Report

Namal University of Mianwali
Department of Computer Sciences

Network Packet Processing by Bypassing the Linux Kernel.

An fyp related to the field of networking especially deal with the packet processing on the receiving side.

By

Zohaib Aslam Khan Bscs201957
Saifullah Khan Bscs201916

Under Supervision

Supervisor: Dr. Muhammad Sheraz Anjum
Co-Supervisor: Dr. Adnan Iqbal

18-7-2023

Acknowledgments

We would like to convey our heartfelt gratitude to our Supervisor Dr Sheraz Anjum and Co-supervisor Dr.Adnan Iqbal for their tremendous support and assistance in the completion of our project.We also like to express our great appreciation to Mr Bacha-Rehman (HOD of BSCS) and all other professors of our department for their contributions to increase our knowledge during the whole journey of bachelor degree. WE would like to offer our special thanks to our family members and friends who support us and boost us to achieve something. In the end we would also like to extend our thanks to the staff of the VSI for their help in offering us this project and also the assistance when needed.

We would like to acknowledge that this project was completed entirely by us and not by some one else.

Muhammad Zohaib Aslam khan
Saifullah Khan

Abstract

Due to a significant increase in network traffic and the current limitations of the Linux kernel, particularly in the network stack, hinders the system's ability to achieve line speed packet processing. This presents a challenge in meeting the growing demands of high-performance network environments. Software optimization is one of the solution to enhance the packet processing capabilities of the system and overcome these bottlenecks. We are developing a system that will be able to process network packets at line speed, a capability currently not possible in the Linux kernel.The final outcome comes from the comparative analysis of different experiments performed throughout the entire process of finding optimized software and hardware. The technologies we are looking forward to developing efficient packet processing systems are XDP (Express Data Path) and DPDK (Data Plane Development Kit).

First of all, we aim to understand how these technologies work with our native Linux kernel. We are conducting experiments to determine which of them is more suitable for our requirements. An important part of this process is understanding and using different traffic generators like testpmd, iperf, and Trex. After conducting initial experiments on the default system settings and using built-in functionalities, we have come to the conclusion that the networking field encompasses many different scenarios, and these tools perform well in their respective contexts. Therefore, we are actively seeking different cases to determine the suitability of each new technology for a particular scenario.

We have implemented two different cases to demonstrate the efficiency brought about by both XDP and DPDK. To achieve this, we begin by reverse engineering the XDP and DPDK code. Subsequently, we develop applications, namely a router implemented in DPDK and a firewall implemented in XDP. We then perform various experiments using these applications to showcase the extent of the efficiency they bring to the overall system performance.

By continuously refining and optimizing our software and hardware, we strive to overcome the limitations of the Linux kernel and achieve line speed packet processing capabilities. This ongoing research and development process holds promising prospects for revolutionizing network systems, enabling more efficient and responsive packet processing in diverse networking scenarios.

Contents

1 Introduction **1**
1.1 Problem . . . 1
1.2 Methodology . . . 1
1.3 Goals . . . 2

2 Literature Review **3**
2.1 Introduction . . . 3
2.2 The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel[3] . . . 3
2.3 Fast Packet Processing: A Survey [2] . . . 3
2.4 Linux Kernel Packet Transmission Performance in High-speed Networks[1] 4

3 Background **5**
3.1 CPU . . . 5
3.2 NUMA . . . 5
3.3 DMA/DCA . . . 6
3.4 PCIe . . . 6
3.5 Ques . . . 7
3.6 Struct . . . 7

4 Related Work **8**
4.1 Traffic generation . . . 8
4.2 Profiling . . . 9

5 XDP **10**
5.1 Network Hooks . . . 10
5.2 Kernel's Networking Layers . . . 10
5.3 BPF . . . 11
5.4 EBPF . . . 11
5.5 EBPF System . . . 11
5.6 EBPF Programs . . . 12
5.7 How and When Are eBPF Programs Executed? . . . 12
5.8 Program Types . . . 12
5.9 Maps . . . 12
5.10 Return Codes . . . 13
5.11 Basic Program Structure . . . 13

6 DPDK **16**
6.1 How can DPDK access User Space ? . . . 17

7 VPP **21**

8 Methodology **23**
8.1 Setting Up Topology . . . 23
8.2 Traffic Generation . . . 24

8.3 Build Application 24
8.4 Packet Processing 24
8.5 Comparative Analysis 25

9 Experimentation 27
9.1 Drop Packet performance with 1 Gb NIC 27
9.2 Software Router 29
9.3 Drop Packet performance with 10 Gb NIC: 36

10 Limitations and Challenges 37
10.1 Technical Complexity: 37
10.2 Steep Learning Curve: 37
10.3 Limited Documentation and Resources: 38
10.4 Hardware and Platform Dependencies: 38
10.5 Performance Optimization Challenges: 38
10.6 Debugging and Troubleshooting: 38

11 Conclusion and Future Work 38
11.1 Conclusion 38
11.2 Future Work 39

A Appendices 40
A.1 References 40

1. Introduction

The exponential growth of data traffic, which is not expected to stop anytime soon, brought about a vast amount of advancements in the networking field. Latest network interfaces support data rates in the range of 40Gbps and higher. To ensure the capabilities of the operating system to produce or receive a given amount of data, we need to assess them through the help of network testing tools. There are two main categories of network testing tools: software and hardware based. Hardware network testing tools are usually seen as accurate, reliable and powerful in terms of throughput but expensive on the other side. While software-based testing might in fact be less trustworthy than hardware-based, it has a tremendous advantage of malleability.
Modifying the behaviour of the software is easily realized, on the other hand it is not only complex in the case of hardware but also likely to increase the price of the product and usually impossible for the consumer to tamper with as they are commonly proprietary products. There is no better solution between the two, it is a different approach to the same problem and hence testing a system from both perspectives if possible shall be recommended. However in this document we will focus solely on software testing as we did not have specialised hardware.

1.1 Problem

The interfaces which can support data rates of 40Gbs and higherdoes not guarantee higher packet processing speeds which are limited due to the overheads imposed by the architecture of the network stack. Nevertheless, there is a great need for a speedup in the forwarding engine, which is the most important part of a high-speed router. For this reason, many software based and hardware-based solutions have emerged recently with a goal of increasing packet processing speeds. An operating system's networking stack is conceived for general purpose communications rather than high-speed networking applications.

1.2 Methodology

Changing different parameters, protocols, and applying these techniques on various hardware configurations, including taking into account hardware limitations, are essential steps in conducting experiments. By repeatedly experimenting with different schemes, collecting statistics, and performing comparative analyses, we can observe the variations in packet processing performance across different systems. This process enables us to identify the best technology that outperforms others and holds promise for further development. However, solely relying on technology is insufficient; we must also focus on optimizing various aspects to achieve speeds comparable to the line rate. In cases where technologies like XDP (Express Data Path) and DPDK (Data Plane Development Kit) do not meet our requirements, it becomes necessary to explore alternative technologies that can better fulfill our needs. This iterative and exploratory approach ensures that we continually seek innovative solutions to maximize system performance and meet evolving network demands.

1.3 Goals

1.3.1 Technologies

We will dive into these technologies(XDP and DPDK) and look for better one so here is the small introduction to the boths.
Dpdk is kernel bypass approach, which hands control of a NIC to user-space programs to greatly reduce the overhead introduced into the kernel by things like context switching, networking layer processing, and interruptions. When you're working at higher networking speeds (10Gbps or higher), this becomes relevant.

Those limitations are exactly why XDP has become the darling of high-performance networking. With this method, user-space programs will be allowed to directly read and write to network packet data and make decisions on how to handle a packet before it reaches the kernel level. In other words, user-space takes care of some of the overhead, so the bulk of these decisions and actions are placed solely on the shoulders of the kernel.[?]

1.3.2 Parameters/Factors

Changing different parameters, protocols, and applying these techniques on various hardware configurations, including taking into account hardware limitations, are essential steps in conducting experiments. By repeatedly experimenting with different schemes, collecting statistics, and performing comparative analyses, we can observe the variations in packet processing performance across different systems. This process enables us to identify the best technology that outperforms others and holds promise for further development. However, solely relying on technology is insufficient; we must also focus on optimizing various aspects to achieve speeds comparable to the line rate. In cases where technologies like XDP (Express Data Path) and DPDK (Data Plane Development Kit) do not meet our requirements, it becomes necessary to explore alternative technologies that can better fulfill our needs. This iterative and exploratory approach ensures that we continually seek innovative solutions to maximize system performance and meet evolving network demands.

1.3.3 Hardware and Software Performance

When it comes to software and hardware, unlocking their full performance potential that they promise us relies on thorough testing and experimentation. Along with measuring the system overall performance increase during the experiments we are able to understand the capabilities and limitation of the hardware and software. For example we can only use that CPU's for the processing of packets that are in the same NUMA node containing the NIC. Another example is that, the network only allows us to send packets of a maximum size of 1500 bytes and a minimum size of 64 bytes, this limit the size of packets we can use during experimentation.

2. Literature Review

2.1 Introduction

Networking, also known as computer networking, is the practice of transporting and exchanging data between nodes(mostly computer systems) over a shared medium in an information system.The exponential growth of data traffic, which is increasing day by day ,this increase brought about a vast amount of advancement in the networking.Considering our area of interest we have latest network interfaces support data rates in the range of 40Gbps and higher. This, however, does not guarantee higher packet processing speeds which are limited due to the overheads imposed by the architecture of the network stack. Their are different methods and tools are available in the market to meet the needs. The goal is simple, as we knew performances reached by the NICs are now extremely high, we need to know if the system that they are supposed to be used with are capable of handling the load without any extra products or libraries.

We have gown through many papers/ project reports who have worked on packet processing capability of system by different techniques. Mains of them are follow

2.2 The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel[3]

This paper gown through performance analyses of three technologies i.e., Linux Kernel, DPDK and XDP. Conclusively XDP is prominent over the other bypassing technique because XDP offer all other features that always compile with networking. These features include retaining kernel security and management compatibility; selectively utilizing existing kernel stack features as needed; providing a stable programming interface; and complete transparency to applications. Main reason of its prominence is its almost non bypassing behavior with Linux, it works within Linux caring about its security and other functions.

DPDK performance is very decent but its load on system makes it incompatible with low specs systems, it totally bypasses Linux Network stack and some of features like security come in doubt.

XDP is still evolving and there are numbers of interesting parts to work on.

2.3 Fast Packet Processing: A Survey [2]

In this paper, researchers investigate different types of packet processing on different server-class network hosts and try to optimize processing by software, hardware or their hybrid. Survey of software solution include batch processing, parallelism and zero-copy techniques. Hardware solutions are discussed in terms of CPU usage , how they are used and their latency. But in the end due to interfaces like 40/ 100Gb/s these all solutions are not able to achieve max because of network stack architecture. After working on all types of parameters optimization, network stack always becomes bottle neck. Due to this reason bypassing network stack functionality has become the last choice for high-rate packet processing.

