Author:
Yan, Yan

Title:
Programmable Smart NIC

Accelerating Intra-Server and Inter-Server Networking

Programmable Smart NIC: Accelerating Intra-Server and Inter-Server Networking

by

Yan Yan

A dissertation submitted to the University of Bristol accordance with the requirements for award of the degree of Doctor of Philosophy in the Faculty of Engineering

October 2023

Word count: 35016

Abstract

Over the past few decades, there has been a substantial increase in network bandwidth, accompanied by significant advancements in network technologies. As a result, there arose a demand for specialized hardware capable of relieving network-related workloads from the CPU. In response, the Smart NIC, a more intelligent network interface card, was introduced to alleviate the burden on server throughput.

The journey of my PhD work and thesis commenced with an exploration of both established and emerging Smart NIC technologies. The overarching objective was to harness FPGA technology to design and implement an innovative and universal Smart NIC architecture. This architecture was engineered to efficiently offload CPU workloads in common usage scenarios, particularly within Cloud and Telecom networks.

Given the adaptable nature of the NIC/Smart NIC, designed to seamlessly integrate with a range of hardware and operating systems, I leveraged various Open Source standards to develop a comprehensive FPGA-based Smart NIC solution. This solution encompassed both hardware and software components, contributing to the acceleration of intra- and inter-server networking.

The focus of my PhD thesis was directed towards the design and implementation of the FPGA-based components. I chose P4 as the foundational architecture, providing the necessary programmability and flexibility to meet our objectives. Multiple algorithms were devised to enhance throughput and reduce latency, with a particular emphasis on two scenarios: intra-server Open Virtual Switch offloading and inter-server Segment Routing.

To validate the effectiveness of our approach, a series of experiments were conducted, utilizing intra- and inter-server use cases. The results of these experiments served as compelling evidence, showcasing the substantial improvements achieved through the deployment of Smart NIC technology. This encompassed enhancements in both intra-server and inter-server throughput and reductions in latency, underscoring the significant impact of Smart NICs in optimizing network performance.

AUTHOR'S DECLARATION

This Page Intentionally Left Blank

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to my esteemed supervisors, Prof. Reza Nejabati and Prof. Dimitra Simeonidou, for their unwavering support in various aspects, both technically and personally. They have not only been mentors but also true friends throughout my research journey.

I also extend my sincere thanks to the remarkable individuals of the High Performance Networks Group, which I became a part of in 2011. The enthusiasm and extensive professional knowledge exhibited by my colleagues have been a continuous source of motivation throughout my research endeavours.

Last but certainly not least, I want to convey my appreciation to my family for their unwavering support. My parents have been a constant source of unconditional support, always standing by me and providing the confidence and security I needed to pursue my dreams. My husband Jianlin has always been there for me. His faith in my abilities has been a steady source of motivation. And my kids Amy and Zachary have been a continuous source of joy and inspiration. They grew so quickly while I pursued my doctoral studies. They also provided their own form of "supervision" by inquiring daily whether I had completed my PhD thesis this year.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

5G Fifth generation of mobile networks

AI Artificial Intelligent

AP Access Point

AR Augmented Reality

ASIC Application-Specific Integrated Circuit

BRAM Block Random Access Memory

C2H Card to Host

CPU Central Processing Unit

DCN Data Center Network

DMA Direct Memory Access

DPDK Data Plane Development Kit

DPU Data Processing Unit

DSP Digital Signal Processor

e2e End-to-End

eBPF Extended Berkeley Packet Filter

eMBB enhanced Mobile Broadband

FCS Frame Checksum Calculator

FF Flip Flops

FIFO First In First Out

GPU Graphics Processing Unit

H2C Host to Card

IaaS Infrastructure as a Service

ICT Information and Communications Technology

IDC International Data Corporation

IoT Internet of Things

ITU International Telecommunication Union

IP Internet Protocol

IT Information Technology

KPI Key Performance Indicator

LAN Local Area Network

LUT Look Up Table

mMTC Massive Machine-Type Communication

MPI Message Passing Interface

MPLS Multiprotocol Label Switching

NAT	Network Address Translation
NF	Network Functions
NFV	Network Function Virtualization
NFVI	NFV Infrastructure
NP	Network Processor
OPEX	OPeration Expenditure
OF	Open Flow
OS	Operating System
OSI	Open Systems Interconnection
PF	Physical Functions
RAM	Random Access Memory
RDMA	Remote Direct Memory Access
TCAM	Ternary Content Addressable Memory
TCP	Transmission Control Protocol
TOE	TCP/IP Offload Engine
TPU	Tensor Processing Units
TSO	TCP Segmentation Offload
P4	Programming Protocol-Independent Packet Processors
P4C	P4 Compiler
PCIe	Peripheral Component Interconnect Express
PTP	Precision Time Protocol
QoS	Quality of Service
Raspi	Raspberry PI
RSO	Receiving Side Offload
RSS	Receive Side Scaling
SDN	Software-Defined Networking
SerDes	Serializer/Deserializer
SR	Segment Routing
SR-IoV	Single Root I/O Virtualization
TDM	Time Division Multiplexing
ToR	Top of Rack
TSON	Time Shared Optical Network
UDP	User Datagram Protocol
UNIVBRIS	University of Bristol
UPF	User Plane Function
URLLC	Ultra-Reliable Low-Latency Communication
USO	UDP Segmentation Offload

VF	Virtual Functions
VM	Virtual Machine
VNF	Virtual Network Function
VLAN	Virtual Local Area Network
VPP	Vector Packet Processing
VPN	Virtual Private Network
VR	Virtual Reality
VXLAN	Virtual eXtensible Local Area Network
WAN	Wide Area Network
WDM	Wavelength Division Multiplexing

Chapter 1. INTRODUCTION

Cloud computing and cloud services [1] grow and expand continuously with new technologies such as serverless computing and containerization. Major cloud providers are also expanding their services to include AI (Artificial Intelligent) and machine learning. Benefiting from the natural remote service attribute, cloud computing and cloud services have gone through the outbreak of application scenarios due to the Coronavirus (COVID-19) pandemic: online education, remote work, online medical and other demand surged [2]. The pandemic has affected how companies and organizations work as well as cloud services. IDC forecasts [3] pointed out that, cloud computing and cloud services have moved from the Internet to the Intranet, which not only promotes the comprehensive digital transformation of enterprises, but also profoundly affects the production and lifestyle of the entire society.

Apart from Cloud, fifth generation of mobile networks network (5G) has grown from infant to mature since 2020. As of September 2021, over 170 operators in 70 countries have launched commercial 5G networks. The number of base stations has also increased rapidly. According to the latest projections from Gartner [4], the revenue generated from worldwide 5G network infrastructure is set to experience robust growth, with a forecasted increase of 39% to reach a total of $19.1 billion in 2024, up from $13.7 billion in 2023. Looking ahead to 2026, these projections anticipate a staggering 3.5 billion 5G connections globally, with 5G networks extending their coverage to over a third of the world's population.

The impetus behind this surge in demand can be largely attributed to the impact of the COVID-19 pandemic, which has led to an increased need for high-speed broadband connectivity [5]. This surge in demand is driven by bandwidth-intensive and latency-sensitive applications, including video calls, online gaming, and streaming video. These emerging technologies and evolving requirements are substantially elevating the need for end-to-end bandwidth and reducing latency demands, thereby reshaping the landscape of inter-server communication.

In the meantime, Moore’s law comes to an end, which leads to the restriction of the growth of the computation power per Central Processing Unit (CPU) core. To adapt to the increasing tasks and algorithms requirements for CPU powers, without double processing powers every 18 to 24 months as suggested by Moore’s law [6], more and more CPU cores are adopted by a single server, which promotes the adoption of virtual technologies (containers, virtual machines) [7] in the Data Centers to meet multi-user applications. The emergence of the virtual technologies enabled hosting diverse range of applications by using multi-core technologies efficiently, but it introduced diversity and complexity to the intra-server network. Servers must provide virtual and physical interconnects to enable web-scale products operational all year long, and the increased throughput to the servers gave more and more pressures on the servers to provide higher throughput and lower latency in the virtual and physical interconnects.

Down to the servers, the traditional way of processing the network packets in the server was using the Network Interface Card（NIC） and CPU cores. The NIC, equipped with Layer 1 Serdes and Layer 2 protocol extraction, focused on dealing with Ethernet Layer 2 to Layer 3 processing, and the CPU cores processed Layer 4 to Layer 7. However, because of the increased complexity of the network with emerging protocols and tunnels and the increased throughput. The traditional way of processing the network packets takes up CPU cores for network processing. A few technologies have been introduced for processing the network packets and offloading the CPUs, the Smart Network Interface Card was one of them. The introductory chapter outlined the innovated technologies towards dealing with emergent network complexity and helping to offload the CPUs. Because of the NIC's nature as part of IT devices in the servers, and controlled by the control plane software, the technologies innovation could be traced from open network and IT to the NIC itself.

1.1 Motivation

1.1.1 Open Network- enabled Hardware Innovation

Over the past 15 years, within the telecommunications industry, innovation often remained theoretical rather than being implemented due to factors such as monopolization, technological complexity, and a lack of competitive dynamics in the market. A significant shift occurred when web-scale companies, with substantial network demands, began to call for more affordable and adaptable technologies. This drove the necessity to democratize networking technologies, thereby accelerating innovation and fostering creativity. Consequently, the last decade has emerged as an era of flourishing network innovation, marked by the introduction of numerous open standards, protocols, and reference implementations.

Software Defined Networking (SDN) [8] represents a significant effort in the endeavor to standardize and generalize network programming. In parallel, Network Function Virtualization (NFV) [9] extends conventional Information Technology (IT) virtualization principles to network infrastructure. SDN and NFV technologies have achieved considerable momentum and widespread adoption as they transitioned from experimental phases to integral components of modern product ecosystems, albeit not always in their original configurations. Yet, SDN, along with open protocols that decouple the control plane from the data plane, faces an inherent challenge—or, from a more pessimistic perspective, hindrances—when it comes to communicating and managing network equipment responsible for handling network packets and serving as forwarding components. These network devices essentially impose limitations on how they can be defined through software.

Comparing to the revolution of the software side of the network, the rigid black boxes-kind of hardware has been the obstacles of supporting the agile network control for the industry for many years. With the software side of the network improves continuedly, such as SDN and NVF, there has also been a few attempts to improve the programmability and efficiency of the hardware-based network data plane platforms.

The forwarding metamorphosis [10] initiative aimed to introduce a match-action processing model as a response to the limitations of the existing OpenFlow-style approach to match and action processing. The prevailing method of match and action processing is restricted to a fixed set of fields and offers a limited array of packet processing actions. In the research paper, a comparative analysis was conducted, evaluating the merits of "Single Match Table," "Multiple Match Tables," and "Reconfigurable Match Tables (RMT)" as popular solutions. RMT, as one of the explored solutions, employs a series of pipeline stages, each equipped with a match table of arbitrary depth and width, enabling the matching of fields. It affords programmers the flexibility to comprehensively modify all header fields, surpassing the capabilities of Open Flow (OF). The paper delves into the design of a 64-port, 10 Gb/s switch chip implementing the RMT model. This implementation serves as a demonstration, dispelling concerns within the community and establishing that flexible OF hardware switch implementations are indeed feasible with negligible additional costs or power consumption.

The Open Compute Project (OCP) [11], originally launched by Facebook with the vision of creating the world's most energy-efficient data center capable of handling massive scale at minimal expense, has transformed into a collaborative community dedicated to revamping hardware technology to meet the increasing demands on computing infrastructure. OCP fosters a dynamic industry ecosystem for the global deployment of data centers in the cloud. Its objective is to gather contributions from a worldwide community of technology pioneers, working together to open proprietary IT infrastructure, initiating innovation from the foundation, and enhancing the efficiency, flexibility, and scalability of hardware.

Griffin [12] proposed programmable optical networking solution to enhance operations and resource efficiency for Data Center interconnects. The paper demonstrated a function-programmable and application-aware intra Data Center platform, which, on hardware side, the self-defined FPGA-based Top of Rach (ToR) switch is equipped with Ethernet over Time Division Multiplexing (TDM) and Wavelength Division Multiplexing (WDM) programmable capability; and on software side, a multi-stage software defined controller was built allowing dynamic allocation of resources and communicate to the hardware. The emphasis is on the optical programmability, and the adoption of more optical devices, not only fibers and transceivers in the cloud-based Data Centers.

P4 [13] serves as a programming language for network data plane customization. Given the ongoing proliferation of new header fields requiring integration into the OpenFlow interface, the Programming Protocol-independent Packet Processors (P4) language was conceived to instruct a target device on how to process packets. P4, a product of extensive research community endeavors, offers a dynamic way to define packet processing pipelines. The advent of P4 elevates the level of abstraction for network programming. It is crucial to note that this consortium, alongside the associated standards and protocols, is still in its nascent stages, making efforts to harness this technology even more significant in contributing to its evolution. Companies like Barefoot Networks [14] advocate for implementing open standards in the data plane. Moreover, major networking chip manufacturers have recently started incorporating P4 capabilities, given the growing traction of this approach [15] [16].

1.1.2 Intra-server Network Innovation

Furthermore, in the realm of software, several open design and standard forums have emerged with the aim of offering open design templates for different sectors within the IT and Networking industries. This initiative serves to encourage community involvement and streamline the fusion of various IT and networking technologies, ultimately enhancing efficiency and performance.

