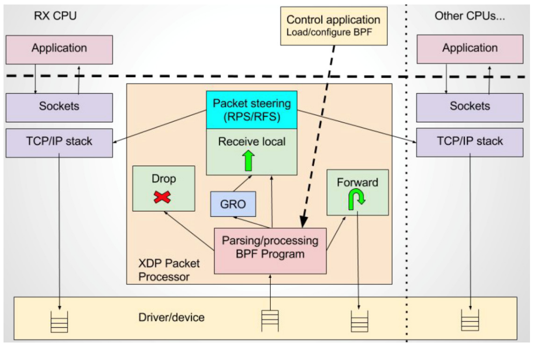

그림 11. XDP 패킷 프로세서[28]
Fig. 11. XDP packet processor[28]

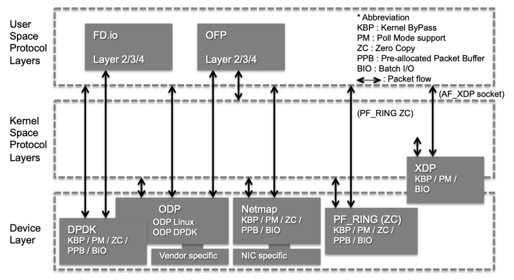

그림 12. 각 기술들간의 계위, 관계 및 패킷흐름
Fig. 12. Hierarchy, relationship and packet flow of the technologies

as needed, providing a stable programming interface with transparency to applications.

To support fast delivery of raw XDP frames into user space, XDP can bypass the Linux kernel network stack via XDP_REDIRECT into a special BPF-map containing AF_XDP socket which is a new address family type. The PMD for AF_XDP is available in DPDK[9].

The technologies mentioned from Section 4.1 to 4.7 is described in Fig. 12, upon their hierarchy and relationship with packet flow. It is also specified in the Fig. 12 for the major techniques (kernel bypass, poll mode support, zero copy support, pre-allocated packet buffer and batch I/O processing) used by the technologies.

4.8 Other technologies

PacketShader[6] is a high-performance PC-based software router platform that accelerates the core packet processing with GPUs, developed by KAIST in 2011. It offloads computation and memory-intensive router applications to GPUs while optimizing the packet reception and transmission path on Linux. It uses GPU for data parallel execution because GPUs are widely used for high-performance parallel applications whose workloads require enormous computation cycles and/or memory bandwidth[30]. It optimizes the packet I/O in Linux with huge packet buffer (instead of allocating metadata and packet data for each packet reception, PacketShader pre-allocates two huge circular buffers with a large array for metadata and packet data, resulting in greatly reduction of the memory allocation/deallocation overhead for high-speed packet reception), batch processing, NUMA-aware data placement and multi-core CPU scalability, while operating GPU accelerated packet processing in user space of Linux with modification of Intel-based NIC (82598/82599) drivers. The drivers bypass the kernel network stack, delivering packet to user space GPU acceleration framework. It was demonstrated close to 40 Gbps throughput of packet forwarding for all packet sizes, with two Xeon X5550 2.66 GHz for CPU and two NVIDIA GTX480 cards for GPU[6].

OpenOnLoad[31] is an implementation of TCP and UDP over IP which is dynamically linked into an application's address space and granted direct access to accelerated network hardware. The network stack interposes network operations from the applications and enables them to be handled completely at user space. In so doing, it bypasses the operating system and significantly improves performance through the removal of disruptive events such as context switches and interrupts which otherwise reduce the efficiency by which a processor can execute application code[31]. Solarflare Communications, Inc. has released its NICs, supporting application transparency with high performance. However, as it is a hardware based solution meaning their customers have to be equipped with the vendor's NICs, the customer space is likely limited.

NetSlices[32] is an operating system abstraction that processes packet in user space and enables a linear increase of performance with the number of cores. Spatial partitioning of hardware resources at coarse granularity is done in order to reduce interference and contention. Its APIs provide applications with fine-grained control of hardware resource. The streamlined path is also brought by NetSlice for packets to move between user space and kernel space. With spatial partitioning (i.e, allocating a RX/TX queue to specific core), network traffic is divided into "slices" and independent packet processing execution contexts allow parallelism and contention minimization. It requires a simple kernel extension which can be loaded at runtime to replace with conventional raw sockets. Contrary to most of other solutions, it does not take advantage of zero copy, but copies each packet once between the user space and kernel space, insisting it gains added portability and usability.