2.4 Linux Kernel Packet Transmission Performance in High-speed Networks[1]

2.4.1 Introduction of document

In the start paper introduce us with this thing that availability of the high speed NICs are easily available in the market, different tools (especially packetgen) and how the kernel interacts with them. During the problem part it questioned the already given solution of introducing a new NAPI, which has some certain advantages (increase the performance) but also have some of the disadvantages like bulking of packets to defer usual actions to the groups rather than per single packet. The methodology is the main part which is similar and about which we are gone talk during this analysis or review. They are doing the experiments by modifying certain parameters to assess the impact of the given parameter over the final result.

2.4.2 Hardware and software

Going into deep details of the system was necessary to interpret the results and hence a great part of this paper was dedicated to understanding various software/hardware techniques and technologies and their working with the Linux operating system. For example, by examining different bottlenecks like the speed of the a PCIe bus or the maximum theoretical throughput on an Ethernet wire. During the hardware details its talk about the CPU, CPU's caches, Symmetric Multiprocessing (involves two or more processing units on a single system which will run the same operating system, share a common memory and I/O devices, e.g. hard drives or network interface cards), NUMA Nodes (Non Uniform Memory Access is a design in SMP architecture which states that CPUs should have dedicated spaces in the memory which can be accessed much faster than the others due to its proximity). DMA (Direct Memory Access is a technique to avoid having to make the CPU intervene between an I/O device and the memory to copy data from one another), PCIE (Peripheral Component Interconnect Express usually called PCIe is a type of BUS used to attach components to a motherboard) and last was the Ethernet. In the software side it discusses the socket buffers, reference counters, NIC drivers and how they handle the packet sending and receiving.

2.4.3 Traffic generator

Then it discussed the different traffic generators like iperf, Kute, PF_RING, Netmap and DPDK. Then there is a brief discussion about the PKTGEN, its commands, transmission algorathims and its working. For result calculation we and they used concept of profiling which is getting records of a system in which they used the perf (It is based on the notion of events, which are trace points that perf pre-programmed inside the kernel). Ebpf which make the packet capturing efficient is also described and its use in the experiments is discussed.

3. Background

This section is dedicated to all those parameters/factors and the technical terms related to them which are necessary to interpret result (like the speed of the a PCIe bus or the maximum theoretical throughput on an Ethernet wire.). firstly we will dive into terms related to hardware than come to software to evaluate hardware's performance.

3.1 CPU

CPU is central processing unit of a system as it execute all the instruction stored in the system.

CPU Caches are a part of the CPU that store data which is supposedly going to be needed again by the CPU. An entry in the cache table in called a cache line. When the CPU needs to access data, it first checks the cache, which is directly implemented inside the CPU. If the needed data is found, it is a hit, otherwise a miss. In case of a miss, the CPU must fetch the needed data from the main memory, making the whole process slower.
In principle, the size of the cache needs to be small. For two reasons, the first one being the fact it is implemented directly in the CPU, making the lack of space an issue, and secondly because the bigger the cache, the longer the lookup therefore introducing latency inside the CPU.
Multi-level caches are a way to counteract the trade-off enforced by the cache size to table lookups issue. There are different levels of caches, which are all "on-chip" meaning on the CPU itself.

1. The first level cache, abbreviated L1 Cache will be small, fast, and the first one to be checked. Note that in real-life scenarios, this cache is actually divided in two caches: the one that stores instructions and one that stores data.
2. The second level cache, abbreviated L2 Cache will be bigger than the L1 cache, about 8 to 10 times more storage space.
3. The third and last level cache, abbreviated L3 Cache is much larger than the L2 cache however this characteristic vastly varieties on the price of the CPU. This cache is not implemented in all brand of CPUs.

Moreover L3 caches have the particularity of being shared between all the cores, which leads us to the notion of Symmetric Multiprocessing.
CPU load is very one of the main parameters in choosing better technology so we will investigate its by **mpstate** tool.

3.2 NUMA

Non Uniform Memory access is one of design in SMP (symmetric Multiprocessing). It state that CPU should have dedicate space in memory that can be accessed faster than other. This is done by segmenting the memory and assigning a specific part

of it to a CPU. CPUs are joint by a dedicated BUS (called the QPI for Quick Path Interconnect on modern systems). The memory segmented for a specific CPU is called local memory of the CPU. If it needs to access another part of the memory than its own, it is designated as remote memory, since it must go through a network of BUS connections in order to access the requested data.
this technique resolve the issue of memory access in SMP architecture, as single bus between multiple cpus can cause latency. A NUMA system is sub divided into NUMA nodes, each numa node represent its local memory and its dedicated CPU. however for cpu to access remote memory (from other node's local memory) can prompts the latency.
For Optimized performance our dedicated NICs and CPU cores must be working in same NUMA node so that latency can be minimised and packet processing can be maximized.

3.3 DMA/DCA

Direct Memory Access was introduced to increase read and write speed of both I/O devices and CPU. According to this technique NIC directly access memory without any participation of CPU. This technique has improved the performance in great manners but Recent I/O technologies such as PCI-Express and 10Gb Ethernet enable unprecedented levels of I/O bandwidths in mainstream platforms.
However, in traditional architectures, memory latency alone can limit processors from matching 10/25/40 Gb inbound network I/O traffic. So DCA (Direct cache Access) technique has significant reduction in memory latency. CPU and I/O devices directly read and write on L3 Cache which is quickly accessible by CPU. New technologies like **DPDK** are working with this technology.

3.4 PCIe

Peripheral Component Interconnect Express usually called PCIe is a type of BUS used to attach components to a motherboard. It was developed in 2004 and as of 2016, its latest release version is 3.1 but only 3.0 product are available. A new 4.0 standard is expected in 2017. PCIe-3.0 (sometimes called Revision 3) is the most common type of BUS found among high-speed NICs; because other standards are in fact too slow to provide the required BUS speed to sustain 40 or even 10 Gigabit per second if the amount of lanes is too little.
Bandwidth To actually understand the speed of PCI-e BUSes we must define the notion of "transfer", as the speed is actually given in "Gigatransfers per seconds" in their specification . A transfer is the action of sending a bit of data on the channel, however it does not specify the amount of bit sent because one needs the channel width to compute it, in other words without the amount of bits sent in a transaction, we can not calculate the actual bandwidth of the channel.
Circumventing the complex design details, on the PCIe version 1.0 and 2.0 an 8/10b encoding is used. This forces to send 10 bit for an 8-bit data transfer, implying an overhead of 1 10 8 = 0:2transfer.
The 3.0 revision uses a 128b/130b encoding, limiting the overhead to 1 128 130 0:015Now that we know the channel width, we can calculate the bandwidth B:

B = T ransfers (1 overhead) 2
The table 2.1 holds the results of the bandwidth calculation. However using a bandwidth less large than the theoretical throughput of a NIC will function (if enough lanes for the device), but it will result in a packet throttling because of the BUS speed.

Version	1.1	2.0	3.0
Speed	2.5 GT/s	5 GT/s	8 GT/s
Encoding	8/10	8/10	128 /130
Bandwidth 1x	2 Gb/s	4 Gb/s	7.88 Gb/s
Bandwidth 4x	8 Gb/s	16 Gb/s	31.50 Gb/s
Bandwidth 8x	16 Gb/s	32 Gb/s	63.01 Gb/s
Bandwidth 16x	32 Gb/s	64 Gb/s	126.03 Gb/s

Table 2.1 PCIe Speeds

3.5 Ques

The queuing system in Linux is implemented through an abstraction layer called Qdisc. Its uses ranges from a classical fifo algorithm to more advanced QoS-aimed queuing (e.g. HTB or SFQ). Though those methods can be circumvented if one user-level application fires multiple flows at once.
Driver queues are the lowest-level networking queue that can be found in the OS. It directly in- teracts with the NIC through DMA.

3.6 Struct

Socket buffers or SKBs are single-handedly the most important structure in the Linux networking stack. For every packet being present in the operating system, a SKB must be affiliated to it in order to store its data in memory. It has to be done in kernel space, as the interaction with the driver happens inside.
The structure sk buff is implemented as a double linked list in order to loop through the different SKBs easily.

.

4. Related Work

4.1 Traffic generation

4.1.1 iperf3

iperf3 is user-space tool made to measure the bandwidth of network due to its design it can not achieve high packet speed because of need of using system call to interact with the lower interface like NIC drivers. It might use zero copy option to make the packet content access faster. It can measure latency using UDP packets. You must have to run two intense on both side to establish client-server connection.

4.1.2 TREX

Rex is an open source, low cost, stateful and stateless traffic generator fuelled by DPDK. It generates L3-7 traffic and provides in one tool capabilities provided by commercial tools.

TRex Stateless functionality includes support for multiple streams, the ability to change any packet field and provides per stream/group statistics, latency and jitter.

Advanced Stateful functionality includes support for emulating L7 traffic with fully-featured scalable TCP/UDP support.

TRex Emulation functionality includes client side protocols i.e ARP, IPv6, ND, MLD, IGMP, ICMP, DOT1X, DHCPv4, DHCPv6, DNS in order to simulate a scale of clients and servers.

It is based on the DPDk. Due to its Totally bypass design from kernel, it can achieve high packet speed.TRex can scale up to 200Gb/sec with one server.

4.1.3 DPDK (Testpmd)

Testpmd is one of the reference applications distributed with the DPDK package. Its main purpose is to forward packets between Ethernet ports on a network interface and simulate traffic patterns to evaluate the performance of the DPDK framework and the underlying hardware. It provides a set of interactive commands and options that enable users to configure packet forwarding rules, set up traffic generation patterns, and monitor various statistics related to packet processing.

Forwarding Modes:

Testpmd has different forwarding modes that can be used within the application

Input/Output mode: This mode is generally referred to as IO mode. It is the most common forwarding mode and is the default mode when TestPMD is started. In IO mode a CPU core receives packets from one port (Rx) and transmits them to another port (Tx). The same port can be used for reception and transmission if required.

Rx-only mode: In this mode the application polls packets from the Rx ports and frees them without transmitting them. In this way it acts as a packet sink.

Tx-only mode: In this mode the application generates 64-byte IP packets and transmits them from the Tx ports. It doesn't handle the reception of packets and as such acts as a packet source.
These latter two modes (Rx-only and Tx-only) are useful for checking packet reception and transmission separately.

4.2 Profiling

Profiling is getting records of a system (or several systems) called the profile. It is commonly used to evaluate the performances of a system by estimating if certain parts of the system are being too greedy/slow, e.g. taking too much CPU cycles for its operations compared to the rest of the other actions to be executed. We will only pay attention to Linux profiling, as the entire subject was based on this specific OS, and therefore will talk about techniques that might not be shared among other commonly used operating systems (e.g. Windows or BSD based).//

The tools like mpstate, perf and PCM (Process Control Monitor) are used to measure the CPU utilization, cache hit miss ratio and PCIE Utilization. During the experiments some libraries of the XDP and DPDK also provide system insights which also help us to see the different information required because in some of the cases these tools bypass the user space and user can not use the utilities that are provided by the linux kernel. The large number of experiments generate a huge amount of data and on the basics of that generate some results. To get useful insights we use some standards approaches of data yielding and evaluation like

- Reproducibility
- Granularity
- Interpretation

The details of above mention approach are given in the comparative analysis which is subpart of the Methodology part.