The Data Plane Development Kit (DPDK) [17] [18], originally conceived by Intel, has matured into an open-source initiative hosted under the aegis of the Linux Foundation. DPDK serves as a versatile software development kit meticulously crafted for the network data plane. Within its arsenal, DPDK houses an array of libraries designed to turbocharge packet processing workloads, ensuring optimal performance across a diverse spectrum of CPU architectures. DPDK distinguishes itself by achieving lightning-fast packet processing, accomplishing this feat by completely circumventing the kernel and conducting all operations within the user space. Moreover, DPDK embodies a low-overhead run-to-completion model that's finely tuned for swift data plane performance. It relies on direct polling to access devices, thereby eliminating the performance penalties often associated with interrupt processing. This streamlined and efficient approach facilitates the seamless acceleration of data plane operations, positioning DPDK as a dynamic and robust solution for demanding networking scenarios.

Open virtual Switch (OvS) [19] [20] [21], is a robust, multilayer virtual switch released under the open source Apache 2.0 license, purpose-built for virtualized environments. Its core design objective is to empower extensive network automation through programmatic extensions, all while steadfastly supporting standard management interfaces and protocols like NetFlow and sFlow. However, as the number of hosted virtual machines (VMs) within a single server increases, OvS's forwarding table expands significantly. This expansion demands more computational resources, which, in turn, leads to performance bottlenecks.

Enter Fast data – Input/Output (FD.io) [22], an open-source solution meticulously crafted to optimize the I/O path within virtualized systems. In contrast to Open vSwitch, which operates within the kernel, FD.io functions in the user space. In [23], it defines FD.io as a compendium of various projects and libraries engineered to enhance the transformative journey initiated by DPDK, thereby supporting adaptable, programmable, and composable services on a versatile hardware platform. The linchpin of FD.io is Vector Packet Processing (VPP), which processes packets in vectorized fashion within a processing graph, markedly amplifying packet processing speed and overall system performance.

The Open Platform for NFV (OPNFV), as outlined in [23], plays a pivotal role in advancing the development and evolution of NFV components across a diverse array of open-source ecosystems. Through comprehensive system-level integration, deployment, and rigorous testing, OPNFV constructs a foundational NFV platform serving as a reference point. This reference platform expedites the transformation of both enterprise and service provider networks, bringing about the realization of cutting-edge capabilities for the benefit of all stakeholders involved.[24] [25]

The IT and networking innovation and open-source collaboration listed above see the efforts for adapting the development of the network, decreasing the network complexity, increasing the network performance, and enabling the white box solution on the network devices to support current data center network changes. Summarizing [17]-[25], the virtualized and kernel bypass technologies grow fast. However, the virtualization and bandwidth in the data center network grow faster. As the CPU has been stretched to its limit, how to employ new or existing hardware to accelerate the network packet processing need innovational design and implementation.

1.1.3 NIC to Smart NIC innovation

When massive amount of data influx into the Data Centers (DC), to the server, it is the network interface card (NIC) which handles the network input/output packets. The NIC was traditionally designed and implemented dealing with the Physical and Data Link layers in Open Systems Interconnection (OSI) model. The higher layers are handled by software drivers of the NIC and by protocol stacks embedded in the host processor and kernel. It means the packets, come from the outside world, go through the physical interfaces and the MAC of the NIC, and then to the software side, processed by the driver and sent to the protocol stacks for the data extraction. All the server and computer-based network traffic processing require intense workload from the CPU, which is primarily designed for compute but not for network packet processing, especially with the extensive traffic bandwidth requirement.

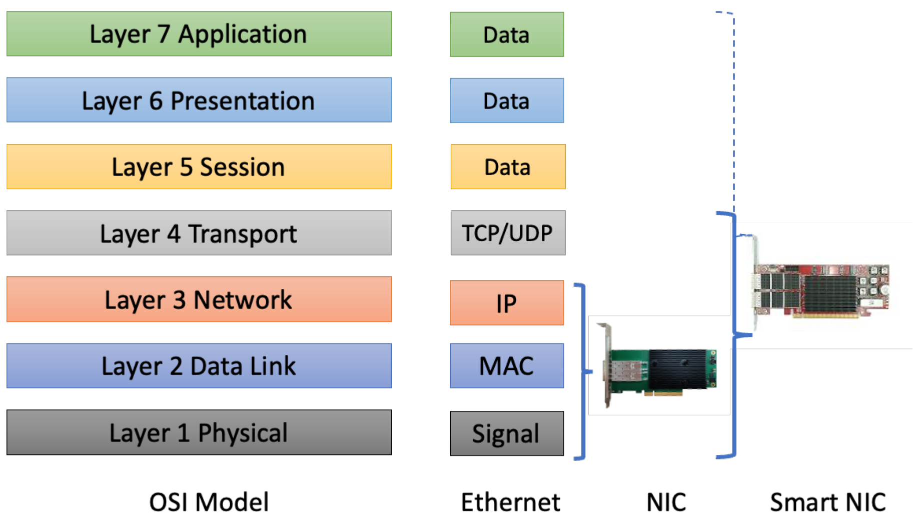


Figure 1:NIC and Smart NIC

Presently, the majority of public cloud-based Data Centers have undergone upgrades, transitioning from 10 Gbps to 25 Gbps, while in some privately managed compute-based Data Centers, individual server bandwidth has even surged to 100 Gbps or 200 Gbps. Operating and managing network processes efficiently is a critical aspect of modern computing, particularly in data-intensive environments. When network speeds are relatively moderate, staying below the 10 Gbps threshold, it is entirely feasible to rely on the inherent capabilities of the Operating System (OS) and the robust multicore processors available in the market. These components work in harmony to handle data transmission, ensuring that the network operates smoothly and efficiently.

However, as the need for higher data rates surpasses the 100 Gbps mark, traditional methods may no longer suffice. To achieve such exceedingly high throughputs without overloading the CPU, servers must embrace innovative technologies. Kernel bypass and offloading are at the forefront of these advancements. They enable servers to bypass the conventional network processing layers, allowing for a more direct and optimized data flow. By offloading some of the network-related tasks to specialized hardware or NICs, servers can attain exceptional data transfer rates while minimizing the consumption of precious CPU cycles.

One notable area where significant progress has already been made is in the realm of offloading various processing and network functions to NICs. Tasks like TCP/IP (Transmission Control Protocol/Internet Protocol) offloading have been developed to streamline the network stack's operations. These offloading technologies effectively free up the CPU from handling these specific tasks, which can be demanding and resource intensive. As a result, the overall performance of the server is significantly enhanced.

In [26], the NIC took over processing the entire TCP/IP stack on the hardware, which means the NIC handles all of the packet formation, checksum and other tasks, the CPU exchanges blocks of data with the NIC. However, since the TCP/IP stack is not a simple mechanism, the NIC itself can be easily overwhelmed with the TCP/IP offload, then causes data loss and worse performance. Since the TCP/IP stack can be very complexity based on the use case, to reduce the resource usage and balance the CPU and NIC workload, there is another TCP/IP offload method, called TCP/IP checksum offload.

TCP/IP checksum offload [27] is a method of offloading the TCP/IP/UDP checksums calculation and verification in the NIC, which allows the packets' TCP/IP checksum to be parsed and calculated by the NIC rather than by the host CPU. The kernel bypasses the calculation of the TCP/IP checksum and then throws the packet to the NIC, which processes the packets, calculates the checksum, and replace the original packet checksum fields with the one it calculated.

With the TCP/IP checksum offload in mind, there is another TCP/IP offload method, called TCP/IP segmentation offload [28], when a host or CPU needs to transmit a large bulk of data to the network, the data must be broken down to smaller packets that can follow the Ethernet or other protocols and pass through all the devices to the destination. Instead of dealing with the whole TCP/IP stack, the TCP/IP segmentation offload allows a TCP/IP stack in the host to emit larger frames and sends them to the NIC which divides the large frame into the frames that can be accepted by the devices. The initial TCP/IP headers needs to be copied and adjusted to the newly formed frames. Furthermore, the TCP/IP/UDP checksums need to be calculated and verified in the NIC as well.

TCP/IP offloading, regardless of the method employed, serves as an effective means of conserving CPU resources. At the system level, the introduction of Single Root I/O Virtualization (SR-IoV) [29] permits direct NIC access to virtual machines without CPU intervention, thereby reducing latency between virtual machines and boosting bandwidth. SR-IoV bypasses the hypervisor, enabling direct communication between the NIC and virtual machine. However, with the emergence and expansion of SDN-based solutions, SR-IoV, being hypervisor-independent, cannot be controlled by the SDN controller.

The integration of SDN applications and infrastructure, including virtual switches, within server environments places increased demands on the underlying hardware. These demands are necessitated by the objective of establishing a highly dynamic operational environment. In essence, the goal is to create a setup where these applications and infrastructure can operate seamlessly and efficiently. A key concept in this context is the programmable data plane, which can exist in both virtual and physical manifestations. This programmability opens exciting possibilities for enhancing the utilization of network and processing resources. The primary outcome is the creation of an environment that's conducive to offloading computing and communication tasks, leading to more efficient and responsive systems.

Hence, the conventional NIC, with its physical and MAC layer packet processing, has evolved into a programmable, accelerated, and offloading-capable counterpart known in the market as the "Smart NIC." In the current market landscape, Smart NICs predominantly fall into four categories: Application-Specific Integrated Circuit (ASIC), Multicore Processors, Network Processor (NP)-enabled, and FPGA-enabled.

When examining these four Smart NIC variants, ASICs stand out for their cost-efficiency but are constrained by fixed functionality, limiting their adaptability to evolving needs. Smart NICs powered by Multicore Processors or NP, while offering programming flexibility, grapple with performance limitations attributable to processing latency. In contrast, FPGAs are renowned for their initial expenses and the need for specialized hardware programming expertise; nevertheless, they excel in terms of rapid time-to-market, low latency, and high-performance capabilities.

In [30] and [31], Microsoft Azure showed their FPGA-enabled Smart NIC application for offloading the CPU and accelerating the cloud network in scale. Their successful use cases encouraged the followers on using FPGA-based solution for the acceleration in the cloud.

To adapt to the hardware programmability requirement in the network, P4 is widely adopted by the hardware devices, such as switches and NICs. To the day of writing this paper, P4-enabled data plane has been implemented on ASIC-based Barefoot Tofino [14] platform, multicore NP-enabled Netronome platform [32], FPGA-enabled NetFPGA [33] and Netcope [34] FPGA platform.

Barefoot is the pioneer in P4 and P4-based chips. The barefoot Tofino is an ASIC solution mainly used on the switches. Barefoot built the P4 data plane pipelines in the chip and enables the registers/Look up Table (LUT)-like modification to select the kind of look up method to implement, and the stages of pipelines to use. They have implemented complex location-based mapping algorithm to map the P4 languages to the chips to achieve programmability.

The Netronome's NP smart NIC is a comparable cheap, easy to do further development solution. Netronome's P4 is based on their well-developed eBPF [35]. But the processing latency is limited due to their NP-based solution. With the bandwidth increasing to 100G and beyond, the performance is reduced because of the nature of NP.

The NetFPGA solution [34], which relies on the Xilinx SDNet tool for translating the P4 file to a new netlist, and then a bit file, is vendor dependant and lacks the flexibility and scalability; for the P4FPGA [36] and Netcope [37] solution, both built their own p4 compiler to compile the P4 file to the FPGA netlist. The critical challenging for P4 and other high level language compiled directly to FPGA netlist is timing closure in the FPGA. The timing closure approaches they supplied, such as insert registers or using pipelined FIFOs, could be easily restricted by the resources of FPGA, had no convincing evidence that the approaches could close the timing in the FPGA in the most applications.

1.2 Research Objectives

The motivation behind embarking on my doctoral research stemmed from the pressing need to relieve the CPU of the arduous task of processing network packets, with the ultimate goal of accelerating network throughput at the server edge. This critical challenge served as the driving force behind my PHD work.

The research endeavor commenced with a deep dive into the world of Smart NIC solutions within the open-source domain, closely aligning with software-defined standards such as Software Defined Networking, P4 programming language, OvS, and other related technologies. The overarching aim of this exploration was to gain an in-depth understanding of the cutting-edge features and applications of programmable Smart NICs, which are pivotal in the modern landscape of network acceleration.

The primary objective that loomed large throughout this research journey was the creation of an innovative white-box Smart NIC solution designed to catalyze and enhance both intra-server and inter-server communication. In essence, this Smart NIC solution would serve as a powerful catalyst for optimizing data transfer and communication within and between servers.

Looking back to the genesis of this research in 2018, it was clear that 100 Gigabit Ethernet (100GbE) line cards were rapidly emerging as a new standard and trend in server networking. This shift in network requirements formed the bedrock upon which my research was built, with a particular focus on 100GbE FPGA-based platforms. This strategic alignment was driven by the realization that the research needed to address the immediate and evolving needs of the industry:

a) Smart NIC architecture design and implementation: To build an open white box Smart NIC solution by leveraging the popular software-defined open-source platforms, such as SDN, P4, OvS and etc.

b) FPGA-based Smart NIC: To focus on the FPGA-based hardware design and implementation.

c) Offloading CPU: Involves delegating the processing of network services to the Smart NIC itself, thereby relieving the CPU of the server from the burden of these network-related tasks. By offloading these operations onto the Smart NIC, the CPU is liberated, allowing it to dedicate its full computing power to application-specific tasks.

d) Intent driven networking: Facilitating the Effortless Deployment and Acceleration of New Applications Based on Their Specific Networking and Computational Needs Through the Utilization of the Proposed Smart NIC Solution.

b) Maximum the throughput and minimum the latency: To develop algorithms to dynamically fit to the network bandwidth and latency requirement.

In summary, my PhD research journey was fueled by the need for efficient and accelerated network communication, marked by a dedicated exploration of Smart NICs, FPGA-based platforms, and the overarching aim of creating a pioneering Smart NIC solution. This research was timely and responsive to the evolving landscape of server networking requirements, paving the way for enhanced network performance in the modern era.