Sophos Ltd., a firewall and security company, presented their proposal and initial implementation for fast packet processing in Linux kernel, in NetDev 1.2[33]. It uses Netmap with enhanced networking stack along with pre-allocated rx/tx buffer, batch I/O capabilities and forward cache prefetching for fast packet processing in Linux kernel as Linux kernel already has the full network stack. It has implementation of fastpath networking stack in Linux kernel for high performance, letting the packets, which can not be processed in the fastpath networking stack in the kernel, be processed in normal Linux networking stack. It showed about 5 times more processing than standard Linux stack for their firewall application.

Snabb was started in 2012 by a free software hacker, Luke Gorrie, providing direct access to the high performance NICs but in addition to that it also provides an environment for building and running network functions[34]. It is a toolkit for developing network functions in user space mostly aimed for high performance networking with user space drivers for supported NICs, written in Lua, a high level programming language. It is composed of **Engine** (that runs network functions), **Libraries** (that ease of the development of network functions), **Apps** (reusable software components that generally manipulate packets) and **Programs** (ready-to-use standalone network functions). A network function in Snabb is a combination of **apps** connected together by links. The Snabb's engine is in charge of feeding the **app** graph with packets and give a chance to every **app** to execute. The first Snabb applications are L2 VPN (Layer 2 Virtual Private Network), IPv6 translation, L7 (Layer 7) firewall, etc. It provides a way to develop network functions easily with high performance using user space drivers, but, still looks in limited market space.

OPNFV **FastDataStack** project[35]. OPNFV, a collaborative project under Linux Foundation, is a project and community that facilitates a common NFVI (Network Function Virtualization Infrastructure), continuous integration with upstream projects (such as OpenDaylight, OpenStack, Kubernetes, Ceph Storage, KVM (Kernel-based Virtual Machine), Open vSwitch, Linux, DPDK, FD.io, etc.), stand-alone testing tool sets, and a compliance and verification program for industry-wide testing and integration to accelerate the transformation of enterprise and service provider networks. Goals include accelerating time to market for NFV (Network Function Virtualization) solutions, easing operational burdens, and ensuring the platform meets the industry's needs[36]. For data plane, OPNFV engages with OVS (Open vSwitch), FD.io and DPDK to address performance, scale and resiliency needs for carrier networks. There are lots of active projects under OPNFV and one of them is FastDataStacks. FastDataStacks was started in spring 2016, to meet the requirements for high performance data plane stack in NFV and virtualized applications, by building solution stacks supplied by FD.io along with functionality for realizing application policies and controlling a complex network topology. The key components are VPP (as a network forwarder) and HoneyComb for dataplane management agent from FD.io, virtual machine controller from OpenStack, a network controller from OpenDaylight, and installation and test suits from OPNFV. The scenarios have been developed upon [OpenStack – VPP] and [OpenStack – OpenDaylight – VPP] with various features and functionalities. Its integration, configuration and test details are in [35].

NetFPGA[37] is a project to develop open-source hardware and software for rapid prototyping of high performance network devices based on FPGA, started in 2007. There are three platforms available, NetFPGA-SUME, NetFPGA-1G-CML and NetFPGA-10G. A solution like NetFPGA SUME can reach the speed of upto 100 Gbps[38], insisting the rapid prototyping can be done with its ready-made reference and modular hardware and software components as flexible building blocks.