Cutting edge Technologies

5. XDP

The importance of eBPF and XDP is highlighted by its fast adoption since its introduction in the Linux kernel in 2014 by both industry and academia. Their use cases have grown rapidly to include tasks such as network monitoring, network traffic handling, load balancing, and operating system insight. Several companies already use eBPF on projects such as Facebook [26], Netronome, and Cloudflare. Before diving deep into the to the xdp we have introduction to the different concepts related to xdp and ebpf.

5.1 Network Hooks

Hooks are used for intercepting packets before the call or during execution in the operating system in the filed of computer networking. We can enable data collection and custom event handling by attaching the eBPF programs to the hooks exposed by the Linux kernel.Many hook points are available in the Linux kernel. We can mainly focus on the eXpress Data Path(XDP) in the networking subsystems. can be used to process packets close to NIC on both RX and TX, enabling the development of many network applications.

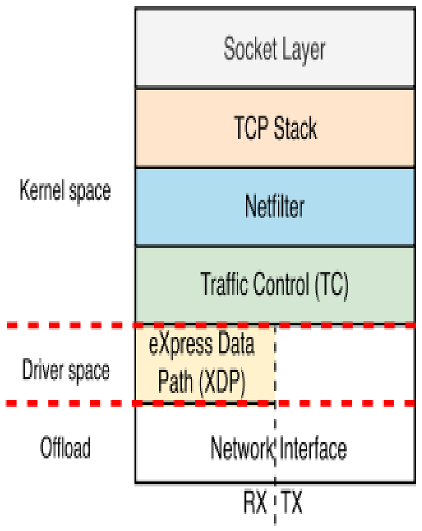


Figure:5.1

5.2 Kernel's Networking Layers

Packets entering the OS are processed by several layers in the kernel. These layers are socket layer, TCP stack, Netfilter, Traffic Control (TC), the eXpress Data Path (XDP), and the NIC. Packets destined to a userspace application go through all these layers and can be interceptedand modified during this process by modules such as iptables, which resides in the Netfilter layer. As explained before, eBPF programs can be attached to several places inside the kernel, enabling packet mangling and filtering.

5.3 BPF

BPF(Berkeley Packet Filter) Inspired by the works that were done on packet filters in the history bpf was proposed by Steven McCanne and Van Jacobson in 1992. There are 2 factors which are fundamental for the good performance of the ebf.
1:- The ability of bpf to run program in the kernel proved to be a good design choice. In the start the bytecode of an application was transferred from the user space to the kernel. Then it was checked to assure security and prevent kernel crashes. After passing the verification,the program attached to a socket by the system and ran on each arriving packet.
2: The Just-In-Time(JIT) compilation engine for bpf in the kernel is another highlighting factor.

5.4 EBPF

The modified version of the BPF is EBPF(Extended Berkeley Packet Filter)

eBPF provides an instruction set and an execution environment that can be used to program the kernel network layer that processes packets closer to the NIC mostly for fast packet processing inside the Linux kernel. It modifies the processing of packets in the kernel and also used for the programming of network devices.

—

The eBPF's instruction set architecture (ISA) was updated to include function calls. Those calls follow the C calling convention. Parameters are passed to functions through registers, just as it happens in native hardware. This allows mapping an eBPF function call to one hardware instruc- tion, which results in almost no overhead. eBPF uses this feature to enable helper functions, allowing programs to make system calls and manipulate storage (maps).
—-

5.5 EBPF System

An eBPF program is written in a high-level language (mainly restricted C). The clang compiler transforms it into an ELF/object code. An ELF eBPF loader can then insert it into the kernel using a special system call. During this process, the verifier analyzes the program and upon approval the kernel performs the dy- namic translation (JIT). The program can be offloaded to hardware, otherwise it is executed by the processor itself. –
XDP is the lowest layer of Linux network stack [31]. It enables developers to install programs that process packet into the Linux kernel. These programs will be called for every incoming packet. XDP is designed for fast packet processing applications while also improving programmability.

5.6 EBPF Programs

Packet filtering, performance analysis, traffic classification and several other type of application can be implemented with ebpf. The offering of a flexible and safe programmable environment inside the Linux Kernel is the main advantage of the ebpf system. Because we know that ebpf programs can be loaded and modified during run time. The aare also capable of interacting with kernel elements such as kprobes, perf events, sockets and routing tables.

5.7 How and When Are eBPF Programs Executed?

We attach the ebpf program to an interface(that allows the custom programming) to execute the program when required. The interface is called the Hook. The registration of certain events can be allowed by the hooks. Here we are looking for the kernel XDP hook to which an ebpf program can be attached.‘ ebpf program executes when that event occur which is registered in parallel to the program. So in the computer networking , common events are receiving and sending packets but we are mostly focused on the receiving.

5.8 Program Types

By seeing the ebpf program type , we can determine the three important aspects, first its input that has been passed to it, second is which helper function it can used and third is the hook to which it is attached.for example socket filter and tracing, sk-skb and xdp are types of ebpf programs.
1:- Socket filter program to perform socket filtering.
2:- Socket tracing program can trace an event like an tcp connection started.
3:- SK-SKB program to access socket buffers and socket parameters (IP ad-dresses, ports, etc) and to perform packet redirection between sockets.
4:- XDP program to be attached to the eXpress Data Path hook

5.9 Maps

eBPF programs have generic key-value stores that are called maps. Keys and values in the maps are treated as binary blob, which allow the storage of the user-defined data structures and tyes and their sizes must be defined during creation of the maps.
Multiple maps can be created by user-space process and they can be accessed by both eBPF programs loaded in the kernel and user-space processes enabling data exchange between the two environments. There are 24 valid different map types.
.... continue

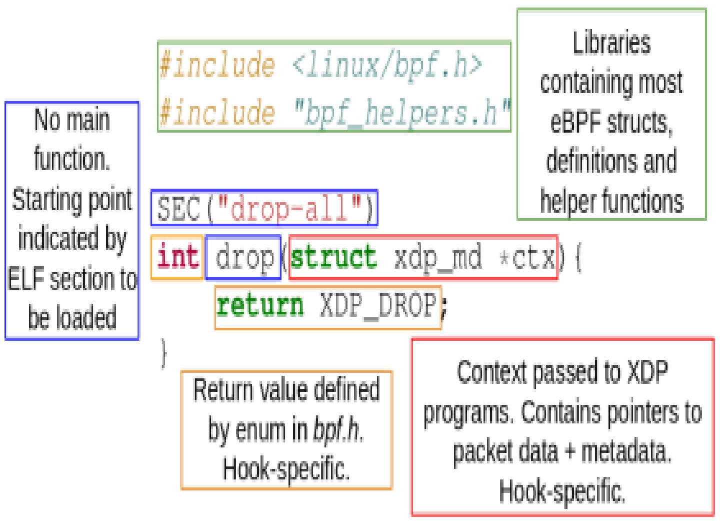
5.9.1 Helper Function

The ability to call helper function maek the eBPF different from the BPF. These are special functions offered by the kernel infrastructure to enable interaction with the context of each hook and other kernel facilities and structures, such as maps, routing tables, tunneling mechanisms, and so on. Interacting with maps, modifying packets, and printing messages to the kernel trace are the tasks that are performed by the helper function.

5.10 Return Codes

The program types of the eBPF determines the value and meaning of the return codes by eBPF programs. lets see this example to understand, an xdp returns a decision about what should happen to the packet after the processing (like pass along, drop ,redirect etc) which is defines by xdp_action in the bpf.h.

5.11 Basic Program Structure



```
#include <linux/bpf.h>
#include "bpf_helpers.h"

SEC("drop-all")
int drop(struct xdp_md *ctx) {
        return XDP_DROP;
}
```

Libraries containing most eBPF structs, definitions and helper functions
No main function. Starting point indicated by ELF section to be loaded
Return value defined by enum in *bpf.h*. Hook-specific.
Context passed to XDP programs. Contains pointers to packet data + metadata. Hook-specific.

Figure:5.2

Figure illustrates the basic structure of an eBPF program. It presents a simple XDP program that drops all received packets. The library linux/bpf.h has all struct and constants definitions used by the eBPF programs, except for specific subsystems such as Traffic Control (TC) and perf, which need extra header files. As a rule of thumb, all eBPF programs should include this file.

Now lets back to the business of the xdp

As we know that eXpress Data Path (XDP) is the lowest layer of the Linux kernel network stack. It is present only on the RX path, inside a device's network driver, allowing packet processing at the earliest point in the network stack, even before memory allocation is done by the OS. It exposes a hook to which eBPF programs can be attached
In this hook, programs are capable of taking quick decisions about incoming packets and also performing arbitrary modifications on them, avoiding additional overhead imposed by process- ing inside the kernel. This renders the XDP as the best hook in terms of performance speed for applications such as mitigation of DDoS attacks. After

processing a packet, an XDP program returns an action, which represents the final verdict regarding what should be done to the packet after program exit.

Value	Action	Description
0	XDP_ABORTED	Error . Drop packets
1	XDP_DROP	Drop packets
2	XDP_PASS	Allow further processing by the kernel stack
3	XDP_TX	Transmit from the interface it came from
4	XDP_REDIRECT	Transmit packet from another interface

Table 5.1 XDP_Actions

***XDP_Actions* :** The above Table lists all possible XDP actions and their description. The action is specified on the basis of program return code, which is stored at register right before the eBPF program exits. The first four actions has simple return value with no parameters to intake , in which first indicate the packet should be dropped while raising an exception (XDP_ABORTED),secondly drops the package silently using (XDP_DROP), third passed packets along to the kernel stack (XDP_PASS) or immediately re-transmitted the packet through the same interface (XDP_TX).

The XDP_REDIRECT action allows an XDP program to create three alternative scenarios in which it redirect packets to (1) another NIC (physical or virtual), (2) another CPU for further processing, or (3) an AF_XDP socket for userspace processing.This action requires a parameter to specify the redirection target so it is different from the others . Two helper functions bpf_redirect() or bpf_redirect_map() help us to do that thing. The former is focused on network devices and receives the target's interface index. The latters a more generic alternative, which performs lookups on an auxiliary map to retrieve the final target, which can be both net devices or CPUs. The second option is recommended, on the basics of the providence of the much better performance if compared to bpf_redirect() by batching packet transmits, and it also offers better flexibility, as map entries can be modified dynamically from user and kernel spaces.