1.3 Contributions

1.3.1 Research works before starting PhD

Before registering as a PhD candidate, my research focused on the high speed, low latency FPGA-based optical/electronic interfaces. The research work and papers submitted and accepted before are listed below.

In [38][39], the research was centered on the metro network and sub-lambda technology. The paper introduced a Layer 2 Time-Shared Optical Network (TSON) metro node, renowned for its ability to provide distinct Quality of Service (QoS) latency levels, all while ensuring uninterrupted, contention-free operation through a variety of time-slice allocation strategies. The TSON node itself was founded on Field-Programmable Gate Array (FPGA) technology, housing both the protocol and algorithms within its FPGA framework.

This TSON node, devoid of any dependence on topology, presented a network capable of delivering divergent QoS latency levels, always upheld as contention-free, thanks to the strategic deployment of diverse time-slice allocation schemes. The experimental findings highlighted exceptional performance metrics, including a remarkable throughput of up to 8.68Gbps per 10Gbps port, achieving a staggering 95.38% of the theoretical maximum throughput. Moreover, it recorded impressively low latency of under 160μsec and jitter of less than 25μsec, underscoring the outstanding capabilities of the TSON system.

In [40], the study delved into the realm of network functions virtualization and the hardware requirements of networking systems. It emphasized the necessity for a certain level of flexibility and programmability in the hardware to enable enhanced control over network-wide operations. Building upon the foundations laid in [30] and [31], this paper took a progressive step forward in the arena of network and node programmability. It reported on the design and realization of the pioneering FPGA-based optical network function programmable node. This innovative node possesses the capability to employ both electronic systems (such as FPGA/SoC) and optical devices (AOD) to deliver a diverse range of network functions on demand.

In [41][42], the research was oriented towards enhancing the capabilities of disaggregated data center networking. The study introduced an FPGA-based optical programmable switch and interface card (SIC), strategically positioned to potentially supplant the conventional network interface card (NIC). When directly integrated into servers, this SIC unlocks the potential for robust intra-rack communication between server blades, presenting a groundbreaking solution for data center networking. The incorporation of the SIC paves the way for the realization of a flat and scalable all-optical data center architecture, applicable to both intra-rack and inter-cluster communication.

The distinctive attributes and functionalities of the SIC facilitate direct blade-to-blade interconnectivity within the same rack, effectively eliminating the need for electronic devices within the Top of Rack (ToR) switch. This innovative approach substantially reduces intra-rack latency. Simultaneously, the SIC serves as an optical NIC, streamlining data transfer between server blades and the ToR. In essence, this paradigm shift revolutionizes data center networking by optimizing data transmission within and between server racks.

Furthermore, by collaborating with other researcher, summarizing my other research outcome [43]-[60], although the use cases or scenarios were various, but my focus and contribution were mainly on how to enable the network interface with more capabilities of high speed, low latency and programmability. Due to the advantage of the researches with no commodity devices available in the market, most of the research work are FPGA-based. The FPGA allows the development of user-defined logics, functions and features; and because of its parallel processing nature, with proper design, the FPGA-based interface can achieve ultra-low latency.

1.3.2 Contributions of PhD Research

The PhD started in January 2018 with the aim of exploring and implementing the FPGA-based Smart NIC solution for offloading CPU and accelerating intra-server and inter-server network. The research work focused on the innovative research and development on the next generation open platform Smart NIC to contribute to network and IT innovations. The thesis provided detailed background on how the existing network technologies, such as virtualization, SDN, P4 and so on affecting the NIC to Smart NIC development. Furthermore, it detailed the factors and functions the Smart NIC should be equipped. With all these in minds, it demonstrated the architecture design of the Smart NIC, the use cases for inter-server and intra-server acceleration, the testbeds setups, and the results. We also contributed a few algorithms for improving the Smart NIC's bandwidth and latency.

The PhD research created an innovative compact programmable Smart NIC solution that enhanced the server-end network in cloud and 5G network applications. The contributions are listed below:

- FPGA-based P4-enabled Smart NIC architecture design and implementation.

FPGA supplied electronic-level programmability and P4 supplied the network-dataplane-level programmability, the PhD research designed and implemented a FPGA-based P4-enabled Smart NIC platform for accelerating the server edge network. The design includes both the application-based architecture design and electronic-based high performance design.

- P4-enabled dataplane optimization

The P4-enabled Smart NIC was planned to be used as a generic network dataplane for multiple use cases acroass cloud networks and 5G networks, such as segment routing (SR) [61], Inband Network Telemetry (INT) [62], and Open Virtual Switch(OvS) offloading.

- Testbed setup and results collections

The FPGA-based Smart NIC was tested in the lab and on the field in the Smart City environment. The results showed the Smart NIC with significant enhance in offloading CPU burden and enhance the throughput and latency performance.

1.4 Publications

The following papers have been published as an outcome of this PhD:

[C1]: Y. Yan, S. Tan, A. F. Beldachi, K. Rajkumar, R. Wang, R. Nejabati and D. Simeonidou, "P4-enabled smart NIC: Architecture and technology enabling sliceable optical DCs", Proc. ECOC, 2019.

For the paper [C1], I initiated the idea of P4-enabled smart NIC for enabling sliceable optical DCs, made the most significant intellectual contribution to the work, including designing the study, acquiring and analysing data from experiments, and writing the manuscript.

[C2]: Y. Yan, J. Zhuang, R. Nejabati and D. Simeonidou, "P4-Enabled Smart NIC for Intra-Server Network Virtualization Acceleration," 2020 Asia Communications and Photonics Conference (ACP) and International Conference on Information Photonics and Optical Communications (IPOC), Beijing, China, 2020.

For the paper [C2], I initiated the idea of P4-enabled smart NIC for intra-server network acceleration, made the most significant intellectual contribution to the work, including designing the study, acquiring and analysing data from experiments, and writing the manuscript.

[J1]: Y. Yan, A. F. Beldachi, R. Nejabati and D. Simeonidou, "P4-enabled Smart NIC: Enabling Sliceable and Service-Driven Optical Data Centres," in Journal of Lightwave Technology, vol. 38, no. 9, pp. 2688-2694, 1 May1, 2020, doi: 10.1109/JLT.2020.2966517.

For the paper [J1], I made the most significant intellectual contribution to the work, including designing the study, acquiring and analysing data from experiments, and writing the manuscript

The following patents have been filed and approved in China:

[P1]: A logic implementation method that can realize both P4 and OvS in Smart NIC/DPU.

Patent No. : ZL201111606889.4

[P2]: A method that can enhance the INT performance in SmartNIC/DPU.

Patent No. : ZL2021111074386.

[P3]: A method that can decrease the matching logic resources in the Smart NIC/DPU.

Patent No. :ZL202111107845.7

1.5 Thesis outline

The subsequent sections of this thesis follow this organization:

Chapter Two: In this chapter, the groundwork is laid with an exploration of essential background information encompassing NIC, Smart NIC, SDN, P4, OvS, and virtualization technologies. This chapter is divided into three sub-sections, each delving into distinct areas: software-defined programmable technologies, virtualization technologies, and Smart NIC technologies.

Chapter Three: The focus of this chapter is on the comprehensive software and hardware solution, which revolves around the design and development of a P4-enabled Smart NIC architecture. This advancement enhances data plane programmability and facilitates high-bandwidth, low-latency server-centric communication. Additionally, the chapter delves into the algorithms that have the potential to boost throughput while minimizing latency.

Chapter Four: Chapter four presents a demonstration of the application of P4-enabled Smart NICs in accelerating inter-server networks in the context of 5G use cases. These use cases encompass 5G UPF, INT, and Segment Routing, with P4 empowering the hardware capabilities necessary for these scenarios. The chapter also includes details about the setup of a testbed at a city-wide scale for Segment Routing, and it provides measurement results.

Chapter Five: This chapter shifts its focus to another key application within cloud-based Data Centers. It primarily explores the use of P4-enabled Smart NICs for offloading the server OvS dataplane. The chapter provides benchmark comparisons between the offload solution and scenarios without offloading.

Chapter Six: The final chapter serves as the conclusion and outlines potential avenues for future work and research.

Chapter 2. BACKGROUND TECHNOLOGIES

The emergence and development of the Smart NIC were stimulated by the emergence of the new technologies and requirements generated by the DCs. The history of data center networking can be traced back to the early days of computer networking when mainframe computers were connected via proprietary networks. These early networks were designed for low-bandwidth communication and were typically limited to a single building or campus.

In the 1980s and 1990s, significant developments in computing and networking technologies led to the evolution of data center networking. During this period, client-server computing gained popularity, where powerful servers served data and applications to multiple client devices. This shift in computing architecture necessitated advancements in networking to facilitate efficient communication between clients and servers. The advancements in client-server computing, the rise of Ethernet for Local Area Network (LAN), and the utilization of TCP/IP for Wide Area Network (WAN) were key drivers in the evolution of data center networking. And these technologies continue to be fundamental building blocks for modern data center infrastructure.

The late 1990s and early 2000s witnessed the dot-com boom, a period of extraordinary growth and speculation in Internet-based companies. This era was marked by a surge in investment and innovation as businesses aimed to leverage the increasing popularity of the Internet and capitalize on its potential. Companies such as Cisco, Juniper, and Brocade played a crucial role in developing high-speed switching and routing technologies to meet the requirements of these rapidly expanding web-based applications. They focused on creating networking equipment capable of handling the massive amounts of data generated by these applications and ensuring seamless connectivity. It's important to note that while the dot-com boom eventually experienced a significant downturn in the early 2000s (known as the dot-com bust), the advancements made during this period in data center networking technology continued to have a lasting impact. The innovations and lessons learned during the boom helped shape the foundation of modern data center networking, which has continued to evolve and adapt to the ever-increasing demands of digital services and applications.

In the mid-2000s, the emergence of cloud computing and virtualization technology brought about significant advancements in data center networking. These developments further accelerated innovation and reshaped the way networks were designed and managed. The mid-2000s marked a significant turning point in data center networking, with cloud computing, virtualization, Virtual Network Function (VNF), and SDN driving innovation and transforming the way networks were designed, deployed, and managed. These advancements laid the groundwork for the modern data center networking architectures and technologies that continue to evolve today.

More recently, the rise of big data, artificial intelligence (AI), and the Internet of Things (IoT) has brought forth new challenges and opportunities in data center networking. High-speed networking technologies, such as 25G and 100G Ethernet, have become common in data center environments. These technologies provide significantly higher bandwidth compared to previous Ethernet standards, allowing for faster data transfer and improved network performance. The adoption of higher speeds helps meet the demands of data-intensive applications, such as big data analytics and AI, which require rapid data processing and analysis. These advancements have prompted the need for more powerful and efficient networking technologies to handle the increasing volume, velocity, and variety of data.

Overall, the emerge of new technologies and application in the network introduced new challenges and opportunities in data center networking. The network landscape has undergone a significant revolution with the transition from rigid to programmable data planes, the shift from separated-controlled to centrally controlled networks, and the transformation of network devices from dedicated usage to general-server based systems. These innovations have revolutionized the network device landscape, allowing for more agile, programmable, and efficient network architectures. Network devices can now be more easily managed and reconfigured, providing greater flexibility and adaptability to changing network requirements. The convergence of networking with general-server based systems has also opened up new possibilities for the integration of networking functions with other IT services and applications, further enhancing the capabilities of network devices.

The programmable Smart NIC, which was researched, designed, and implemented in the PhD, was based on the data center networking innovations mentioned above. This chapter covers the background technology of these innovations, especially the development trends.

2.1 The road to programmability: from SDN to P4 and beyond

In traditional networking, network devices like switches and routers have embedded control and data planes. These devices handle both the management of network traffic (control plane) and the forwarding of data packets (data plane). This setup often results in complex and distributed configurations that can be challenging to manage and scale.

In the late 2000s, several research projects and publications contributed to the development of SDN concepts. These included projects such as Ethane, NOX, and Beacon, which explored the benefits of decoupling network control from forwarding functions and promoting programmability.

In 2008, researchers at Stanford University introduced the concept of OpenFlow, now seen as the flag protocol of SDN. OpenFlow is a protocol that enables the separation of the control plane and data plane in network switches and routers. It allows for centralized control and programmability of network devices.

SDN severs the connection between the control plane and the data plane. The control plane assumes the role of overseeing network traffic management, formulating decisions regarding packet forwarding, and configuring network devices in alignment with these determinations. The control plane operates on a central controller, which can be a dedicated server or software running on a server, separate from the network devices themselves. On the other hand, the data plane remains on the network devices and is responsible for the actual forwarding of data packets based on the instructions received from the central controller. Network devices in an SDN architecture are often referred to as forwarding devices or switches.

The division of the control plane and data plane in SDN offers numerous advantages. It centralizes the management and control of the network, allowing for more efficient configuration, monitoring, and optimization. Network administrators can programmatically control and configure the behaviour of the network through the central controller, enabling easier management and rapid adaptation to change network conditions.

The researchers at Stanford University developed a prototype SDN architecture called OpenFlow (shown in Figure 2), which became one of the initial and widely adopted standards for implementing SDN. OpenFlow provided a standardized protocol for communication between the central controller and the network devices. It allowed researchers to experiment with new network protocols and architectures without requiring modifications to the network devices themselves.