Smart NICs. One of the ways to accelerate the data plane processing is to hire additional hardware processing capability on top of standard NICs, providing certain functionalities and offloading them from the host system's CPU[39], which is called as a smart NIC. Although the definition and supporting capabilities of NICs may be different from NIC vendors and researchers, it is becoming an emerging solution as another approach to address the limitations of the kernel network stack. The functionalities of smart NICs can be developed with ASIC (Application Specific Integrated Circuit), SoC or FPGA, whose choice depends on the use cases of the NICs[40]. One of the examples for massive deployment in the cloud space is AccelNet, Microsoft's solution for offloading host networking to hardware using custom Azure smart NIC based on FPGA, and it showed 32 Gbps throughput with AccelNet (comparing it was 5 Gbps without it)[41]. A smart NIC (it was called as hardware-assisted NIC in the paper) was used for 5G (fifth generation) PoC (Proof od Concept) by Intel Corp. and SK Telecom Co., Ltd., to offload virtual switching and tunnel endpoint termination (VLAN, VxLAN, GRE) for network overlays in KVM virtualization environment[42]. There have been many studies and architecture proposals to use smart NICs to offload network functions mainly targeting cloud area, such as UNO[43], PANIC[44], AccelNet[41], etc.

Ⅴ. Conclusion

Linux has been a most widely used operating system for networking, though it was initially developed for general computing. Most of network equipment vendors have used Linux as it is open source and it has been evolved continuously with lots of contributors' efforts to integrate new features and solutions. That is why all of the solutions mentioned in this article have focused on Linux. The issue of general purpose operating systems, such as Linux, is that the throughput of the high bandwidth network interfaces, such as 10 Gbps, 40 Gbps or even more, can be hardly supported by the operating system's network stack, because the operating system is originally designed for general purpose not network specific. To resolve it, lots of studies and proposals have been conducted since early 2000.

DPDK has the biggest name in this area with lots of active contributors and users. The products based on DPDK solution are already commercially available with various forms, UTM (Unified Threat Management), mobile core equipment, router, etc. It can be, therefore, easily inferred that it will keep its position as one of main streams in this area for certain period of time.

Some solutions are based on hardware, like GPU or FPGA, however, the majority is based on software upon commodity hardware though there may be some limitations of software based solutions especially on NICs for whether or not to be supported. Although there are some issues in SDN (Software Defined Network) in terms of expandability and centralized architecture[45], it is expected that the way to SDN/NFV will be continued. In this environment, software solution can give lots of flexibilities for deployment options, it is likely the software's will keep the majority, letting the hardware's be on the specific vertical applications.

Most of solutions are based on kernel bypass, however, there are still efforts to enhance the kernel's, like XDP or Sophos as the kernel itself is proven to be secure, full featured, and transparent to applications. However, these solutions need to be kept and maintained as a part of Linux kernel. Without it, the users of the solutions may need to do it by themselves and it will limit the use of the solutions.

Virtualization environment is getting more popular and it became a common platform especially for the cloud area. The applications requiring high bandwidth traffic like VNF (Virtual Network Function) need to process traffic from high bandwidth NICs, such as 10Gbps or 40Gbps. It requires fast packet processing not only for VNF applications but also for hypervisor side. OVS (Open vSwitch) or Linux bridge can be a solution for software virtual switch of hypervisor, however, as long as these are based on Linux kernel, it can be a same bottleneck in hypervisor as well. To overcome the bottleneck of virtual switch, OVS-DPDK has been released since OVS 2.2. For I/O of virtual machines, the para-virtualization driver, called virtio, has been developed and widely used, in order to enhance the performance between virtual machines and hypervisor. Also, device passthrough solutions, like SR-IOV (Single Root I/O Virtualization), have been supported from major NIC vendors. Ptnetmap and ptnet in Section 4.5 can be comparable solutions for this area. The software based technologies, like DPDK, can be easily integrated into the virtualization environment. The VNF solutions based on DPDK already showed the line rate performance with multiple 10 Gbps NICs for both of bare-metal and virtualization environment (with its own virtual switch) a few years ago[46].

Most of solutions mentioned in this article have focused on how to handle packets quickly at low level (i.e, device driver level), but missing part would be how to process packets in protocol stacks of layer 2-4. Some solutions like OFP address it, but, still in early phase. FD.io can be a candidate as its VPP is already proven in the market, but FD.io is a comparably new project (founded in 2016).