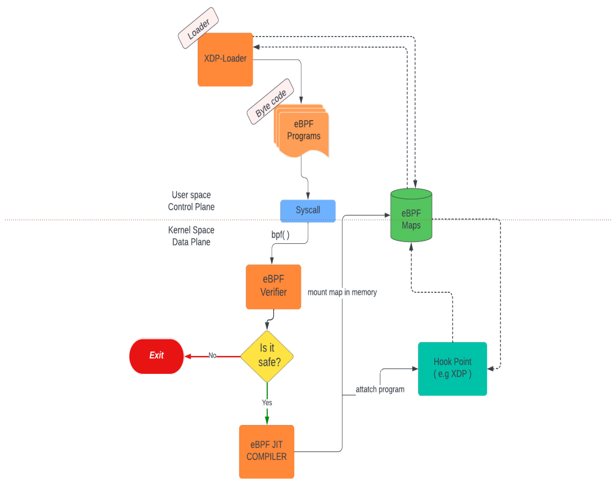


Figure:5.3 : XDP_Flow Diagram

XDP Modes of Operation **:** eBPF programs are attached to the XDP to alter the packet processing at the device driver level, which requires explicit support by the associated network driver to increased performance

Some drivers already have such functionality for high-speed devices such as i40e, nfp, mlx* and the ixgbe family.XDP programs are executed directly by the driver, even before these are handled by the operating system. This is defined as ***XDP NATIVE*** mode. Too see the an up-to-date list of XDP-enabled drivers go to the BCC Project.

However, ***XDP GENERIC***, the compatibility mode offers by the the kernel, which enables XDP program execution at the driver level for devices without native support. On this mode, Operating system itself do the XDP execution , emulating native execution. This way can have programs attached to the devices even they do not have explicit XDP support.

Socket buffer allocation extra steps required to perform the emulation cause the reduced performance. When loading the eBPF program system automatically chooses between these two modes. Once loaded,using the ip tool, it is possible to check the mode of operation.

XDP OFFLOAD is yet another mode of operation. As the name suggests, the eBPF program is offloaded to compatible programmable NICs ,if compared to the other two modes it achieves even greater performance when loading the program.

6. DPDK

DPDK (Data Plane Development Kit) is set of libraries for implementing totally user space drivers for Network Interface Controllers. It Provides s with a framework and Common APi for achieving higher performance. API for high speed network application and allow us for achieving fast packet processing.

DPDK is an open source project is being managed by Linux Foundation, with a large development community that are working on its improvement. it has been designed to provide a simple. yet complete framework for fast Packet Processing.

How does DPDK WORK?

Dpdk Provide a set of data plane libraries and networking Interface controller Polling drivers. DPDK make it possible to communicate between user space and NIC without involvement of Kernel. Networking application can run faster because native kernel networking stack is very slower, can't keeps up with emerging hardware throughput.

DPDk has five core components: the EAL (Environment Abstraction Layer for access to low level resources, like core, memory), the MBUF (a data structure that carries network packets), the MEMPOOL (the library for creating memory for packet allocation), the RING (the library manage messages between threads, cores etc.) and the TIMER for asynchronistic callback functions).

Why use DPDK

DPDk enables more efficient computing of packet processing than the standard Linux kernel interrupt processing. It eliminate inttrupts over heads and uses PMD (polling mode driver) that continuously monitors the NIC activity. It can caused more cpu cycle consumption but performance with high bandwidth NICs is very smooth.

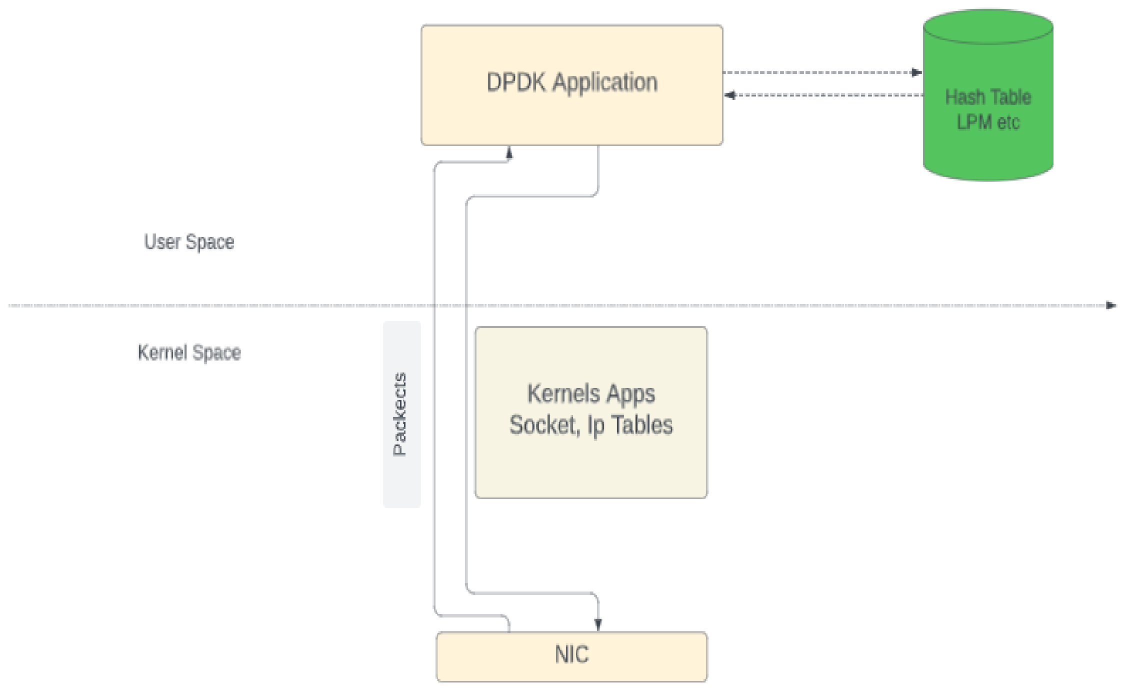


Figure:6.1: DPDK FLOW Diagram

6.1 How can DPDK access User Space ?

Before going into deep of DPDK, look at NIC drivers work[5]. How user access hardware and what are its gains and lost.

Linux networking software Stack

Any user application which wants to communicate with other node in internet it has to establish TCP/UDP connection. This is acheived by Socket API.

In case of TCP, Internally SYScalls are used:

- Socket() - craet a endpoint and return file descriptor
- bind() - bind to local ip address and port

There are many other calls , call to each of these invoke a context switch, which eats up precious memory and time of system. During context switch, following events occur:

- user space (socket API) call kernel function
- CPU state of user application is stored in memory
- CPU privilege mode changed, privilege mode is required to access ip tables etc.
- CPU state of kernel restored in memory
- kernel function invoked
- after completion, context re switch to user space and all step happened again in reverse.

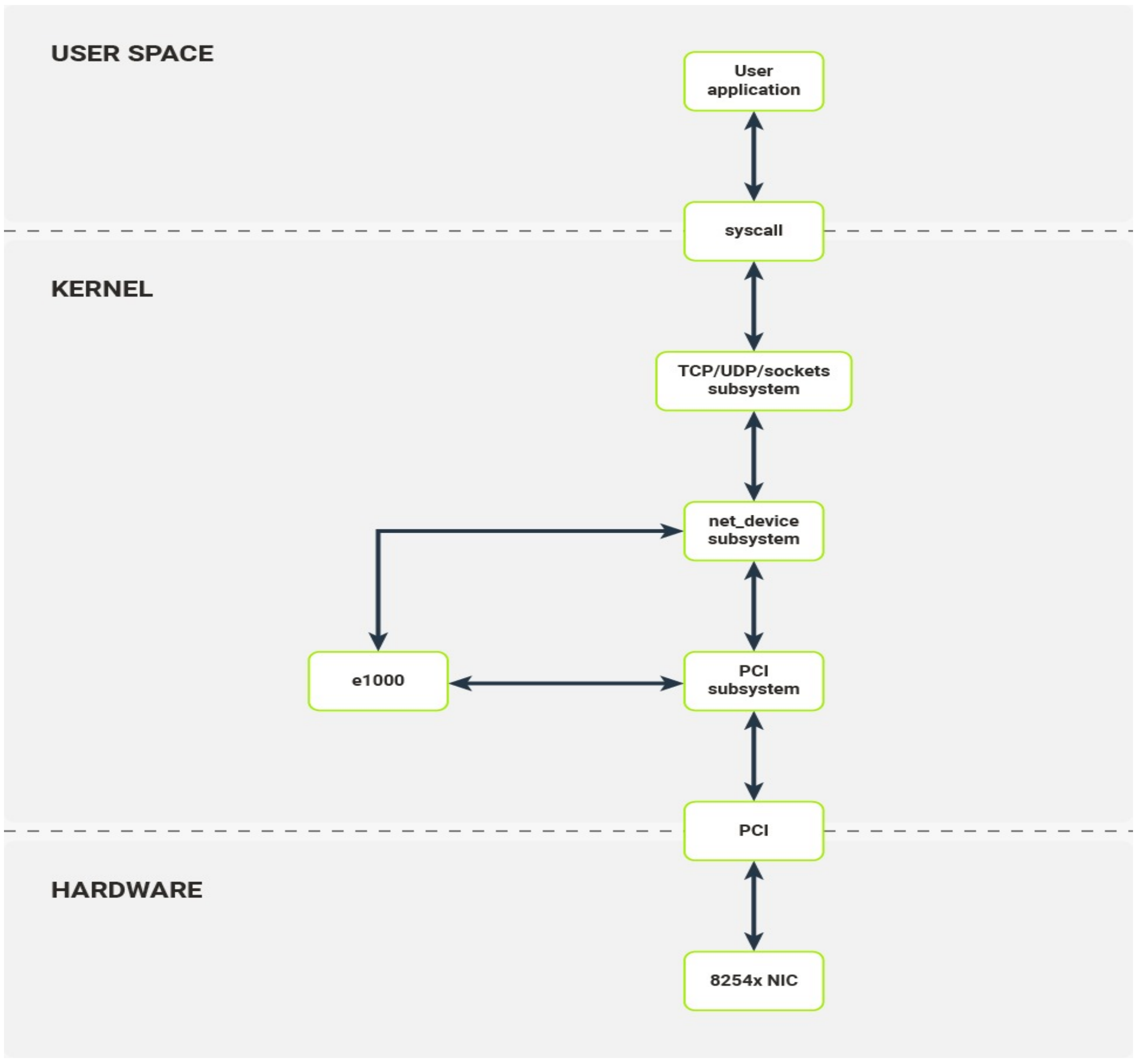


Figure:6.2: NIC to DPDK

Let assume we wants to send data, before data goes onto the wire it is gone through different layers of Kernel. Kernel Socket API maker a buffer called sk-buff, which is used to contain packet s and their metadata. Depending upon network layer and Transport layer, socket API pushes sk-buff data structure to underlying protocol sending data implementation. These implementation add IP address and port like information to data. After higher layers, sk-buff pushed toward lower layers i.e netdev subsystem. The netdev system allow any module to register as network device, layer 2 device. Module has to configure device's offload and transmit packet. When ready for transmission, sk-buff moved to netdev's TX callback. After receiving TX request from upper layers, NIC driver signal the device that there is new packet to be sent and notifies upper layers when transmission has done.