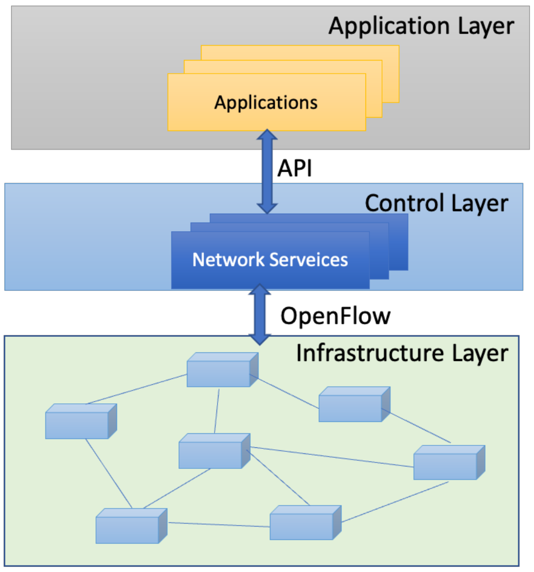


Figure 2 :A sample of SDN Architecture

Since its introduction, SDN has gained significant traction and has been embraced by both academia and industry. It has enabled the development of more agile and programmable networks, facilitating innovations such as network virtualization, network slicing, and dynamic service provisioning. SDN has also contributed to the evolution of network management and automation, allowing for more efficient and scalable network operations.

In 2011, the Open Networking Foundation (ONF) was founded to promote and advance the adoption of SDN. The ONF played a significant role in developing the OpenFlow protocol, which became the de facto standard for SDN implementations. OpenFlow provided a standardized approach for communication between the central controller and network devices, enabling interoperability and facilitating the deployment of SDN solutions. Since its inception, SDN has continued to evolve, and numerous vendors have developed their own SDN solutions and frameworks. This has led to a vibrant and diverse ecosystem of SDN technologies, allowing organizations to choose from a range of options that best suit their specific requirements.

Today, SDN finds applications in various types of networks, including data centers and telecommunications networks. In data centers, SDN enables efficient management of network resources, dynamic provisioning of services, and improved traffic engineering. It simplifies network operations, enhances scalability, and enables rapid adaptation to changing demands. In telecommunications networks, SDN facilitates the creation of NFV and allows for the centralized management of network services, leading to increased agility and cost savings. SDN also enables service providers to offer more flexible and customizable network services to their customers.

By decoupling network control from network devices and enabling centralized control through software, SDN empowers network administrators to manage and control the flow of network traffic more dynamically and responsively. They can define and enforce policies, allocate resources, and respond to changing network conditions in real-time, without relying on manual configuration of individual network devices. The software-based control and programmability of SDN enable more efficient utilization of network resources, leading to potential cost savings. By dynamically adjusting the network configuration based on traffic patterns and demands, organizations can optimize resource allocation and achieve better performance and scalability.

However, Traditional network devices, such as switches and routers, are often designed with a bottom-up approach, relying on external chips from third-party silicon vendors. These chips serve as the heart of the system and determine the capabilities and functionality of the device's operating system. OpenFlow provides a standardized way to communicate between the SDN controller and the network devices. However, OpenFlow has limitations, such as being protocol-dependent and not being able to handle all types of network traffic.

The limitations of traditional networking devices become apparent when adding new features or functionality, as the fixed-function nature of the chips makes it challenging to reconfigure the internal packet processing pipeline at runtime. Consequently, introducing new feature sets often involves complex processes and lengthy chip redesigns, which can take months to complete.

To address these limitations, the P4 language [62] was developed as an alternative approach. P4 embodies a "top-down" paradigm, akin to the operational principles of CPUs, GPUs, or digital signal processors (DSPs). In these processing units, specialized code written in distinct programming languages, such as C programming for CPUs, CUDA programming for GPUs, or Matlab programming for DSPs, is compiled and integrated into the processor. This integration empowers these units with a degree of flexibility and programmability, enabling them to execute tailored functions. P4 applies the same principle to network devices. Instead of being limited by fixed-function chips, P4 allows programmers to define the network's feature set through a P4 program. Figure 3 demonstrated the P4 flow. The program specifies how packets should be processed, forwarded, and modified by the network device. The P4 code is then compiled and injected into the network device, enabling the desired configuration.

P4 represents a new approach to network programmability that builds on the principles of SDN but goes beyond the limitations of traditional SDN protocols like OF. P4 provides greater flexibility and control over how packets are processed in the network, allowing for more customized and efficient network architectures. It allows network administrators and operators to define and customize the behavior of their network devices according to their specific requirements. With P4, adding new features or modifying existing ones becomes a more streamlined process, as it no longer requires lengthy chip redesigns.

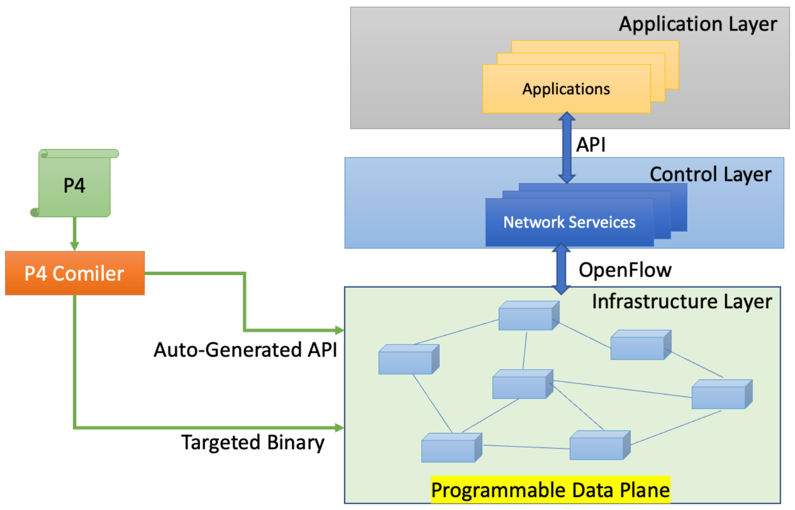


Figure 3:P4 approach for programmable data plane

The P4 language has evolved over time to address the needs of network programmability and packet processing. The first P4 language released was P4_14, it was the initial standardized version of the P4 language and was released in 2015. It laid the foundation for programmable data planes in networking devices. P4_16 [63] was released in 2016 and introduced significant improvements and enhancements compared to P4_14. Some notable features and differences include:

a. More expressive type system: P4_16 introduced a more refined type system, allowing for better control over packet formats and sizes. This enables precise manipulation of packet headers and fields.

b. Control flow constructs: P4_16 introduced control flow constructs like loops and conditional statements, enabling the creation of more complex and dynamic packet processing logic. This allows for more advanced and flexible forwarding behavior.

c. Support for extern objects: P4_16 introduced the concept of extern objects, which are user-defined functions that can be invoked within a P4 program. This enables modular code organization and the use of external functionality for packet processing.

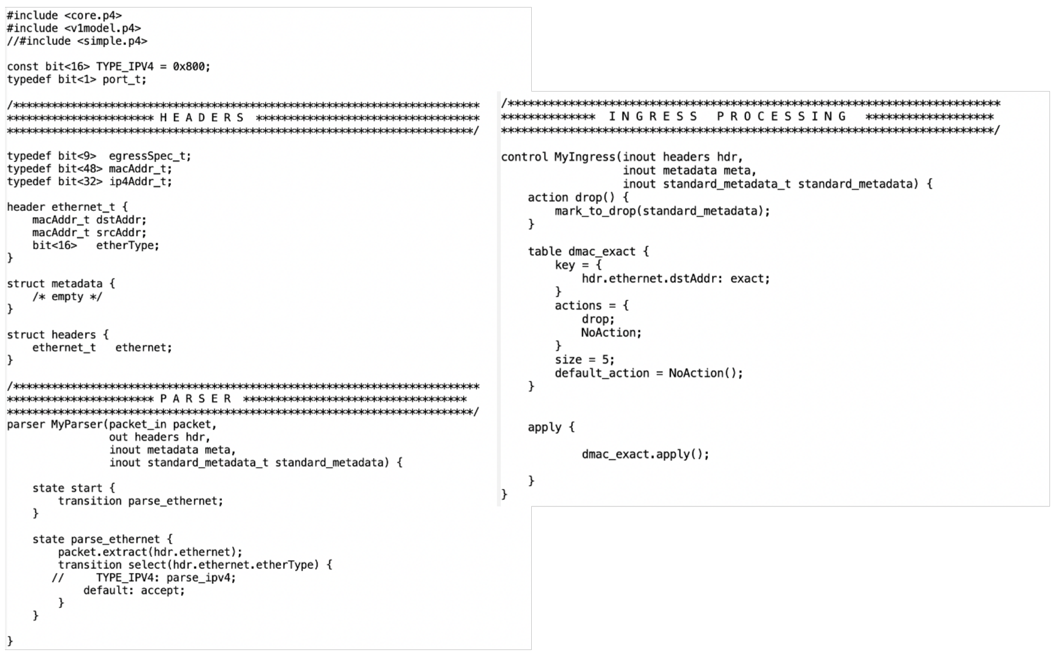
The P4 architecture model, introduced in P4_16, provides a framework for organizing the components and layers involved in the P4 programming language. It helps define the behavior of network switches and routers. By adhering to the P4 architecture model, network device manufacturers can provide programmable interfaces while maintaining the necessary fixed functionality for efficient packet processing. P4 programs can then be developed and deployed on devices that support the specific architecture model, allowing for greater flexibility and innovation in network designs.

P4 has steadily risen in prominence within the network industry, positioning itself as the logical progression in the evolution of SDN. SDN and P4 are synergistic technologies that, when combined, deliver a powerful toolset for achieving programmability and adaptability in network devices.

SDN lays the foundation with its overarching architectural framework, notably featuring the separation of the control plane from the data plane. P4, on the other hand, steps in to offer a highly nuanced and customizable approach to governing the packet processing behavior within the data plane. Together, they create a robust partnership, with SDN providing the overarching structure and P4 delivering the granular control and tailor-made customization essential for optimal network performance and flexibility.

In P4, a programmer has the liberty to define with complete flexibility how packets moving through programmable data plane components will undergo processing. Within the realm of P4, a frequently employed term is "target," which encompasses a diverse range of devices that can be programmed using P4.

These devices can include switches, routers, NICs inserted into servers, and software switches. P4 provides a programming model that can be applied to a range of network devices, allowing for consistent and customizable packet processing across different hardware and software platforms. The great advantage of P4 is its protocol independence. While traditional network devices are typically designed to process specific protocol headers, P4 offers the flexibility to define the processing behavior for any protocol, including proprietary or custom protocols. This flexibility is particularly useful in scenarios where non-standard or application-specific protocols need to be processed within the network devices.



```
#include <core.p4>
#include <v1model.p4>
//#include <simple.p4>

const bit<16> TYPE_IPV4 = 0x800;
typedef bit<1> port_t;

/*************************************************************************
*********************** H E A D E R S  ***********************************
*************************************************************************/

typedef bit<9>  egressSpec_t;
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;

header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16>   etherType;
}

struct metadata {
    /* empty */
}

struct headers {
    ethernet_t   ethernet;
}

/*************************************************************************
*********************** P A R S E R  ***********************************
*************************************************************************/
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {
        transition parse_ethernet;
    }

    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
       //     TYPE_IPV4: parse_ipv4;
            default: accept;
        }
    }

}
```

```
/*************************************************************************
**************  I N G R E S S   P R O C E S S I N G   *******************
*************************************************************************/

control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
    action drop() {
        mark_to_drop(standard_metadata);
    }

    table dmac_exact {
        key = {
            hdr.ethernet.dstAddr: exact;
        }
        actions = {
            drop;
            NoAction;
        }
        size = 5;
        default_action = NoAction();
    }


    apply {

            dmac_exact.apply();

    }
}
```

Figure 4:An example of P4 program

Figure 4 showcases a sample P4 program, which exemplifies the meticulous nature of P4 programming. In P4, the definition of all header types, encompassing both well-established and customized ones, as well as the parsing process for these headers, necessitates explicit detailing within the program. It is the responsibility of the programmer to craft what is referred to as a "match-action pipeline" within the designated dataplane block. This pipeline can be composed of one or multiple tables where the matching of parsed header fields takes place.

For instance, let's delve into the "table dmac_exact" within the example. The key section of this table comprises a match field, specifically "hdr.ethernet.dstAddr," which corresponds to the destination MAC address. Each table can be assigned one or more actions, to be executed at runtime. Actions, in turn, determine how packets are to be handled, whether it involves modifying selected header fields, dropping the packet, forwarding it to a specified physical port, and more. It's worth noting that while tables can process packet header fields, they can also operate on standard or user-defined metadata linked to packets.

Crucially, all aspects pertaining to tables and their internal structure, including the number of tables, match fields, and actions for each table, and the behavior of these actions, are open to customization by the P4 programmer. This adaptability is what makes P4 a potent and versatile solution in the realm of network programming.

Within the framework of a P4 architecture, the intricate specification of the target device's pipeline is furnished by the manufacturer through a P4 library file. Typically denoted as "<some_architecture_model>.p4," this file comprises all the requisite declarations for functional blocks existing in the target pipeline. These declarations encompass information on block types, data types, constants, externs, and more. It is essential to recognize that the architecture definition model includes solely the declarations of programmable blocks, leaving fixed-function blocks, if present within the target processing pipeline, beyond the scope of manipulation by any P4 program. As a result, the primary focus centers on defining and configuring the programmable blocks nested within the pipeline.

The architecture file serves as a blueprint for the P4 program and is provided by the manufacturer. It outlines the functional blocks and their relationships within the pipeline. The manufacturer specifies the capabilities and features of the programmable blocks in this file.

2.2 The road to virtualization: server virtualization and network virtualization

The requirements of the programmable Smart NIC, researched in the PhD, was also stimulated by virtualization technologies. In modern DCN and cloud computing environment, server virtualization and network virtualization are complementary technologies that work together to create a more flexible, scalable, and efficient network infrastructure.