The 5G is one of the hottest topics in mobile space and one of the main keywords of it is cloud native [47,48], implying 5G core components need to be developed and verified in a cloud native virtualization environment[47]. One of the feasible ways to go to 5G could be to port their products using commodity server at bare metal instead of their proprietary hardware, then, to migrate to virtualization environment. Some 5G players have already moved to this way with their vEPC (virtualized Evolved Packet Core) a few years ago[49,50]. Any equipment requiring high bandwidth traffic processing and low latency can be utilized with the technologies mentioned in this paper. In considerations of 5G equipment, EPC or 5GC (5G Core) is likely the one bottlenecked because they are supposed to process lots of traffic with complex protocols (GTP (GPRS: General Packet Radio Service) Tunneling Protocol, VxLAN, GRE, IPSec, etc) along with typical Layer 2/3 processing in data plane side, as a core equipment. As the traffic from control plane protocols is relatively much smaller than the data plane's, how to accelerate the data plane in a standard commodity server and general purpose operating system over cloud virtualization environment is a key factor for actual deployment. A smart NIC was used for 5G PoC a few years ago, to offload some complex protocols (VLAN, VxLAN, GRE) and virtual switching[42]. However, vendor dependancy especially for hardware limits the flexible deployment in a telecom operator's perspective, so, its use needs to be carefully considered. DPDK is one of the biggest and most active projects in this area supporting multiple CPU architectures with lots of techniques already implemented and it is market proven. It could be inferred that most of 5G suppliers are likely to use this technology in their products, though it is not mentioned in their product datasheets.

In this article, after checking the packet processing of general purpose operating system, focusing on Linux, and its issues, then, various techniques for fast packet processing have been checked. It has been, then, reviewed for the different types of data plane acceleration technologies with the packet processing techniques.

References

[1] I. Marinos, R. N. Watson, and M. Handley, "Network stack specialization for performance," in *Proc. twelfth ACM Workshop on Hot Topics in Network*, p. 9, ACM, 2013.

[2] T. Barbette, C. Soldani, and L. Mathy, "Fast userspace packet processing," in *Process. Eleventh ACM/IEEE Symp. ANCS '15*, IEEE Computer Society, pp. 5-16, 2015.

[3] D. Cerovic, V. D. Piccolo, A. Amamou, K. Haddadou, and G. Pujolle, "Fast packet processing: A survey," *IEEE Commun. Surv. Tuts.*, vol. 20, no. 4, pp. 3645-3676, 4thQuart., 2018.

[4] J. Garcia-Dorado, F. Mata, J. Ramos, P. Santiago del Rio, V. Moreno, and J. Aracil, "High-performance network traffic processing using commodity hardware," *Data Traffic Monitoring and Anal.*, 2013.

[5] J. H. Salim, "When NAPI comes to town," *Linux 2005 Conf., 2005.*

[6] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: A GPU-Accelerated software router," *ACM SIGCOMM Comput. Commun. Rev.*, Aug./Sep. 2010.

[7] G. Liao, Z. Zhu, and L. Bhuyan, "A new server I/O architecture for high speed networks," in *Proc. Symp. High-Performance Comput. Architecture*, 2011.

[8] Y. Go, M. A. Jamshed, Y. Moon, C. Hwang, and K. Park, "APUNet: Revitalizing GPU as packet processing accelerator," *14thUSENIX Symp. Netw. Syst. Design and Implementation*, Mar. 2017.

[9] https://www.dpdk.org/ecosystem/#support, (accessed in Sep. 2019)

[10] A. Yemelianov, "*Introduction to DPDK: Architecture and principles*," in https://blog.selectel.com/introduction-dpdk-architecture-principles/, Nov. 2016.

[11] *DPDK Programmer's Guide*, Rel. 19.08.0, http://fast.dpdk.org/doc/pdf-guides/prog_guide-master.pdf, (accessed in September, 2019)

[12] A. Burakov, "*Memory in DPDK, Part 1: General concepts*," article in https://www.dpdk.org/blog/2019/08/21/memory-in-dpdk-part-1-general-concepts/, Aug. 2019.