User Space

In kernel, interface and user application communicate by kernel's PCI subsystem and memory management system. But in user space, this can be exposed by utilizing UIO subsystem.

UIO (user space input/output) is a kernel module responsible for user space abstraction in communication of user process with hardware. To use UIO for PCI, we need to use uio-pci-generic drivers. this process requires a unbinding of device from kernel drivers like e1000, ixgbe etc. and bind device to uio-pci-generic drivers. Now our user process is able to communicate with hardware directly.

To properly implement a pocket processing , we have to define RX/TX descriptors buffers. Each buffer has pointers to actual allocated packet's buffer. As Linux kernel guarantees that the physical location of data will not be changed at run time so huge pages can be used safely.

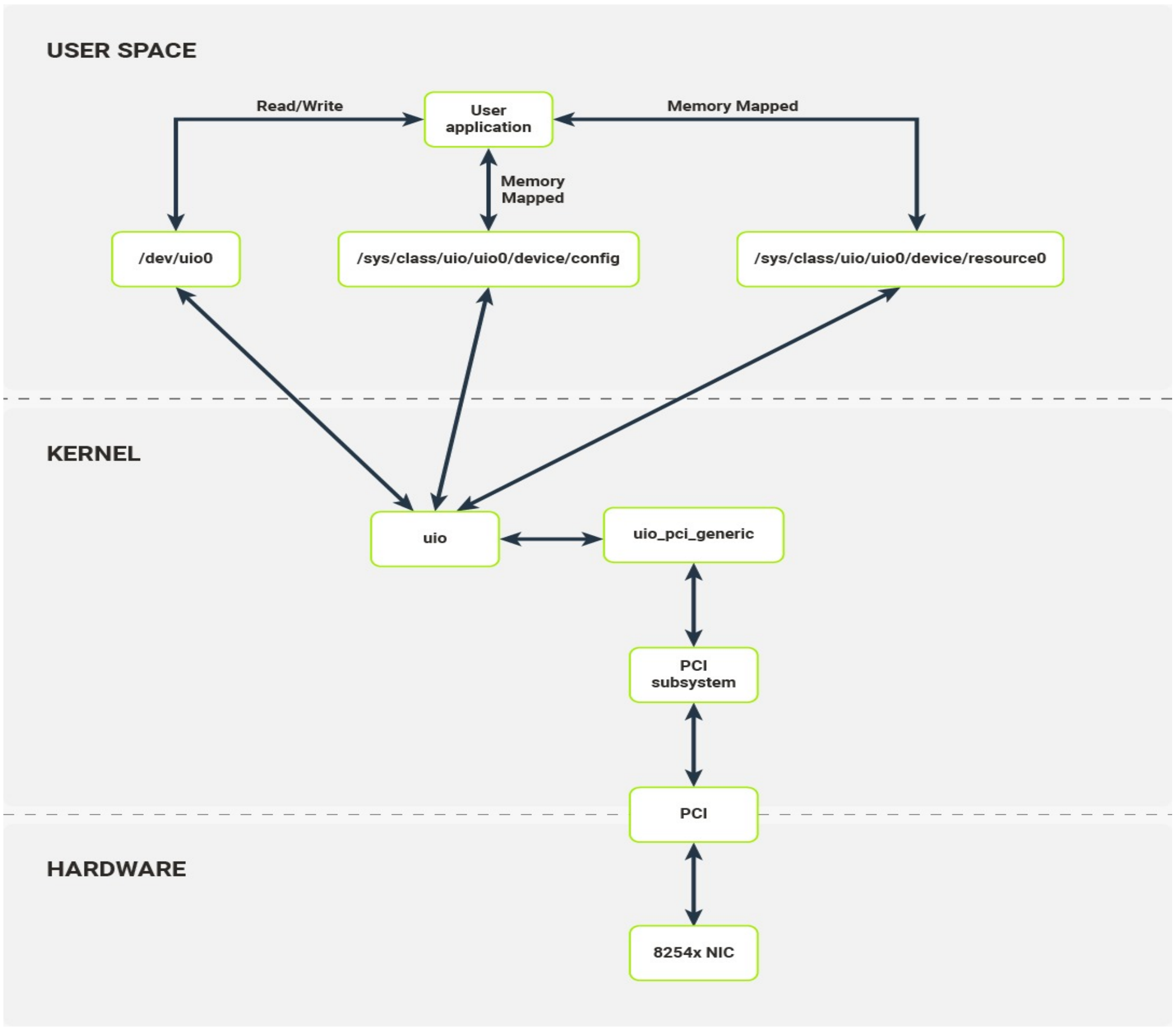


Figure:6.3 Userspace

Huge Pages

on x86 architecture, default physical pages size is 4KB. Huge pages introduction can solve two problem

- First, Reduce the number of page table entries required to represent large chunks of physical memory. This reduce TLB usage. TLB is Translation Look-aside Buffer in cache for page table entries.
- Second, Reduce overall size of page Table

Depending upon hardware support huge pages cab be 2MB or 1GB in size.

7. VPP

VPP (vector packet processing) is a high performance packet-processing stack that is part of Linux and FD.io. Regardless of any software or hardware packet processing is generally sequence of operation that are performed on packet[4].

For example, packet received at NIC will go through all layers sequentially. Firstly header1, header2 and go on. To understand the idea of VPP compare it with scalar processing.

In scalar processing, each packet process separately. Every time when new packet received at NIC, CPU needs to fetch data and instruction in cache for every header processing and these all steps will be repeated for each new packet. Cache hit/miss is one of main factor in efficient consumption of resources. While in VPP each packet is not processed separately. Instead group of packet called vector processed at the same time. let assume Vector size is 512 packets, so first header of all these packets will be processed collectively than moved to next header. This increase the cache hit ratio.

The key characteristics of VPP frame work are the following:

- It can be run ass Linux user-space process
- can be deployed at VM, bare metal and containers.
- use DPDK drivers, PMD

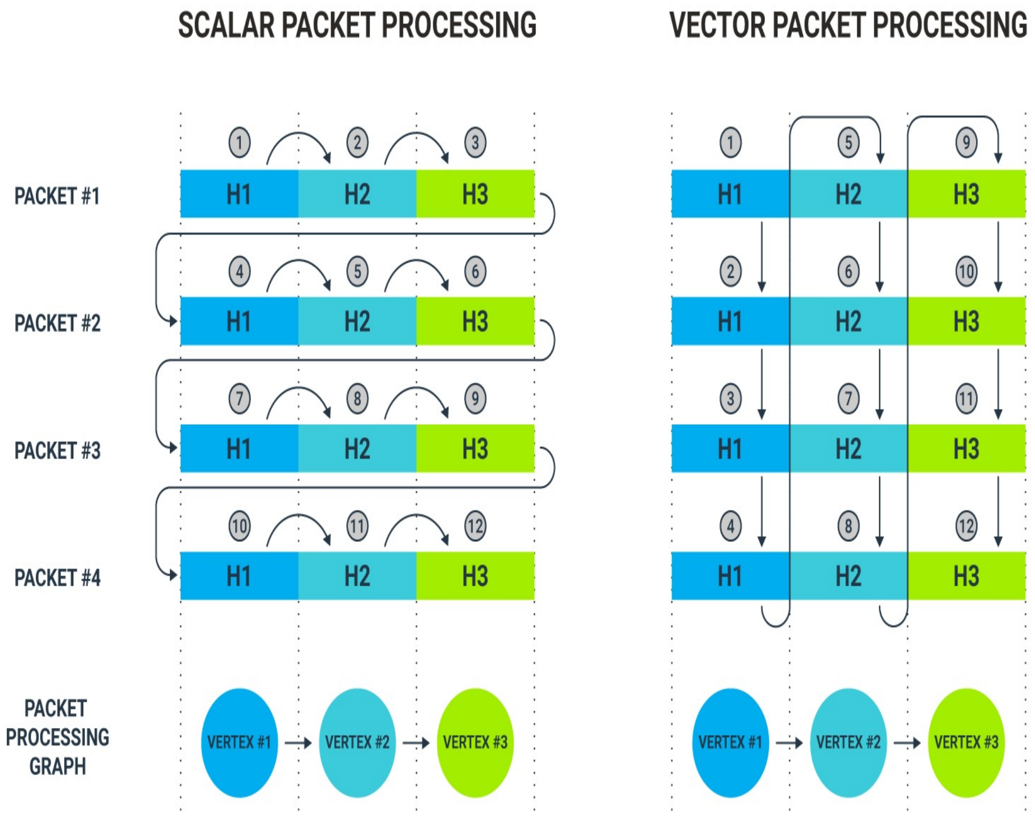


Figure:7.1

Although VPP (Vector Packet Processing) is an impressive technology used by Cisco in its routers and switches, we did not explore it as part of this project. VPP is known for its high-performance packet processing capabilities and has gained significant recognition in the networking industry. However, our focus for this particular project was on XDP and DPDK, both of which offer their unique advantages for programmable packet processing in the Linux kernel. While VPP is not directly involved in this project, we acknowledge its merits and its widespread adoption by industry-leading companies like Cisco.

8. Methodology

Throughout our project, we have go through extensive testing which allows us to investigate the impact of various traffic conditions on network performance. By conducting thorough research and collecting relevant measurements, we ensure that our methodology setup operates as intended and produces reliable data for analysis.

By carefully implementing methodology, we can generate controlled traffic scenarios and accurately measure the performance on the other side. This knowledge contributes to a deeper understanding of network behavior and aids in the development of improved network processing techniques.