2.2.1 Server virtualization

Server virtualization, the practice of creating multiple virtual instances of a single physical server, has its roots in the mainframe era of computing. The concept of server virtualization can be traced back to the 1960s when IBM introduced "time sharing." This allowed multiple users to access a single mainframe computer simultaneously by dividing its resources into virtual partitions. In the 1970s, IBM developed the VM/370 operating system, which introduced full virtualization on mainframe systems. VM/370 allowed multiple instances of operating systems, known as virtual machines, to run concurrently on a single mainframe.

Modern server virtualization, as we know it today, took shape in the early 2000s with the release of VMware's virtualization software for x86-based servers. In 2001, VMware introduced its product called VMware ESX, which allowed IT administrators to partition physical servers into multiple virtual machines (VMs) running different operating systems and applications. VMware's virtualization software gained rapid adoption in enterprise IT departments. By leveraging virtualization, organizations could maximize the utilization of their server hardware, reducing the need to purchase and maintain a large number of physical servers. This resulted in cost savings, improved server efficiency, and more streamlined IT infrastructure management.

Following VMware's success, other companies entered the virtualization market. Microsoft introduced Hyper-V as its virtualization software, while Citrix launched XenServer. The presence of multiple offerings in the market fostered competition and innovation, leading to advancements and improvements in virtualization technologies.

Alongside proprietary virtualization solutions, open-source options emerged. Projects like Xen (2003) and KVM (2007) provided open-source hypervisors that gained popularity due to their flexibility, performance, and compatibility with the Linux kernel. The advent of cloud computing further propelled the adoption of server virtualization. Virtual machines became a fundamental building block for cloud infrastructure, enabling the dynamic provisioning of resources and the efficient allocation of computing power.

In the 2010s, containerization technologies, such as Docker, emerged as an alternative approach to server virtualization. Containers offer lightweight, isolated runtime environments without the need for full OS virtualization. While different from traditional server virtualization, containers complement virtualization technologies and have found applications in areas like microservices and DevOps.

The emergence and adoption of these server virtualization technologies led to the consolidation of multiple VMs and containers on a single physical server. To enable communication between VMs and containers on the same server or across different servers, a virtual networking infrastructure was required.

In early 2000s, with the rise of x86 server virtualization technologies, such as VMware ESX, the need for virtual networking components emerged. Initially, basic bridging and networking capabilities were provided by the hypervisor itself, acting as a virtual switch.

In 2007, VMware introduced the Virtual Distributed Switch (VDS) as part of VMware Infrastructure 3. This virtual switch offered advanced networking features, including VLAN support, link aggregation, and traffic shaping. It enabled centralized management of virtual networks across multiple hosts.

In 2008, Microsoft introduced the Hyper-V Virtual Switch as part of its Hyper-V hypervisor. The virtual switch allowed virtual machines to communicate with each other and the physical network. It provided basic networking features such as VLAN support and traffic filtering.

In 2009, The Open virtual Switch (OVS) project was initiated by Nicira, then later was acquired by VMware. OVS aimed to provide a flexible, open-source virtual switch that could be integrated into virtualized environments. Soon after SDN emerged, OVS hugged SDN to support the programmability for SDN deployments. The OVS project was released as an open-source project under the Apache 2.0 license. This move encouraged community contributions and fostered the growth of OVS as a widely adopted virtual switch. OpenStack, an open-source cloud computing platform, integrated OVS as its default networking component. This integration further increased the visibility and adoption of OVS within the cloud computing ecosystem.

Open vSwitch provides the functionality of a traditional physical switch, but in a virtualized environment. By It allows for the creation of virtual bridges and ports to connect virtual machines (VMs) and containers within a network. OVS supports various networking protocols and features, including VLANs, VXLAN, GRE, QoS, access control lists (ACLs), and more.

With the rise of containerization and container orchestration platforms like Kubernetes, OVS extended its support to container-based networking. OVS became an essential component in providing networking connectivity and advanced features within containerized environments.

Today, server virtualization is a common practice in enterprise IT environments, with many organizations running hundreds or even thousands of virtual machines on a single physical server. The technology has continued to evolve, with new features such as live migration and disaster recovery capabilities that make it even more powerful and useful for businesses of all sizes.

2.2.2 Network virtualization

Network virtualization, on the other hand, is a technology that allows multiple virtual networks to be created on top of a physical network infrastructure. The concept of network virtualization has been around for several decades, with roots in the early days of computer networking. In the 1970s, researchers at Xerox PARC developed the first LAN, which allowed multiple devices to connect and communicate with each other. This early network architecture was the precursor to modern-day virtualization technologies.

In the 1990s, the advent of virtual private networks (VPNs) enabled users to securely connect to private networks over public networks such as the internet. VPNs allowed organizations to extend their private networks to remote locations and provide secure access to employees and partners.

In the early 2000s, advances in server virtualization technology paved the way for network virtualization. By decoupling the underlying physical infrastructure from the virtualized environment, server virtualization made it possible to create multiple VMs that could run on a single physical server.

The emerging of SDN separates the network control plane from the data plane, and enables operators to manage the network as a single entity rather than as a collection of individual devices. This approach enables greater flexibility and agility in network management, as well as improved scalability and network efficiency.

Today, network virtualization is a critical component of modern networking architectures, allowing operators to create flexible, scalable, and secure networks that can meet the needs of a wide range of applications and users. It enables network operators to abstract the underlying physical network and create logical networks that can be customized to meet the needs of specific applications or user groups.

Network virtualization can be achieved using various techniques, such as virtual local area networks (VLANs), virtual private networks (VPNs), and SDN. By virtualizing the network, operators can improve network utilization, increase flexibility, and reduce costs by sharing resources and optimizing network capacity. Virtual networks can also be isolated from each other to improve security and provide dedicated resources for specific applications or customers. Overall, network virtualization is an important technology for modern networks that require high levels of flexibility, scalability, and agility.

Virtualization can be viewed as part of an overall trend in enterprise IT. The usual goal of virtualization is to centralize administrative tasks while improving scalability and workloads. Server virtualization and network virtualization technologies are often used together in modern data center environments to create virtualized infrastructures that are more flexible, scalable, and efficient. For example, virtualized servers can be connected to virtual networks, which can then be easily configured and managed using network virtualization tools. This allows organizations to quickly and easily provision new resources, adjust network configurations, and respond to changing business needs.

By virtualizing servers and networks, organizations can more efficiently utilize their hardware resources. Instead of deploying one physical server or network device for each application or service, virtualization allows multiple instances to run on the same hardware, reducing the total number of physical devices needed.

Virtualization enables organizations to quickly and easily provision new resources, adjust configurations, and respond to changing business needs. This makes it easier to scale up or down as needed, without the need for additional physical infrastructure. By reducing the number of physical devices needed, organizations can save on hardware, power, cooling, and maintenance costs. Additionally, virtualization allows organizations to extend the life of their existing hardware investments by making more efficient use of the resources they already have. Network virtualization enables organizations to create isolated virtual networks, which can improve security and simplify network management. Similarly, server virtualization enables organizations to manage and secure their virtualized servers more easily.

Virtualization offers a wide array of advantages to both desktop users and server administrators. To desktop users, one of the primary benefits is the ability to run applications originally designed for a different operating system, all without the hassle of switching between computers or rebooting into an alternate system. This flexibility allows users to seamlessly access and use software that may not be native to their current environment, enhancing productivity and convenience.

For server administrators, virtualization is a powerful tool with multifaceted advantages. It enables the operation of diverse operating systems on the same physical server, thereby optimizing resource utilization. However, its impact extends far beyond this. Virtualization empowers administrators to segment a large server system into smaller, manageable partitions. This segmentation enhances overall server efficiency by ensuring that various users or applications with distinct requirements can coexist harmoniously on the same hardware. The server's resources are efficiently allocated, and the performance of each virtual environment is isolated from the others, contributing to a more stable and predictable server environment.

Additionally, virtualization offers a crucial feature: isolation. This feature ensures that programs running within a virtual machine remain completely insulated and secure from the activities taking place in another virtual machine on the same host. This isolation is especially valuable for enhancing security, as it prevents any potential vulnerabilities or issues in one virtual environment from affecting others. In summary, virtualization is a versatile technology that serves as a bridge between different operating systems for desktop users and as a powerful resource management and security tool for server administrators.

While network virtualization and server virtualization offer many benefits, there are also some limitations. Virtualization requires additional resources to support the virtualization layer, which can result in increased resource overhead and decreased performance compared to running applications directly on physical hardware. Furthermore, the virtualization both in network and server introduce the complexity to the intra-server and inter-server communication. The CPU resources are occupied for the virtualization itself which stimulate new hardware and technologies to offload the processing in the CPU and increase the throughput of the whole server, Smart NIC is one of the technologies and hardware.

2.3 The road to smarter network interface card: NIC to Smart NIC and beyond

As mentioned above, web scale companies, with their immense scale and data-intensive operations, have unique requirements that traditional networking technologies often struggle to address effectively. They require cost-effective solutions that can handle their high volumes of traffic, provide scalability, and offer flexibility to adapt to their dynamic environments.

To meet these demands, web scale companies have pushed for innovation and openness in networking technologies. They have advocated for more accessible and customizable solutions that empower them to optimize their networks for their specific needs. This demand has led to the development and adoption of technologies like smart NICs.

Smart NICs offer benefits such as offloading processing tasks, hardware acceleration, enhanced security features, and network analytics. These capabilities align with the requirements of web scale companies, allowing them to improve network performance, increase efficiency, and optimize their operations.

Moreover, the demands of web scale companies have driven increased competition in the networking market. Traditional telecom companies and networking vendors have had to respond to these demands by introducing more innovative and cost-effective solutions. This competition has spurred further advancements in networking technologies, including smart NICs, as companies strive to differentiate themselves and cater to the evolving needs of the industry.

The democratization of networking technologies, fuelled by the demands of web scale companies, has led to a more competitive and innovative landscape. It has opened opportunities for smaller players and startups to enter the market with disruptive solutions, challenging the dominance of traditional telecom providers. This increased competition and innovation benefit not only web scale companies but also other industries and end-users who can leverage more efficient and cost-effective networking technologies.

2.3.1 Network interface card at a glance

NIC, alternatively referred to as a network adapter, is a hardware component that plays a pivotal role in enabling a computer to establish connections with a network. Typically situated within a computer's hardware, the NIC functions as the physical intermediary, bridging the computer and the network. Its primary duty is to furnish a seamless communication channel that facilitates the transmission and reception of data across the network. The NIC establishes its connection to the computer's motherboard via a bus, such as PCI or PCIe, and is usually equipped with one or more connectors for attaching network cables.

The inception of the NIC can be traced back to the 1970s, a time when computer networks were in their nascent stages of development. It was during this era that the first network adapters emerged. These early adapters typically took the form of expansion cards that integrated with the computer's bus architecture, including notable standards like the S-100 bus and the ISA bus. Their fundamental purpose was to establish rudimentary connectivity to the early iterations of local area network (LAN) technologies, such as Ethernet and Token Ring.

In the 1980s, Ethernet became the dominant LAN technology. NICs evolved to support Ethernet standards, such as 10BASE5 and 10BASE2 coaxial cables. The NICs included connectors for attaching the appropriate Ethernet cable and provided the necessary circuitry for data transmission and reception.

Throughout the 1990s and early 2000s, NICs saw significant advancements in terms of speed and standards. Faster Ethernet standards like 100BASE-TX and Gigabit Ethernet emerged, offering higher data transfer rates. NICs also began supporting additional networking technologies like Fast Ethernet, Token Ring, and FDDI (Fiber Distributed Data Interface).

NICs continued to evolve to support higher-speed Ethernet standards like 10GbE and even faster technologies like 25GbE, 40 GbE, and 100 GbE. NICs can support a variety of network protocols and standards, such as Ethernet, Wi-Fi, and Bluetooth. With the increasing of the bandwidth and the low latency and high throughput requirements, some NICs include features such as offloading, which allows the NIC to perform certain processing tasks, such as packet checksum calculations, to free up CPU resources on the computer. The NIC itself for a quite long period, although it is an essential component for enabling computers to connect to and communicate over a network, it is transparent to the users or network administrators.

2.3.2 Smart NIC

In some use cases and scenarios, there are requirements to contain advanced hardware such as CPUs, memory, and programmable logic to perform network-related tasks in the NIC with minimal involvement from the host CPU. It is so called Smart NIC, which history can be traced back to the development of specialized network processors in the 1990s.

One of the earliest examples of a smart NIC was the iWarp adapter, which was developed by Intel in the early 2000s. [64] The iWarp adapter was designed to offload TCP/IP processing from the host CPU and improve network performance by using a specialized network stack implemented in hardware. The iWarp adapter also included a programmable logic unit that allowed developers to customize the adapter's behavior to suit their applications.

In the years that followed, several other vendors, such as Chelsio Communications, Netronome, and Mellanox Technologies, entered the market and began developing their own smart NIC solutions. These smart NICs offered various features such as offloading of security protocols, acceleration of virtualization tasks, and support for software-defined networking (SDN).

In recent years, the adoption of smart NICs has increased significantly, especially in data center environments. This is partly due to the rise of virtualization and containerization technologies, which have increased the demand for high-performance networking solutions that can handle large volumes of traffic with minimal overhead. Smart NICs can offload a variety of network processing tasks from the host CPU, such as TCP/IP processing, encryption and decryption, and packet filtering. In the meantime, Smart NICs can use specialized hardware to accelerate network performance by optimizing network traffic flows, reducing latency, and increasing throughput. This can be particularly useful in high-performance computing environments where low-latency network communication is critical. Because of the evolution of the network agility and programmability, Smart NICs can provide support for advanced networking features such as SDN, network virtualization, and network overlays. These features can help to improve network flexibility and scalability and enable the creation of more complex network topologies. With the nature of the only network path in the server, Smart NICs can enhance server security by provide hardware-based security features such as encryption and decryption, firewalling, and intrusion detection and prevention. This can help to improve network security and protect against various types of cyber threats. With the distributed computing becomes more and more popular, especially for AI, Smart NICs can be used to support distributed computing environments, such as those based on the message passing interface (MPI) or remote direct memory access (RDMA). This can help to improve the performance and scalability of distributed applications and enable the use of new programming models and paradigms.