[13] *ODP (Open Data Plane)*, https://opendataplane.org

[14] *ODP User Guide*, https://opendataplane.github.io/odp/users-guide/, (accessed in Sep. 2019)

[15] *OFP (Open Fast Path)*, https://openfastpath.org, (accessed in Sep. 2019)

[16] "*6WINDGate – Architecture Overview,*" white paper downloadable in https://www.6wind.com/wp-content/uploads/2018/10/1-6WINDGate-Architecture-Overview-1.pdf

[17] *FD.io (Fast Data Input Output)*, https://fd.io, (accessed in Sep. 2019)

[18] *FD.io white paper for VPP*, https://fd.io/wp-content/uploads/sites/34/2017/07/FDioVPPwhitepaperJuly2017.pdf

[19] L. Rizzo, "netmap: A novel framework for fast packet I/O," in *Proc. 2012 USENIX Conf. Annu. Technical Conf.*, Jun. 2012.

[20] L. Hao, "*Embedded network architecture optimization based on DPDK*," PPT downloadable in https://www.dpdk.org/wp-content/uploads/sites/35/2018/06/DPDK-China2017-Lin-Telco-Data-Plane-Status.pdf

[21] S. Garzarella, G. Lettieri, and L. Rizzo, "Virtual device passthrough for high speed VM networking," *ACM/IEEE Symp. Architectures for Netw. and Commun. Syst. (ANCS)*, May 2015.

[22] V. Maffione, L. Rizzo, and G. Lettieri, “Flexible virtual machine networking using netmap passthrough,” *IEEE Int. Symp. Local and Metropolitan Area Netw. (LAN/MAN)*, Jun. 2016.

[23] R. Russel, “virtio: Towards a De-Facto standard for virtual I/O devices,” *ACM SIGOPS Op. Sys. Rev.*, vol. 42, no. 5, pp. 95-103, Jul. 2008.

[24] L. Deri, “Improving passive packet capture: Beyond device polling,” in *Proc. SANE*, 2004, downloadable in http://luca.ntop.org/Ring.pdf

[25] L. Deri, “ncap: Wire-speed packet capture and transmission,” in *IEEE End-to-End Monitoring Techniques and Serv.,* pp. 47-55, 2005.

[26] ntop, https://www.ntop.org/products/packet-capture/pf_ring/, (accessed in Sep. 2019)

[27] *IO Visor open source project*, https://www.iovisor.org/technology/xdp, (accessed in Sep. 2019)

[28] T. Herbert and A. Starovoitov, “*eXpress data path (XDP) : Programmable and high performance networking data path*,” presentation file downloadable in https://github.com/iovisor/bpf-docs/blob/master/Express_Data_Path.pdf

[29] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller, “The eXpress data path: Fast programmable packet processing in the operating system kernel,” in *CoNEXT ’18: Int. Conf. Emerging Netw. Experiments and Technol.*, Heraklion, Greece, Dec. 2018.

[30] http://shader.kaist.edu/packetshader/index.html, (accessed in Sep. 2019)

[31] S. Pope and D. Riddoch, “*Introduction to OpenOnload*,” white paper downloadable from http://www.moderntech.com.hk/sites/default/files/whitepaper/SF-105918-CD-1_Introduction_to_OpenOnload_White_Paper.pdf

[32] T. Marian, K. Lee, and H. Weatherspoon, “NetSlices: Scalable multi-core packet processing in user space,” in *Proc. Eighth ACM/IEEE ANCS’12*, Austin, TX, USA, Oct. 2012.

[33] N. Shah and J. Motvani, “*Linux forwarding stack fastpath*,” PPT downloadable in https://netdevconf.org/1.2/slides/oct7/03_Linux_Forwarding_Stack_Fastpath.pdf in Netdev 1.2, Tokyo, Japan, 2016.

[34] D. Pino, “*Snabb – A toolkit for user-space networking*,” presentation in FOSDEM 2018, PPT in https://www.slideshare.net/igalia/snabb-a-toolkit-for-userspace-networking-fosdem-2018

[35] *FastDataStacks*, https://wiki.opnfv.org/display/fds, (accessed in Sep. 2019)

[36] *Open Platform for NFV* (OPNFV), www.opnfv.org, (accessed in Sep. 2019)

[37] *NetFPGA*, https://netfpga.org/site/#/, (accessed in Sep. 2019)

[38] N. Zilberman, Y. Audzevich, G. Kalogeridou, N. M. Bojan, J. Zhang, and A. W. Moore, “NetFPGA – Rapid prototyping of networking devices in open source,” *SIGCOMM ’15*, Aug. 2015.