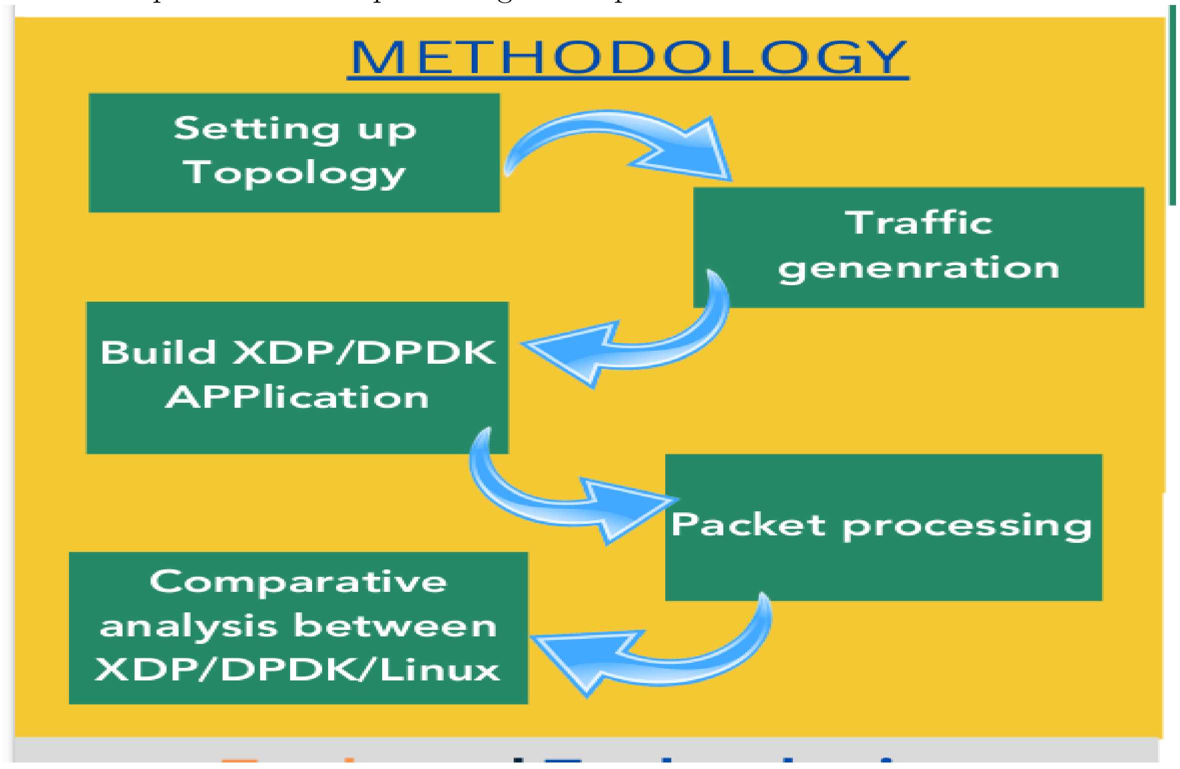

METHODOLOGY
Setting up Topology
Traffic genenration
Build XDP/DPDK APPlication
Packet processing
Comparative analysis between XDP/DPDK/Linux

Figure 8.1

8.1 Setting Up Topology

We have designed and implemented a network topology to facilitate the generation and reception of traffic between two computers. This network configuration plays a important role in our analysis of packet processing performance of the system. Our network topology consists of two computers directly connected through PCIE cables enabling traffic generation from one computer and receiving on the other on which we are performing different packet processing operations along with varying different parameter. The network diagram illustrates the layout of topology, connections, and Devices involved in this setup.

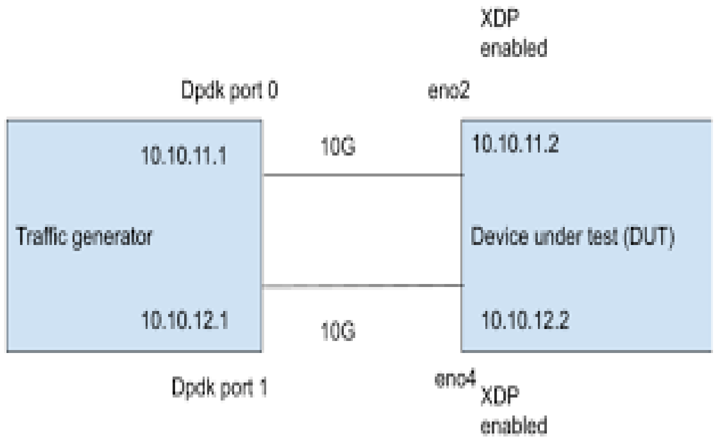


Figure8.2: General topology we used in our experiments.

The IP addressing and subnet configurations in our topology ensure proper communication between the computers and facilitate the routing of generated traffic. This helps maintain the integrity and reliability of the traffic flow within the network.

8.2 Traffic Generation

One computer serves as the traffic generator, and we have employed specific tools and configurations to simulate traffic flow. This setup allows us to analyze the behavior of the system under various traffic conditions and study its impact on performance metrics.Various traffic conditions are generated by adjusting parameters and settings, we can control the type, volume, and characteristics of the generated traffic.
We worked on 3 different traffic generator which include Iperf, DPDK's Trex and Testpmd. But in the end we go with the Trex because

Implement a server-client model that operates in a similar manner to Iperf.
Better than other because it works also with MAC topology as it is required because in DPDK we completely bypass the Network stack.

8.3 Build Application

In Linux we have builtin tools like IP Tables which make it possible and very easy to do stuff, we want with packet. Like firewall and routers are configured with commands. But in XDP and DPDK, we have to built application from scratch. These both are system level applications means they are working directly with hardware and kernel, so these are built in C language.

8.4 Packet Processing

On the receiving computer or called it DUT(device under testing), we have established the necessary configurations for XDP and DPDK and utilities to capture, analyze the incoming traffic and also measure the system performance. This enables us to measure and evaluate performance metrics such as latency, throughput, packet loss, CPU usage cache hit to miss ratio, memory utilization and PCIE traffic speeds. By closely examining, we gain valuable insights into the network's behavior and identify potential bottlenecks or issues.

8.4.1 Overview of Packet Receiving at DUT

We must understand how packet reception works to better analyze network bottlenecks and performance issues.The receive path is where frames are often lost so packet reception is important in network performance tuning.A significant penalty to network performance when the packets are lost in the receive path.After receiving each frame the linux kernel subjects it to a four step process:

***Hardware_Reception* :**The network interface card (NIC) receives the frame on the wire. Depending on its driver configuration, the NIC transfers the frame either to an internal hardware buffer memory or to a specified ring buffer.

***Hard_IRQ* :** The NIC asserts the presence of a net frame by interrupting the CPU. This causes the NIC driver to acknowledge the interrupt and schedule the soft IRQ operation.

***Soft_IRQ* :** This stage implements the actual frame-receiving process, and is run in softirq context. This means that the stage pre-empts all applications running on the specified CPU, but still allows hard IRQs to be asserted.

In this context (running on the same CPU as hard IRQ, thereby minimizing locking overhead), the kernel actually removes the frame from the NIC hardware buffers and processes it through the network stack. From there, the frame is either forwarded, discarded, or passed to a target listening socket.
When passed to a socket, the frame is appended to the application that owns the socket. This process is done iteratively until the NIC hardware buffer runs out of frames.

***Application_receive* :** The application receives the frame and dequeues it from any owned sockets via the standard POSIX calls (read, recv, recvfrom). At this point, data received over the network no longer exists on the network stack.

8.5 Comparative Analysis

8.5.1 Data yielding:

During the experiments the ability to create a lot of data without having to constantly monitor the execution of it became quickly a need. The principle is to create convenient methods that could automatically variate the parameters, more or less brutally according to the needs and store the results, along with the experiments settings, into a file to be post-processed into human-understandable data; e.g. plots. The solution was to create a program that would take as parameters the setting(s) to variate, the step- ping and the limit to be reached.

8.5.2 Data evaluation:

The data acquired was created through empirical testing, adjustment and tuning to the fit the situation. The data acquired must follow several rules in order to be kept as a final result of this work:

Reproducibility: the experiment must yield the same results by being ran over the same settings. While it may sound obvious, a lot of data has been discarded after several hundreds tests due to the behaviour not being exactly reproducible. It also does not necessarily mean that the results are bogus but is either due to bad measuring or that the anomaly is spurious and would take too much time pinpointing.

Granularity: As this thesis focused mostly on high performance, a single byte might or might not change the outcome of an experiment. Therefore the experiments were first run with an average stepping; meaning with settings variation large enough to end a set of experiments in a reasonable amount of time (e.g. few hours) but small enough to reduce an anomaly to a particular range of settings. Of course finding this trade-off has also been part of the work and required experimenting.

Interpretation: to ensure that the results are correctly interpreted, extra profiling tests were always ran to be certain that they are not being compromised by another program that would conflict in any way.

.

.

9. Experimentation

:

In this section, we present the experimental findings and comparative analysis of a series of carefully conducted experiments on XDP and DPDK. By comparing the results under different conditions, we aim to understand how different factors affect system performance. The comparative analysis helps us identify patterns and draw meaningful conclusions about the impact of these variables. Through this investigation, we aim to contribute to existing knowledge and provide practical insights for further research in the field of packet processing.

Procedure:

1. Connected Computer A and Computer B using Ethernet cables and fiber optics.
2. Configured the IP addresses of Computer A and Computer B in the same subnet.
3. Run the packet processing application on Computer B to measure throughput.
4. Start the experiment by sending packets from Computer A to Computer B with varying packet sizes (e.g., 64 bytes, 128 bytes, 256 bytes, and 512 bytes).
5. Record the latency measurements from the network analyzer tool and the throughput values from the packet capture tool for each packet size.
6. Repeat the experiment multiple times to ensure the reliability of the results.
7. Perform necessary calculations and data analysis to interpret the findings.

.

9.1 Drop Packet performance with 1 Gb NIC

Initial experiments are being held in our local environment. we have connected our two system with LAN cable. Due to the Limitation of hardware i.e 1 Gigabit Ethernet card, we are only able to circulate 940 Mb/s it is only possible when we increase the size of the packets which in results decrease the number of packets we are sending per sec. But we are looking forward to increase the number of packets so we compromise the total amount of traffic. As we know the minimum data size for the packet is 64 bytes.

We conducted a small test to compare the performance of native Linux system, DPDK and XDP packet processing. We have performed the operation of packet drop. Here are the results of the experiment which includes that how much cpu is used when we altered the packet sizes in each scenario.

The traffic is low(less than 1 Gb) so we can handle all packet processing using a single core. Increase in packet size decrease numbers of packet, so CPU consumption reduce to minimum. s

Linux

Ip table is most common way to drop the packets.

packet size	cpu usage
64 bytes	76%
512 bytes	32%
1500 bytes	9%

Table 9.1.1

Only tuning done till now is increase in size of RX ring of Ethernet

DPDK

packet size	cpu usage
64 bytes	100%
512 bytes	100%
1500 bytes	100%

Table 9.1.2

Dpdk always use 100% CPU even there is no pocket processing.

XDP

packet size	cpu usage
64 bytes	60%
512 bytes	29%
1500 bytes	8%

Table 9.1.3

We used xdp in SKB mode because native and offloading modes are not supported by our device drivers. SKB mode performance is lower than other modes.

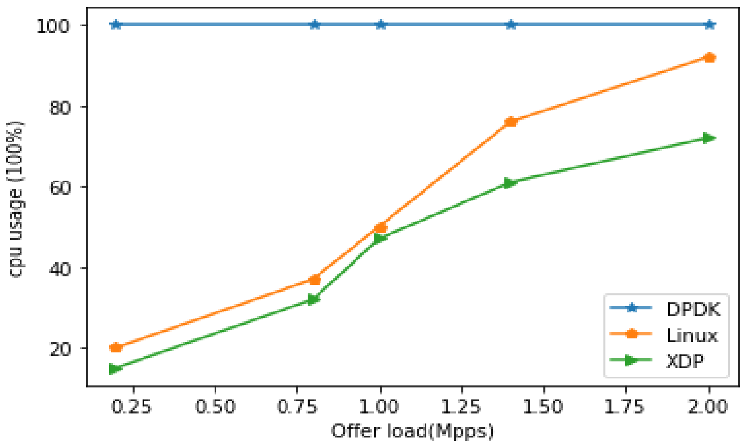


Figure 9.1: CPU usage for each scenario using 1 core only. Each method stop at its max while DPDK always show 100%, because it totally hijack all resources it given.