To the date of writing the thesis, there are a few types of Smart NICs. The comparation is in Table 1.

Table 1:Typical Smart NIC types [77] [78] [79]

Types	Benefits	Weakness
FPGA-based	High flexibility: FPGA-based smart NICs can be programmed to perform a wide range of network processing tasks, allowing for greater flexibility in network configuration and optimization than other three types of Smart NICs. High performance: FPGAs are designed for parallel processing and can perform network processing tasks much faster than traditional CPUs, resulting in improved network performance. Lower latency than CPU-based network processing: By offloading network processing tasks to the smart NIC, the time it takes for the CPU to receive and process network traffic can be reduced, resulting in lower latency than NP-based and SoC-based Smart NICs. Low power consumption: FPGAs can perform network processing tasks using less power than NP-based, SoC-based, resulting in improved energy efficiency.	High cost: FPGA-based smart NICs can be more expensive than other types of Smart NICs, as they require specialized FPGA hardware and software. Complex programming: Programming FPGAs can be more complex than programming other three types of smart NICs, requiring specialized knowledge and tools. Limited availability: FPGA-based smart NICs may not be as widely available as other three types of Smart NICs, limiting the range of hardware options for system architects and designers.
NP-based	Increased network performance: NP-based smart NICs can offload network processing tasks from the CPU, improving network performance and reducing CPU utilization. Lower latency than CPU-based network processing: By offloading network processing tasks to the smart NIC, the time it takes for the CPU to receive and process network traffic can be reduced. NP-based smart NICs had lower latency than SoC-based smart NICs, but higher than FPGA-based or ASIC0based Customizability: NP-based smart NICs can be customized and programmed to perform specific network processing tasks, allowing for optimization of network performance.	Limited functionality: NP-based smart NICs are designed to offload network processing tasks, and as such, may not be able to handle other processing tasks that the SoC-based and FPGA-based Smart NICs can perform. This can result in a more limited functionality for the overall system. Cost: NP-based smart NICs can be more expensive than ASIC-based and SoC-based smart NICs, as they require specialized network processors. Compatibility: NP-based smart NICs may not be compatible with all types of systems, and may require specific hardware and software configurations to work properly.
ASIC-based	High performance: ASIC-based smart NICs are designed specifically for network processing tasks and can perform these tasks much faster than traditional CPUs or other three types of smart NICs, resulting in improved network performance. Lower latency than CPU-based network processing: Specifically designed for network processing tasks, ASIC-based smart NIC could achieve the lowest latency than other three kinds of smart NICs.	Limited flexibility: ASIC-based smart NICs are designed for specific network processing tasks and cannot be reprogrammed to perform different tasks like FPGA-based smart NICs. High non-recurring engineering (NRE) cost: ASIC-based smart NICs can be more expensive in NRE than other three types of smart NICs, as they require integrated circuit design and tape-out. Long lead times: The development and production of ASIC-based smart NICs can take longer than other types of smart NICs,

	Energy efficiency: ASIC-based smart NICs can perform network processing tasks using less power than traditional CPUs or other three types of smart NICs, resulting in improved energy efficiency and lower operating costs. Low programming complexity: ASIC-based smart NICs do not require specialized programming like FPGA-based smart NICs, making them easier to integrate into network architectures than other three types smart NICs.	which may impact their availability and adoption in the market.
SoC-based	Integration: SoC-based smart NICs offer integrated processing capabilities that can include CPUs, GPUs, and other processing units along with network processing, leading to a smaller system footprint. Lower latency than CPU-based network processing: By offloading network processing tasks to the smart NIC, the time it takes for the CPU to receive and process network traffic can be reduced, resulting in lower latency and improved overall system performance. However, the SoC-based smart NIC could lead to higher latency than other three types of smart NICs. Customizability: SoC-based smart NICs can be customized and programmed to perform specific network processing tasks, allowing for optimization of network performance. Cost-effective: SoC-based smart NICs can be more cost-effective than other types of smart NICs, as they integrate multiple processing units onto a single chip.	Limited flexibility: SoC-based smart NICs are designed for specific network processing tasks and may not be as flexible as other three types of smart NICs. Performance limitations: SoC-based smart NICs may not be able to achieve the same level of performance as dedicated network processing units like ASICs or FPGAs. Complexity: The integration of multiple processing units onto a single chip can make SoC-based smart NICs more complex to design and program.

As described in Table 1, in general, FPGA-based smart NICs are based on FPGAs and can be programmed to perform a variety of network processing tasks. They offer high flexibility and can be reprogrammed to adapt to changing network requirements. NP-based smart NICs offer a way to improve network performance and reduce latency by offloading network processing tasks from the CPU. However, they may have limited functionality, be more expensive, and require specific hardware and software configurations. The suitability of NP-based smart NICs depends on the specific network requirements and system architecture. ASIC-based smart NICs offer high performance and energy efficiency for specific network processing tasks but may have limited flexibility and higher costs. The suitability of ASIC-based smart NICs depends on the specific network requirements and system architecture. SoC-based smart NICs offer a cost-effective way to integrate network processing capabilities along with other processing units on a single chip, leading to smaller system footprints. However, they may have limited flexibility and performance compared to dedicated network processing units like ASICs or FPGAs. The suitability of SoC-based smart NICs depends on the specific network requirements and system architecture.

There is no universally accepted standard definition of a Smart NIC. The term "Smart NIC" generally refers to a network interface card that goes beyond the traditional functions of a standard NIC and includes additional processing capabilities.

2.3.3 *Beyond Smart NIC*

As data processing requirements grow in fields such as artificial intelligence (AI), machine learning (ML), and big data analytics, traditional processors like CPUs and GPUs are struggling to keep up. The definition of DPU emerged because of the increasing demand for efficient and specialized hardware accelerators to handle the rapidly growing volume and complexity of data processing tasks.

The history of DPUs can be traced back to the emergence of specialized ASICs such as GPUs and FPGAs in the early 2000s. These processors were designed to offload specific tasks from general-purpose CPUs, such as graphics rendering, machine learning, and encryption. Over time, the demand for specialized processing units grew, leading to the development of more advanced processors such as tensor processing units (TPUs) and digital signal DSPs.

DPUs are specialized hardware accelerators that are designed to offload specific data processing tasks from general-purpose processors, allowing them to perform these tasks more efficiently and effectively. A DPU typically includes multiple processing elements such as CPU, memory, networking interfaces, and accelerators such as GPUs or FPGAs, all integrated into a single chip. The purpose of a DPU is to offload data processing tasks from the CPU, freeing up the CPU to handle other system tasks and improving overall system performance.

DPUs are often used in modern data center architectures to accelerate tasks such as network virtualization, security processing, and AI inference. They are also used in edge computing devices to perform complex data processing tasks locally, without the need to send data to remote data centers for processing.

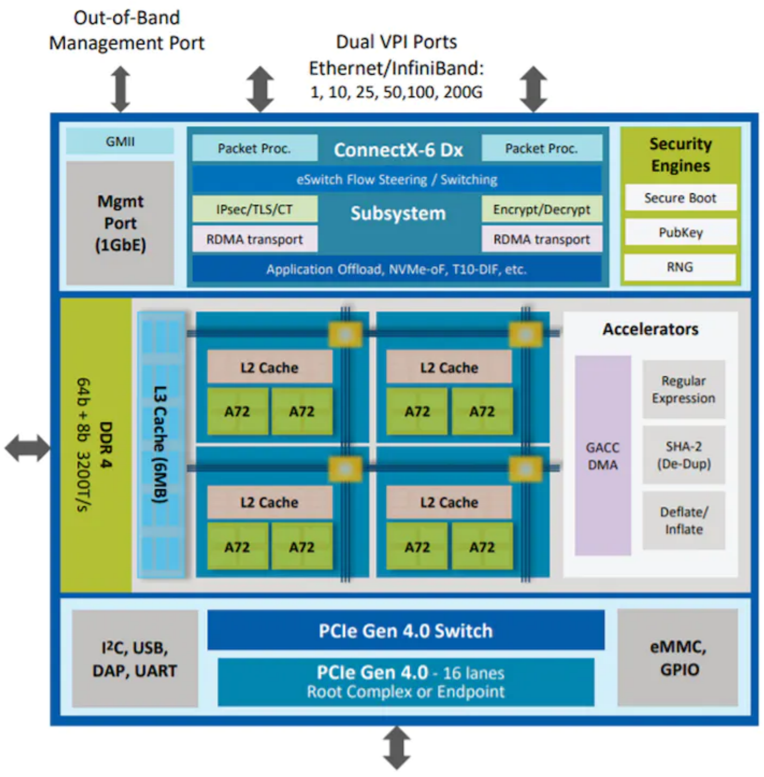


Figure 5:Mellanox's Bluefield 2 Architecture as taken from the product datasheet.[65]

In the contemporary market, notable DPU (Data Processing Unit) products include NVIDIA's BlueField DPUs and Intel's IPUs. NVIDIA, a prominent player in this domain, outlines the primary functions of DPUs as encompassing processing, networking, and acceleration. These DPUs are typically realized as System on a Chip (SoC) designs, uniting three core components:

Multi-core CPU: DPUs feature a high-performance, multi-core CPU that adheres to industry standards. This CPU is known for its software programmability and is usually built upon the Arm architecture, a type of Reduced Instruction Set Computing (RISC). This multi-core CPU serves as the computing powerhouse within the DPU, capable of executing diverse software applications and workloads.

High-Performance Network Interface: DPUs are equipped with a high-performance network interface that excels in data analysis, processing, and efficient data transfer at line rate—matching the speed of the network. This network interface plays a pivotal role in ensuring that data can be seamlessly transmitted between the DPU, GPUs, CPUs, and the network, enabling rapid data processing and transfer.

These combined elements empower DPUs to significantly enhance data processing, networking capabilities, and acceleration functions. As a result, DPUs have become indispensable components in contemporary computing and networking infrastructure. The synergy of a robust multi-core CPU, an efficient network interface, and potent acceleration capabilities allows DPUs to consistently deliver peak performance across a wide spectrum of applications and workloads.

Chapter 3. FPGA-BASED SMART NIC SOLUTION DESIGN AND IMPLEMENTATION

At the time of writing the thesis, there was no standard definition of a Smart NIC. The term Smart NIC was used to refer to NIC that were capable of offloading network processing tasks from the host CPU. The specific functions of a Smart NIC varied depending on the application, and there was no clear consensus on what constituted a Smart NIC. The PhD focused specifically on FPGA-based Smart NICs for intra-server and inter-server acceleration. Smart NICs for offloading network processing tasks from the host CPU, enabled faster and more efficient data transfer. As the technologies background mentioned in the previous chapter, the usage of the NIC itself is inserted into the server, communicating, and collaborating with CPU in the server. By offloading network processing tasks from the host CPU, Smart NICs can reduce network latency and enable more efficient use of resources, ultimately leading to better performance and increased scalability.

This chapter, being the core of the paper, focused on the Smart NIC solution design and implementation. It listed the Smart NIC design considerations, the architectures and detailed design. The PhD work focused mainly on the FPGA-based firmware design and architecture.

3.1 Smart NIC design and implementation considerations

3.1.1 FPGA-based Platform as the Smart NIC platform

The FPGA-based platform was chosen as the Smart NIC solution platform for its programmability, performance, flexible network interfaces, and homogeneous.

One of the main advantages of using FPGA-based platform for Smart NIC implementation is its programmability. FPGAs can be reprogrammed to adapt to changing network requirements and to implement innovative ideas. The ability to reprogram the FPGA and modify the smart NIC functionality is the key advantage of FPGA-based Smart NIC enabled DCN architecture. The requirements of the FPGA-based Smart NIC solution should be flexible and programmable to support various use cases and adapt to changing network requirements. This flexibility allows for greater customization and futureproofing of the network infrastructure. This allows for greater customization and flexibility in the Smart NIC design.

Another advantage of FPGA-based platform is its parallel processability. FPGAs can handle multiple tasks in parallel, which makes them highly efficient at processing large amounts of data in real-time. FPGAs processing time is in nanoseconds, which is much faster than traditional processors such as CPUs or GPUs. This allows the Smart NIC to quickly process and transmit data, reducing latency and improving overall network performance.

In addition, FPGA-based platform supplied flexible network interfaces, for example, the SERDES of 28Gbps, could be implemented to 10GbE, 25GbE, 40GbE, 100GbE Ethernet interface, which made it easy to demonstrate the Smart NIC solutions in various use cases.

Last but not the least, for the Smart NIC used mainly in the data center environment, the FPGA-based Smart NIC could supply homogeneous for the DCN. The identical FPGA-based platforms could be inserted to the servers, programmed with different firmware, and running diverse applications.

Overall, the FPGA-based platform is a powerful and ideal solution for Smart NIC implementation, especially in the PhD research.

3.1.2 FPGA-based Smart NIC enabled all-optical intra-rack communication

The proposed FPGA-based Smart NIC intra-rack architecture is shown in Figure 6, the highlight of the design includes the intra-rack server-to-server direct interconnect, which can minimize the intra-rack communication latency.

The design draws inspiration from the "Bump in the Wire" [30] [31] architecture, featuring a comprehensive mesh interconnect within the rack's servers. This setup enables direct communication among the servers within the same rack, bypassing the need to traverse through the ToR switch. By taking this approach, it mitigates over 80% of the intra-rack link latency that would have been introduced if the communication had to pass through the ToR switch.