[39] K. Deierling, “*What is a SmartNIC?*,” blog in https://blog.mellanox.com/2018/08/defining-smartnic/, Aug. 30, 2018.

[40] K. Deierling, “*Achieving a cloud scale architecture with SmartNICs*,” blog in https://blog.mellanox.com/2018/09/why-you-need-smart-nic-use-cases/, Sep. 11, 2018.

[41] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, H. K. Chandrappa, S. Chaturmohta, M. Humphrey, J. Lavier, N. Lam, F. Liu, K. Ovtcharov, J. Padhye, G. Popuri, S. Raindel, T. Sapre, M. Shaw, G. Silva, M. Sivakumar, N. Srivastava, A. Verma, Q. Zuhair, D. Bansal, D. Burger, K. Vaid, D. A. Maltz, and A. Greenberg, “Azure accelerated networking: SmartNICs in the public cloud,” in *NSDI USENIXAssociation*, 2018.

[42] D. Lee, J. Park, C. Hiremath, J. Mangan and M. Lynch, “*Towards Achieving High Performance in 5G Mobile Packet Core’s User Plane Function*,” white paper in https://builders.intel.com/docs/networkbuilders/towards-achieving-high-performance-in-5g-mobile-packet-cores-user-plane-function.pdf, 2018.

[43] Y. Le, H. Chang, S. Mukherjee, L. Wang, A. Akella, M. Swift, and T. V. Lakshman, “UNO:

unifying host and smart NIC offload for flexible packet processing," *SoCC'17*, Sep. 2017.

[44] B. Stephens, A. Akella, and M. Swift, "Your programmable NIC should be a programmable switch," *HotNets-XVII*, Nov. 2018.

[45] J. H. Ryu, W. S. Kim, and C. H. Yoon, "A technical trend and prospect of software defined network and OpenFlow," *KNOM Rev.*, vol. 15, no. 2, pp. 1-24, Dec. 2012.

[46] *6WIND Speed Series Performance Validation in SDxCentral*, Sep. 2015, downloadable in https://www.6wind.com/wp-content/uploads/2016/03/Final-SDxCentral-6WIND-Speed-Series-Performance-Report.pdf

[47] Samsung Electronics Co., Ltd., "*5G Core Vision: Revolutionary changes in core with the arrival of 5G*," white paper in https://images.samsung.com/is/content/samsung/p5/global/business/networks/insights/white-paper/5g-core-vision/5G_Core_Vision_Technical_Whitepaper.pdf, 2019.

[48] Huawei Technologies, Co., Ltd., "*5G Network Architecture: A High-Level Perspective*," white paper in https://www.huawei.com/minisite/hwmbbf16/insights/5G-Nework-Architecture-Whitepaper-en.pdf, 2016.

[49] Press Release, https://www.thefastmode.com/technology-solutions/6757-samsung-sk-telecom-complete-poc-on-vepc-in-sdn-based-network

[50] Press Release, https://www.nec.com/en/press/201603/global_20160311_02.html

김 용 근 (Yongkeun Kim)

1988년 2월 : 아주대학교 전자계산학과 학사
1990년 2월 : 아주대학교 컴퓨터공학과 석사
1990~1998 : 쌍용정보통신(주) 선임연구원
1995년 : 전자계산조직응용기술사
1998년~2001년 : Lucent Technologies 부장
2001년~2002년 : Jetstream Comm. 이사
2003년~2018년 : 6WIND S.A. 부사장
현재 : 코리아퀘스트(주) 대표이사
2019년 9월~현재 : 동서울대학교 정보통신과 겸임교수
<관심분야> 네트워크가상화, 고속패킷처리, 네트워크보안
[ORCID:0000-0001-7930-5680]