9.2 Software Router

<u>Linux:</u>

Kernel bypass networking offers the potential for better performance compared to traditional networking Linux stack. However, before we can measure the actual performance increase, it is essential to establish a baseline. In this test, we will begin by setting up a benchmark to measure the performance of the network using **standard kernel-based networking techniques**. This baseline measurement will serve as a reference point for comparing the performance improvements achieved through kernel bypass networking.

Test Setup:

Test1: Static Packet forwarding

In the first test, we performed a straightforward setup. we sent packets from the IP address 10.10.11.1 to 10.10.12.1 and vice versa, using the DUT (Device under Test) as the intermediary for routing the packets. The DUT had two interfaces, eno2 and eno4, through which the packets were transmitted.

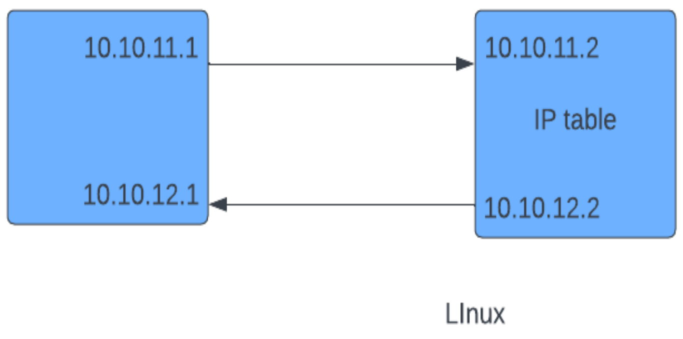


Figure:9.2

Furthermore, we performed these tests using both a single flow and with a larger scale of 10,000 flows, allowing me to evaluate the performance and behavior under different traffic loads.

IP commands:

```
iptables -I FORWARD -d 10.10.12.1 -s 10.10.11.1 -j to 10.10.12.2
iptables -I FORWARD -d 10.10.11.1 -s 10.10.12.1 -j to 10.10.11.2
```

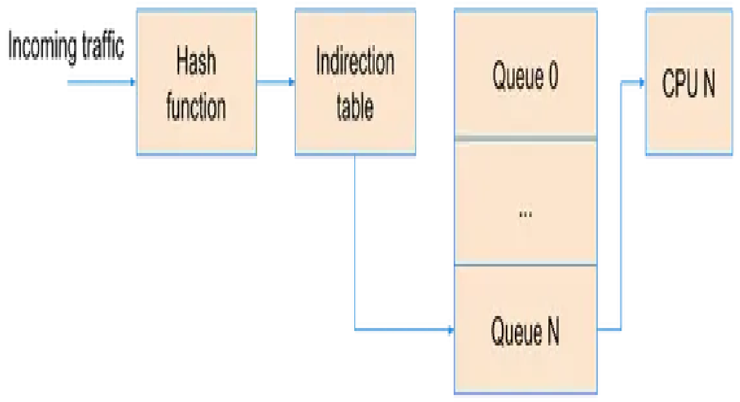


Figure 9.3: **RSS scaling**

Receive Side Scaling (RSS) is an important feature of the NIC that distributes different flows onto separate NIC receive queues, allowing for horizontal scaling of the system. Each queue is then processed by a different core, enabling improved performance. However, it's crucial to note that the system's capability may still be limited by the processing capacity of a single core, depending on the traffic patterns.

Now, let's move on to the results! In the context of Linux networking, the primary performance indicator we're focusing on is Packets Per Second (PPS), which measures the rate at which packets are processed. The amount of data carried in each packet is less relevant in this regard. Another important metric to consider is the number of interrupts per second that the system can handle.

Results:

packet size	Flow	pps	cpu usage
64	1	1.3M	1 core
64	4	4.4M	4.3 cores
64	8	7.5	8 cores

Table 9.2

Linux is able to route packets but at cost of many cores. Here Flows are streams of packets generated from traffic generator. Linux is able to redirect 1 million packets in 1 core, 4.4 M packets in 4.3 cores and 7.5 million packets required 8 cores. getting high performance from Linux require lot of resources that sometime are not available.

Test 2: Introducing a NAT rule

In this test, we're starting from scratch as we did in test 1 and we are adding a simple nat rule which causes all packets going through the DUT to be rewritten to a new source IP. These are the two rules:

IP commands:

```
iptables -I POSTROUTING -t nat -d 10.10.12.1 -s 10.10.11.1 -j SNAT — to 10.10.12.2
iptables -I POSTROUTING -t nat -d 10.10.11.1 -s 10.10.12.1 -j SNAT — to 10.10.11.2
```

Results:

packet size	Flow	pps	cpu usage
64	1	1.1M	1 core
64	4	4M	4 cores
64	8	5.5	8 cores

Table 9.3

The results show that rewriting the packets is quite a bit more expensive than just allowing or dropping a packet.

conclusion:

So yes, you can build a close to line rate router in Linux, as long as you have sufficient cores and don't do too much packet manipulations. All in all, an interesting test, and now we have a starting benchmark for future (kernel bypass / user land) networking experiments on Linux!

.

Kernel Bypass Networking with DPDK

User land networking is a concept where the task of handling network communication is moved from the traditional kernel in an operating system to a user-level program. In this approach, a technology called Data Plane Development Kit (DPDK) is often used to achieve exceptionally fast networking.

DPDK stands out because it doesn't rely on the usual interrupt-based approach. Instead, it continuously reads packets directly from the network hardware without waiting for interrupts. This constant polling enables it to process packets very efficiently and quickly.

To use DPDK effectively, we need a separate program that takes the packets received by DPDK and performs specific tasks with them, like acting as a virtual switch or router. By dedicating certain CPU cores to this process, servers can handle large amounts of network traffic more effectively.

9.2.1 Test: Setup

In our setup, we are utilizing a Layer 3 (L3) forwarding application. This application uses a hash table stored in memory, which is specifically allocated for the DPDK (Data Plane Development Kit) application. As incoming packets arrive from the Network Interface Controller (NIC), the DPDK app examines them. Based on the IP addresses present in the packets, the app looks up the appropriate rules in the hash table.

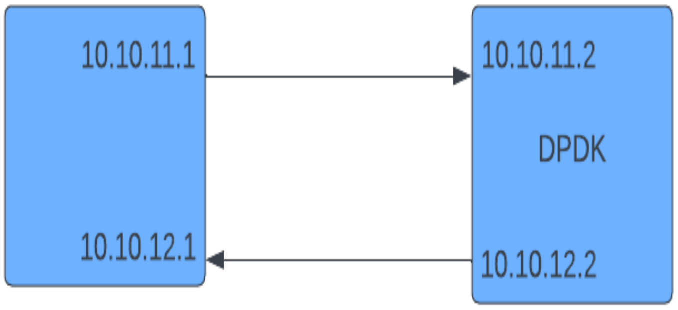


Figure:9.4

Using the information from the hash table, the DPDK app then redirects the IP packets to their corresponding output ports as specified by the rules. This way, the L3 forwarding application efficiently handles the routing of IP traffic based on predefined rules, optimizing network communication.

Results:

Here are the results of DPDK,

packet size	Flow	pps	cpu usage
64	1	1.1M	2 core
64	4	4M	2 cores
64	8	7.5	2 cores

Table 9.4

9.2.2 conclusion

There are some remarkable numbers showed by DPDK!, DPDK performance is very satisfying because it uses very low resources but it is not very popular yet. This is because there is little learning curve but it is very prominent in packet processing.

Router with XDP

Our goal is to create an XDP (eXpress Data Path) program that can forward packets at line-rate between two 10G Network Interface Cards (NICs) while utilizing the standard Linux routing table. This means we can add static routes using the 'ip route' command and even employ open-source BGP daemons like Bird or FRR.

We started with an XDP program, named 'xdp_router,' uses the bpf_fib_lookup() function to determine the egress interface for incoming packets based on the Linux routing table. Once the egress interface is identified, we utilize the bpf_redirect_map() action to direct the packet to the correct egress interface. The code is relatively concise, comprising about a hundred lines.

Test:Setup

After compiling the code (using 'make' in the parent directory), we load it using the './xdp_loader' program from the repository. We attach the 'xdp_router' code to both 'eno2' and 'eno4,' our two NICs, using the appropriate command. Then, we use './xdp_prog_user' to populate and query the redirect_params maps.

We should note that we pin the BPF (Berkeley Packet Filter) resources, specifically the redirect map, to a persistent filesystem by mounting it with the command: 'mount -t bpf bpf /sys/fs/bpf/'.

By following these steps, our XDP program will enable efficient packet forwarding between the two 10G NICs, while leveraging the flexibility of the Linux routing table for routing decisions.

```
#pin BPF resources (redirect map) to a persistent filesystem
mount -t bpf bpf /sys/fs/bpf/

# attach xdp_router code to eno2
./xdp_loader -d eno2 -F — progsec xdp_router
# attach xdp_router code to eno4
./xdp_loader -d eno4 -F — progsec xdp_router

# populate redirect_params maps
./xdp_prog_user -d eno2
./xdp_prog_user -d eno4
```

So for good we have built XDP router. This router is able to redirect packet from eno2 to eno4 and vice versa, pretty awesome!

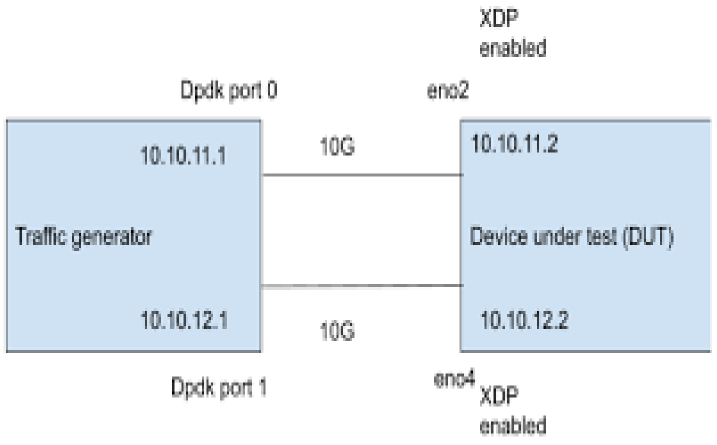


Figure:10.5

Results:

Here are the results of DPDK,

packet size	Flow	pps	cpu usage
64	1	1.1M	1core
64	4	4M	2cores
64	8	7.5	3 cores

Table 9.5

A word For XDP and Number of Queue

We incrementally increase the number of queues on the NIC and observe the impact on performance until achieving a loss-free transfer of packets between the two ports on the traffic generator. Specifically, for a 8Mpps transfer from port 0 to port 1 on the traffic generator through our XDP router (which forwards between 'eno2' and 'eno4'), we used the following configuration:

```
#ethtool commands
ethtool -L eno2 combined 4
ethtool -L eno4 combined 4
```

Subsequently, for the higher than 8 Mpps testing, we made the following adjustment:

```
#ethtool commands
ethtool -L eno2 combined 9
ethtool -L eno4 combined 9
```

By working collaboratively and incrementally tuning the number of queues while observing the packet transfer rate, we can effectively determine the optimal CPU core utilization for XDP processing. This approach ensures efficient packet handling while minimizing any potential packet loss.

conclusion:

We can build a high-performance peering router using just Linux and relying on XDP to accelerate the data plane, while leveraging the various open-source routing daemons to run our routing protocols. That's exciting!