To facilitate the intra-rack communication, an FPGA-based Smart NIC was designed with two transceiver interfaces, one to connect with the existing NIC and the other to connect with the ToR switch. The FPGA communicates with the server through PCIe interface. By employing the FPGA-based Smart NIC in line between the normal NIC and the ToR switch, the Server-FPGA bandwidth was doubled since it introduced the normal NIC in the loop. This increased the actual bandwidth from the Smart NIC to the server, which could go beyond 200 Gbps when calculated with PCIe Gen3*16lanes or Gen4*8lanes bandwidth. This ease the bandwidth pressure between the transceivers and the servers, which is critical in this architecture.

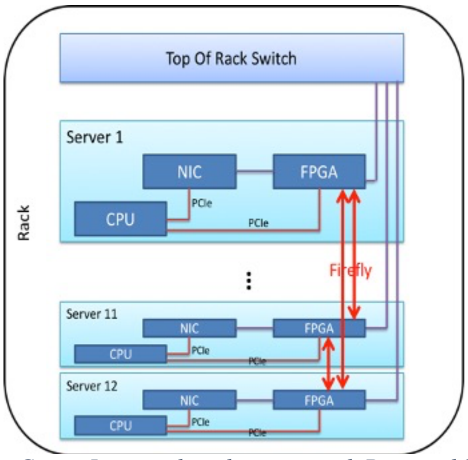
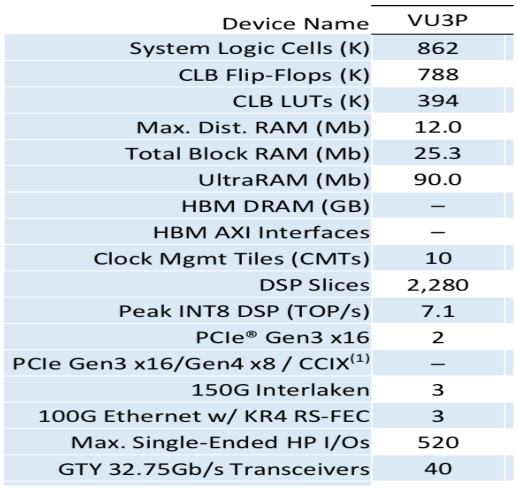


Figure 6:Data Center Intra-rack architecture with Proposed Smart NIC

3.1.3 FPGA Selection

The PhD research work started in 2018, at that time, Xilinx had their newest generation Ultrascale+ FPGA out with higher performance and lower latency than Virtex 7 FPGAs. Therefore, I chose Xilinx Ultrascale+ FPGA as the main chip of the Smart NIC. It had high-speed transceivers that support speeds of up to 32.75 Gbps, making them suitable for applications that require high-speed data transfer. It also supports a wide range of high-speed interfaces, including PCIe, DDR4, and 100G Ethernet.

The Xilinx Ultrascale+ series devices ranged from 862K system logic cells to 3780K, I chose VU3P(resource shown in Figure 7) featured with 862K System Logic Cells, 12Mb Dist. RAM, 25.3Mb Block RAM, 90Mb UltraRAM, PCIe Gen3x16 and 40 GTY 32.75Gbps Transceivers. The RAMs could be used for buffering the packets, the 16 PCIe Gen3 lanes could meet the requirement of 100Gbps traffic, and 40 GTY 32.75Gbps Transceivers supplied variable intra-rack card-to-card communication architectures. Considering the design size and the cost of the Smart NIC hardware, VU3P is the most economical and powerful choice in the Ultrascale+ family.



Device Name	VU3P
System Logic Cells (K)	862
CLB Flip-Flops (K)	788
CLB LUTs (K)	394
Max. Dist. RAM (Mb)	12.0
Total Block RAM (Mb)	25.3
UltraRAM (Mb)	90.0
HBM DRAM (GB)	–
HBM AXI Interfaces	–
Clock Mgmt Tiles (CMTs)	10
DSP Slices	2,280
Peak INT8 DSP (TOP/s)	7.1
PCIe® Gen3 x16	2
PCIe Gen3 x16/Gen4 x8 / CCIX$^{(1)}$	–
150G Interlaken	3
100G Ethernet w/ KR4 RS-FEC	3
Max. Single-Ended HP I/Os	520
GTY 32.75Gb/s Transceivers	40

Figure 7: Xilinx Ultrascale+ VU3P resources

3.1.4 FPGA-based platform features

1. *For the FPGA-based Smart NIC implementation, I initiated the hardware requirements and assigned the FPGA pins for a new FPGA-based hardware platform with chosen Xilinx Ultrascale+ VU3P FPGA. A variety of factors were taken into account, including intra-rack architecture, network interface, processing requirements, programmability and flexibility, then a hardware engineer designed the required FPGA-based Smart NIC platform as shown in Figure 8.*

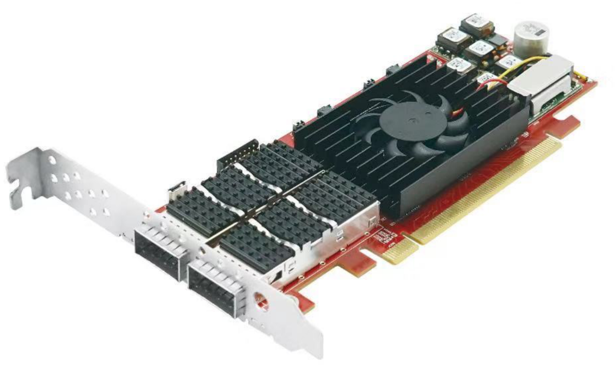


Figure 8:FPGA-based Smart NIC Platform

The FPGA-based hardware platform was with the features below for the intra-rack and inter-rack acceleration:

a. Firefly-enabled multi-lanes 25Gbps link enabling Data Cetner Intra-rack networking Architecture: The smart NIC serves as a connection point within the data center rack and enables intra-rack communication. Considering the architecture i ensure that the smart NIC can efficiently route traffic between servers within the rack, reducing latency and improving overall performance.

b. QSFP28-enabled 100Gbps link enabling high speed Network Interface: The smart NIC must support the desired network interface, such as Ethernet or InfiniBand. The choice of interface can also impact the FPGA selection and design considerations. The design ensured that the smart NIC is compatible with the network infrastructure and protocols used within the data center or cloud environment.

c. Ultrascaleplus-enabled processing capabilities enabling real-time parallel processing: The processing requirements of the smart NIC will depend on the intended use case. For example, a smart NIC used for packet filtering and load balancing will have different processing requirements than one used for encryption/decryption. The design should ensure that the FPGA and associated components can handle the processing requirements of the specific use case. However, the Ultrascale+ VU3P supplied almost a thousand FPGA System Logic Cells that which the use cases mentioned above possible.

d. DDR4-enabled Storage capabilities enabling buffering the 100Gbps traffic on the board: Basically, the Smart NIC is a NIC that need to deal with the incoming and outgoing traffics of the servers. With 100Gbps traffic, no matter how big the storage on the Smart NIC card is, the traffic would easily flush the card. Nevertheless, although on-board-DDR4 memories couldn’t store seconds of data, but it could made the card be able to tolerant the network jitter.

e. Low Power Consumption Design: The FPGA is the key component of the Smart NIC and its selection is critical to the performance and efficiency of the system. High-speed transceivers and high logic capacity are generally preferred, as they enable the smart NIC to process and transmit data at high speeds. However, power consumption is an important consideration, especially in power-constrained environments such as data centers. The FPGA and associated components were designed for low power consumption, while still maintaining high performance.

3.1.5 FPGA-based Smart NIC architecture

The nature of the Smart NIC is inserting into the PCIe slot of a server and collaborating resiliently with CPU, memory, and other server components. Therefore, the FPGA-based Smart NIC is not just a standalone device. The Smart NIC was designed to offload the CPU which had close connection with the software workloads. Considering both FPGA-based firmware part and the software part, the high level system architecture design was demonstrated in the Figure 9.

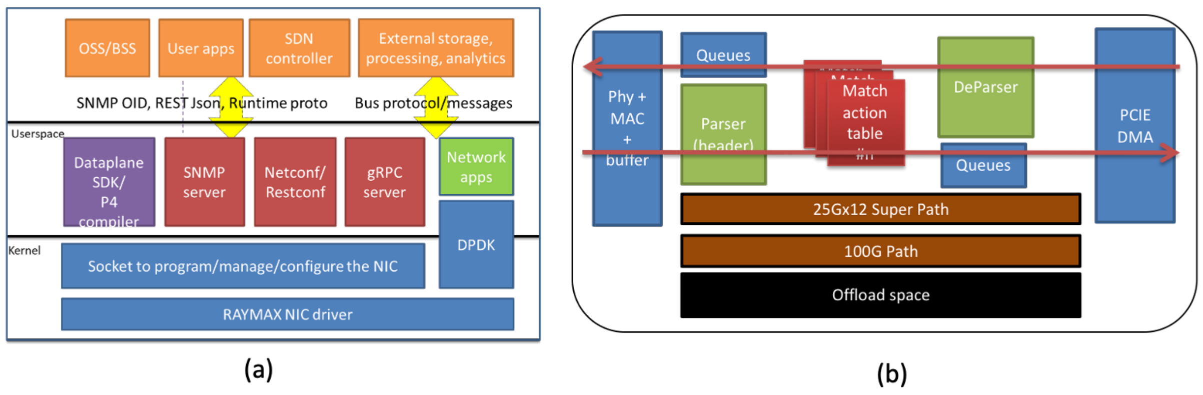


Figure 9:(a) Software architecutre (b) FPGA-based firmware architecutre

The software component, depicted in Figure 9(a), encompasses open networking initiatives and established protocols, including the P4 compiler, ONOS, DPDK, and interfaces like gRPC. This software segment plays a pivotal role in facilitating communication between the FPGA-based Smart NIC and the server.

The FPGA-based hardware component, as illustrated in Figure 9(b), primarily handles network dataplane processing. The subsequent sections below provide comprehensive insights into the system architecture design of the FPGA-based Smart NIC.

3.2 FPGA-based Smart NIC System software architecture design

The software part of the FPGA-based Smart NIC was displayed in Fig. IT consists the Kernel-based driver part, the user-space part and the APP part. The idea of the software part of the Smart NIC architecture design is using the open-source projects. And the main design works are FPGA-based Smart NIC DPDK driver, and the P4C agent.

3.2.1 FPGA-based Smart NIC DPDK driver

For running and testing FPGA-based Smart NIC, DPDK drivers need to be developed. Furthermore, a P4 agent needs to be designed and implemented after the P4C to translate the json file to the binary files that FPGA can understand.

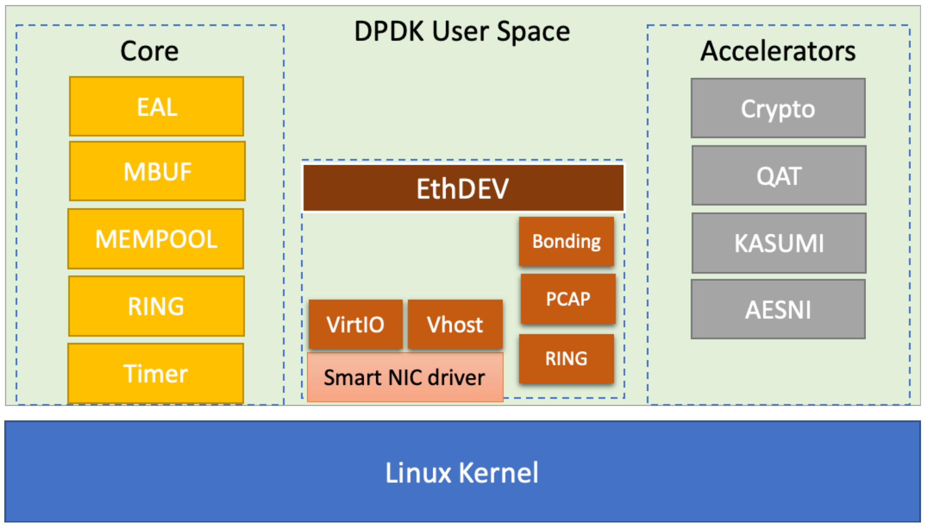


Figure 10:Smart NIC DPDK Driver

The Smart NIC driver for DPDK was designed to support polling mode, which allows for more efficient data transfers between the Smart NIC and the application. The driver layer, displayed in Figure 10, operates several key components such as the initialization of the igb_uio module, addition of devices, PCIe device driver attribute definitions, and the ability to perform binding operations to trigger driver probes. Additionally, the driver can perform unbind operations to find a concrete device abstraction through the passed device identity and release it.

The igb_uio module initialization is an important step in the driver layer operations as it allows the driver to communicate with the hardware and perform low-level operations. This module is responsible for providing a user space driver for devices that do not have native support in the Linux kernel.

The addition of devices is another important aspect of the driver layer operations. This step involves identifying and registering the devices that the driver will interact with. Once a device is added, the driver can start sending and receiving data packets to and from the device.

PCIe device driver attribute definitions are also a key component of the driver layer operations. These definitions specify the properties of the device and its capabilities. This information is used by the driver to determine how best to communicate with the device.

The ability to perform binding operations is critical to the driver's functionality. This process involves associating a driver with a specific device. When a device is bound to a driver, the driver is responsible for managing the device's communication with the rest of the system.

The driver can perform the unbind operation, which allows it to release a device abstraction by passing a device identity. This process frees up system resources and ensures that the device is no longer tied to a specific driver.

3.2.2 Socket to program and manage the FPGA-based Smart NIC

There are a few sockets I developed for program and manage the FPGA-based smart NIC.