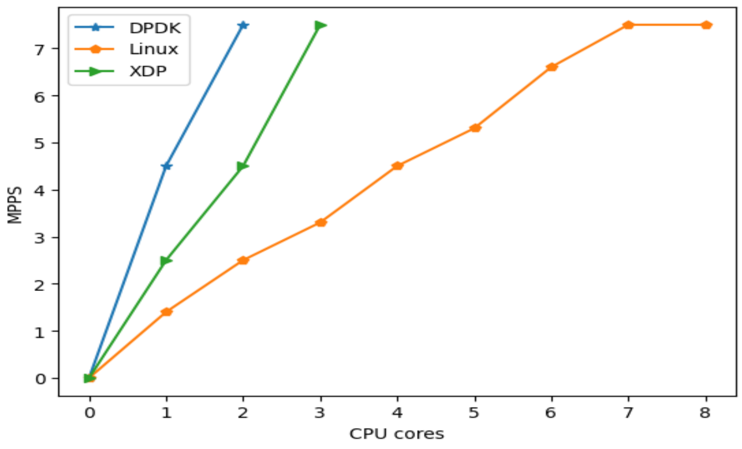


Figure:9.6 Number of packets redirect in XDP, DPDK and Linux.

9.3 Drop Packet performance with 10 Gb NIC:

In the Drop packet performance, we also observe the performance of the Linux networking stack in two configurations: one using the "raw" table of the iptables firewall module for packet dropping, which ensures the earliest possible drop in the network stack processing, and the other employing the connection tracking (conntrack) module, which carries higher overhead but is enabled by default on many Linux distributions. These two modes showcase the performance range of the Linux networking stack, from 1.8 Mpps with conntrack on a single core, up to 4.8 Mpps in raw mode. Additionally, the figure demonstrates that Linux performance scales linearly with the number of cores when there are no hardware bottlenecks.

Comparatively, XDP with its 14 Mpps on a single core offers a five-fold improvement over the fastest processing mode of the regular networking stack.

During our test in Linux raw mode, we measured the overhead of XDP by installing an XDP program that only updated packet counters and passed the packets on to the stack. We observed a drop in performance to 4.5 Mpps on a single core, corresponding to 13.3 ns of processing overhead. However, this difference is too small to be displayed legibly on the figure. DPDk performance is more better than other, As show in figure

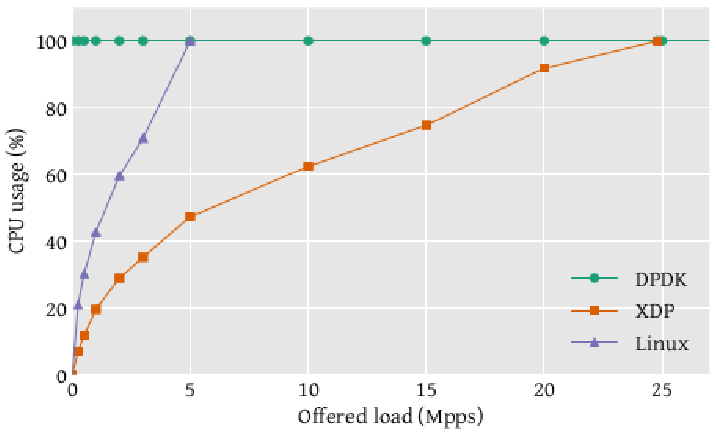


Figure:9.7 CPU Usage in drop Scenario.

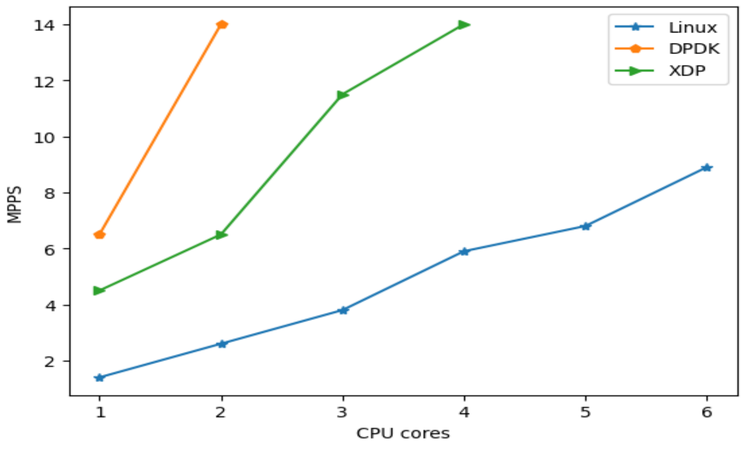


Figure:9.8 packet drop performance of XDP,DPDK and Linux.

10. Limitations and Challenges

Learning XDP (eXpress Data Path) and DPDK (Data Plane Development Kit) can present certain limitations and difficulties. Here are some potential challenges you may encounter:

10.1 Technical Complexity:

XDP and DPDK are advanced technologies that involve low-level network programming and optimization. They require a deep understanding of networking protocols, packet processing, and system internals. The technical complexity involved can be daunting, especially for those without a strong background in networking and systems programming.

10.2 Steep Learning Curve:

Learning XDP and DPDK typically requires a significant investment of time and effort. You may need to familiarize yourself with complex concepts such as packet filtering, eBPF (extended Berkeley Packet Filter), driver development, and performance tuning. The learning curve can be steep, especially for beginners.

10.3 Limited Documentation and Resources:

XDP and DPDK are relatively niche technologies, and finding comprehensive and up-to-date learning resources may be challenging. The official documentation and community support might not be as extensive as those available for more mainstream technologies, making it harder to find detailed examples, tutorials, and troubleshooting guidance.

10.4 Hardware and Platform Dependencies:

XDP and DPDK often require specific hardware capabilities and platform support. Some advanced features may only be available on certain network interface cards (NICs) or operating systems, limiting the scope of experimentation and hands-on learning.

10.5 Performance Optimization Challenges:

While XDP and DPDK offer high-performance data plane processing, achieving optimal performance can be complex. It involves considerations such as memory management, parallel processing, lock-free algorithms, and system tuning. Optimizing performance may require deep understanding and experimentation.

10.6 Debugging and Troubleshooting:

Debugging issues and troubleshooting problems in XDP and DPDK environments can be challenging due to their low-level nature. Identifying and resolving issues related to packet drops, memory corruption, or performance bottlenecks may require advanced debugging techniques and familiarity with the underlying system internals.

To overcome these challenges, it's helpful to leverage available documentation, online communities, forums, and experiment with practical examples to deepen your understanding of XDP and DPDK. Starting with foundational knowledge in networking and systems programming will also facilitate the learning process.

11. Conclusion and Future Work

11.1 Conclusion

We have introduced XDP, a system that facilitates safe and fast programmable packet processing within the operating system kernel. Through our evaluation, we have demonstrated that XDP achieves raw packet processing performance very higher than Linux.

While XDP's performance may not match that of the state-of-the-art kernel bypass-based solutions like DPDK, we believe that it offers compelling advantages that more than compensate for the performance difference. Some of these advantages include maintaining kernel security and management compatibility, selectively utilizing existing kernel stack features as needed, providing a stable programming interface, and offering complete transparency to applications. Additionally, XDP allows dynamic re-programming without service interruption and doesn't require specialized hardware or dedicated resources solely for packet processing.

On the other hand, DPDK excels at achieving even higher packet processing rates, as it can achieve up to 43.5 Mpps in our evaluation. It provides a highly efficient poll mode driver and direct access to network hardware, bypassing much of the kernel's networking stack.

We firmly believe that both XDP and DPDK have their unique strengths, and their suitability depends on specific use cases and requirements. While DPDK delivers exceptional performance and is commonly used in high-speed networking applications, XDP's advantages lie in its ability to coexist with the kernel's features and security while offering fast programmable. The real-world adoption of both XDP and DPDK in various applications further validates their significance, and their ongoing evolution promises further improvements in the future.

11.2 Future Work

Everything is not perfect and does not stop evolving so there is always some scope for improvement. That's why there are several potential areas of future work and advancements. Here are some notable directions for future research and development in XDP and DPDK:
Performance Optimization **:** Enhancing the performance of XDP and DPDK remains an active area of research. Future work may focus on optimizing packet processing algorithms, improving memory management techniques, exploring new parallel processing methods, and fine-tuning system configurations to achieve even higher data plane performance.
Security and Traffic Analysis **:** XDP and DPDK can be leveraged for security applications, such as DDoS (Distributed Denial of Service) mitigation, intrusion detection, and network monitoring. Future work may involve developing advanced techniques to identify and mitigate network threats, perform real-time traffic analysis, and enable more efficient security operations.
Programmability and Flexibility **:**Simplifying the programmability of XDP and DPDK is an ongoing effort. Future work may involve developing higher-level programming abstractions and frameworks that make it easier for developers to utilize and extend the capabilities of XDP and DPDK. This can include domain-specific libraries, language bindings, and tooling for rapid development and deployment
Hardware Acceleration **:** Taking advantage of hardware accelerators, such as SmartNICs (Smart Network Interface Cards) and FPGA (Field-Programmable Gate Array), can further boost the performance of XDP and DPDK. Future work may explore techniques to offload certain packet processing tasks to specialized hardware, leveraging their capabilities to improve throughput and reduce latency.

A. Appendices

A.1 References

Bibliography

[1] Clement Bertier. Linux kernel packet transmission performance in high- speed networks. *KTH ROYAL INSTITUTE OF TECHNOLOGY SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY.*

[2] IEEE Valentin Del Piccolo Ahmed Amamou Member IEEE Kamel Haddadou Member IEEE Danilo Cerovic, ´ Graduate Student Member and IEEE Guy Pujolle, Senior Member. Fast packet processing: A survey. *IEEE Communications Surveys Tutorials*, June 2018.

[3] Daniel Borkmann Cilium.io daniel@cilium.io John Fastabend Cilium.io john@cilium.io Tom Herbert Quantonium Inc. tom@herbertland.com David Ahern Cumulus Networks dsahern@gmail.com David Miller Red Hat davem@redhat.com oke Høiland-Jørgensen Karlstad University toke@toke.dk, Jesper Dangaard Brouer Red Hat brouer@redhat.com. The express data path: Fast programmable packet processing in the operating system kernel. *ACM Digital Library*, 2018.

[4] Pawol Parel. Why vector packet processing is worth your time. *Codilime.com.* URL https://codilime.com/blog/why-vector-packet-processing-is-worth-your-time/.

[5] Dariusz Sosnowski. How can dpdk access devices from user space? *Codilime.com.* URL https://codilime.com/blog/how-can-dpdk-access-devices-from-user-space/.