3.2.2.1 SNMP

SNMP, or "Simple Network Management Protocol," is a vital application layer protocol that forms an integral part of the internet protocol suite—a comprehensive ensemble of communication protocols widely employed in the realm of networking. Its fundamental purpose lies in the systematic collection of data pertaining to changes occurring within a network and the real-time status of network-connected devices. This profound capacity for data acquisition equips IT professionals with the tools needed to maintain a comprehensive and informed overview of all the devices and applications they oversee.

SNMP operates as a dynamic instrument for making inquiries and assessments across the network landscape. By utilizing SNMP, TCP, and various other probe types, it facilitates real-time queries to every device within the network, affording administrators a wealth of performance metrics. Moreover, SNMP is equipped with an intelligent system of predefined thresholds that, when surpassed by specific values, serve as triggers for notification mechanisms. These notifications, often executed through specialized software, swiftly apprise system administrators of emerging issues, granting them the ability to access pertinent data and promptly rectify the problem.

The mechanism through which SNMP operates is reliant on the transmission of messages referred to as Protocol Data Units (PDUs) to devices in the network that offer support for SNMP. These messages, more precisely designated as SNMP Get-Requests, serve as the communication bridge between administrators and the network devices. By deploying these requests, network administrators can systematically monitor virtually any data values they wish to track, providing a dynamic means of surveillance and control.

The informational wealth gathered by SNMP isn't confined within its own ecosystem. Instead, it is designed to be made available to requesting products or applications, fostering a cooperative environment where data can be seamlessly disseminated to meet the specific needs of administrators. This flexible framework enables administrators to choose whether to display the real-time data or archive it for subsequent analysis, in line with the demands of the ever-evolving network landscape. Thus, SNMP remains a pivotal element in the arsenal of network management tools, underpinning efficient and responsive administration across diverse IT ecosystems.

3.2.2.2 Netconf

The Network Configuration Protocol, more commonly known as NETCONF, stands as a pivotal internet management protocol that has been meticulously crafted and refined by the esteemed Internet Engineering Task Force (IETF). Its primary mission revolves around delivering a secure and methodical approach to handle the installation, alteration, and removal of configuration data on network devices, thereby enhancing network management on a profound scale.

At the heart of NETCONF's operation lies the utilization of the Remote Procedure Call (RPC) protocol. This RPC protocol serves as the indispensable bridge that enables seamless and secure communication between clients and servers within the network infrastructure. One of the paramount advantages of RPC is its ability to facilitate program-to-program interactions without requiring an in-depth understanding of the underlying network intricacies. Essentially, it permits one program to solicit services from another program, ushering in a world of seamless interaction and cooperation.

A defining feature of the RPC messages is their encoding in Extensible Markup Language (XML). This choice of format not only ensures that data is structured and easily comprehensible but also allows for the inclusion of meta-information to accompany the payload. The result is a versatile and powerful means of transmitting instructions and queries within the network.

Moreover, these RPC messages traverse the network within the confines of secure, connection-oriented sessions. This added layer of security bolsters the integrity of the communication, making it resistant to common network issues and vulnerabilities.

In essence, NETCONF, with its reliance on RPC and XML, provides a robust foundation for network management. It streamlines the handling of configuration data on network devices, simplifies program-to-program communication, and maintains a strong emphasis on security. As networks continue to grow in complexity and importance, NETCONF's role in shaping efficient, secure, and structured network management becomes increasingly vital. It exemplifies the commitment of the IETF to elevating internet management to new heights.

3.2.2.3 gRPC

gRPC, an innovative technology, is tailor-made for the seamless implementation of Remote Procedure Call (RPC) APIs, and it does so by building on HTTP 2.0 as its foundational transport protocol. At first glance, the choice of gRPC and HTTP 2.0 may seem counterintuitive because they seemingly operate on disparate conceptual models. However, this apparent contrast in models is precisely what makes gRPC unique and versatile.

In the realm of gRPC, the architecture adheres to the RPC model. Here, procedures take center stage as the primary addressable entities, and the data resides within these procedures. This design revolves around the idea of method invocation, where clients make requests to specific methods or functions on the server, passing data and expecting a response. The RPC model is inherently service-oriented, focusing on operations and interactions, often aligning closely with programming paradigms.

Conversely, HTTP, the backbone of web communication, is rooted in a different paradigm. In the world of HTTP, the fundamental entities are data resources, which are often referred to as "resources" in the specifications. These resources are addressable entities, and their behaviours are hidden behind the data. The core principle here is that the web's functionality is achieved through the creation, modification, and deletion of these resources, typically expressed through HTTP methods like GET, POST, PUT, and DELETE.

In essence, gRPC not only bridges the gap between RPC and HTTP but also pushes the boundaries of modern network communication. It brings the structured and efficient RPC model into the world of web-based applications, offering a versatile and compelling approach to building robust and high-performance APIs. Its adoption is particularly advantageous in situations where real-time interactions and low latency are critical, underscoring its significance in the ever-evolving landscape of network technologies.

3.2.3 P4 compiler and agent

There are a few open source projects for P4 compiler [66], such as:

P4C : P4C is a P4 compiler for programming network devices. P4C compiles P4 code into binary code that can be run on network devices. The compiled code can be used to program various data plane devices such as switches, routers, and NICs. By using P4C, network engineers can specify how packets are processed by the network, allowing them to customize and optimize network behaviour for their specific needs.

P4c-bm2-ss: P4c-bm2-ss is a variant of the P4c compiler that generates programs specifically for the Barefoot Tofino switch chip. It is optimized to generate efficient code for Tofino's programmable pipeline architecture and provides a high-level interface to program the switch hardware.

P4c-ebpf: P4c-ebpf is a P4 compiler that generates eBPF (extended Berkeley Packet Filter) programs that can be used in Linux-based systems. It translates P4 programs into eBPF bytecode, which can be loaded into the Linux kernel to perform packet processing in user space. P4c-ebpf is particularly useful for implementing network functions in Linux-based systems, such as firewalls, load balancers, and intrusion detection systems.

P4c-xdp: P4c-xdp is a P4 compiler that generates programs for the XDP (eXpress Data Path) framework used in Linux-based systems. It generates eBPF bytecode that can be loaded into the XDP framework to perform fast packet processing in the Linux kernel. P4c-xdp is particularly useful for implementing high-speed network functions in Linux-based systems, such as DDoS protection and load balancing.

In summary, P4c-bm2-ss targeted Tofino swtich chip, P4c-ebpf and P4c-xdp targeted Linux-based systems. P4C is more general, targeted multiple backends. There are also a few other P4 compilers, such as P4@ElTE [67], P4FPGA [68], PISCES [69], and so on. Most of these tools use the P4c frontend provided by the P4 Language Consortium [70] as a starting point to reduce development effort. Therefore, P4C was decided to be used in the Smart NIC solution, as it can compile P4 code into binary code, however, the binary code cannot be read directly by the FPGA-based Smart NIC. To support the proposed FPGA-based Smart NIC, a backend solution needs to be built to interpret the P4 code into registers and LUTs that the FPGA can understand. Additionally, a P4 compiler agent needs to be developed to translate the compiled JSON file into a P4 binary file that the FPGA can understand.

3.3 *FPGA-based Smart NIC Firmware Architecture Design*

The PhD work focused more on the FPGA-based design, which is responsible for implementing the offload and low latency work.

To achieve ultra-low intra-rack latency, the system design includes a super path for 25Gbps intra-rack server-to-server communication data paths, which has the highest priority. In contrast, the ToR-server path, shown as the 100Gbps path in Figure 11, is designed as a service-driven path that supports P4 and virtual switch data paths for CPU offload and multiple-VM applications. The FPGA-to-NIC path follows the same design principles.

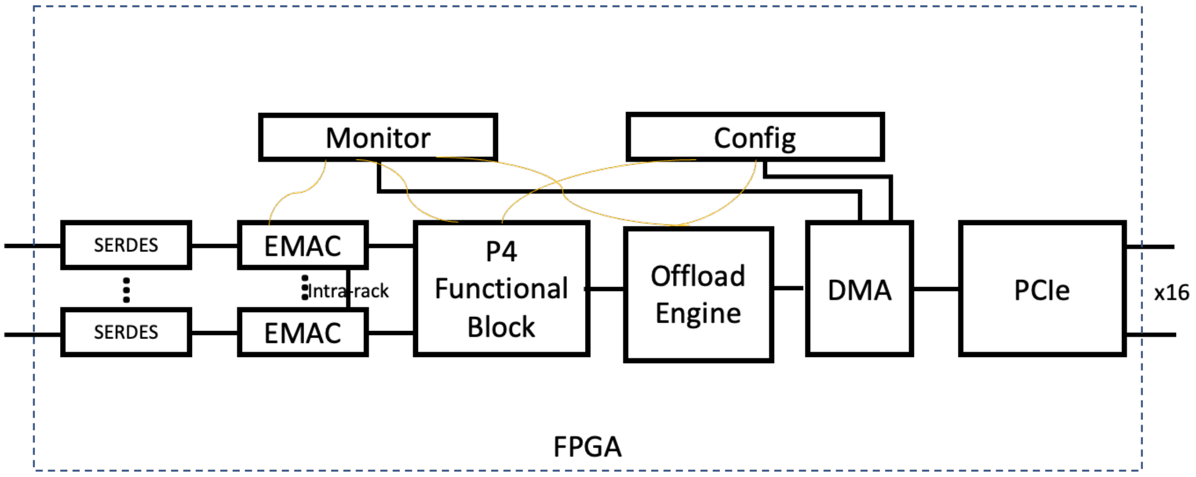


Figure 11:P4-enabled FPGA-based Smart NIC architecture in a glance

3.3.1 *Serdes and Ethernet MAC*

While I use Xilinx Ultrasclae+ VU3P FPGA as the main chip for the Smart NIC design and development, the SERDES and Ethernet MAC employed in the design for both 100GbE and 10GbE/25GbE was from Xilinx. The Xilinx 25G/100G Ethernet IP core are programmable Ethernet MACs that can be used with Xilinx FPGAs to enable high-speed Ethernet connectivity. The core supports full-duplex operation at speeds of up to 25 Gbps/100Gbps, and includes features such as VLAN tagging and filtering, Jumbo Frame support, and Quality of Service (QoS) features. The Xilinx 25G/100G Ethernet IP core also includes support for the IEEE 1588 Precision Time Protocol

(PTP), which enables accurate synchronization of clocks across a network. This is an important feature for many high-performance networking and communication systems.

In the design, I utilized the statistics gathered from the MAC IP core, such as total packets, good packets, bad FCSs and so on. The features help for the flow visibility and debugging.

3.3.2 P4 functional Block

The FPGA-based Smart NIC offers programmability and flexibility, rendering it well-suited for the P4 data plane. Conforming to the most recent P4_16 language specification, the FPGA was used to implement the P4 data plane block. Figure 12 illustrates the fundamental architecture of the Smart NIC, incorporating P4-enabled functions. The MAC component manages the processing of Layer 2 Ethernet frames, while the P4 ingress and egress functional blocks handle incoming and outgoing packets. The PCIe/DMA functions oversee processing and transmission to and from the server. The P4 data plane functionality is structured into discrete blocks, including the parser, match and action, deparser, and deMUX, as depicted in Figure 12.

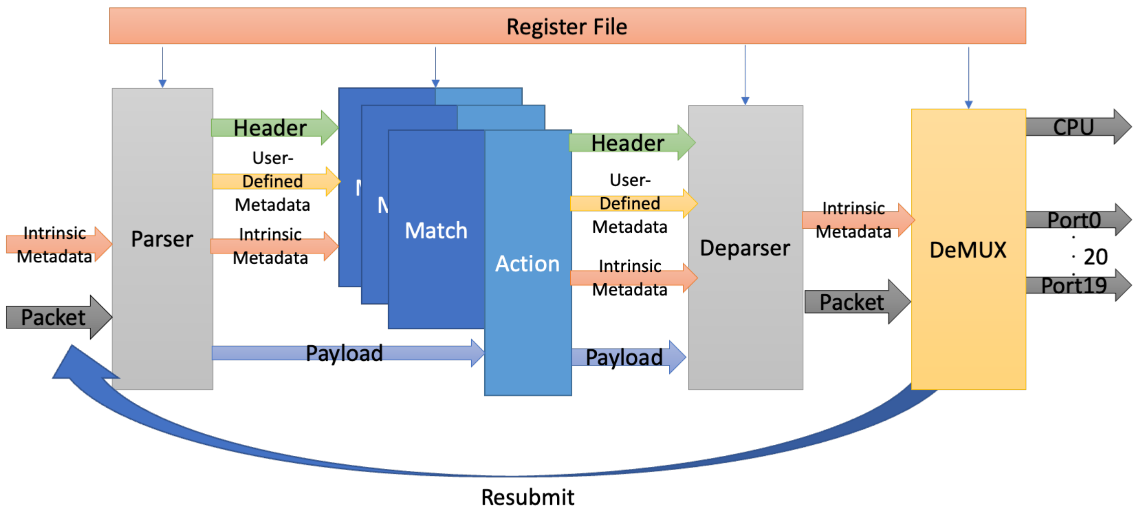


Figure 12:FPGA-based Smart NIC P4-enabled Dataplane architecture

The parser block of the P4 data plane is responsible for extracting packet headers from the incoming packet and parsing them into fields that can be matched against the rules defined in the match-action block. The parser block performs its operations in two phases. In the first phase, it reads the packet headers from the input buffer and converts them into a byte stream. In the second phase, it parses the byte stream into fields that can be matched against the rules defined in the match-action block. To accomplish this, the parser block uses a set of parsing rules that specify how to extract the header fields. These parsing rules define the structure of the packet headers and how they should be parsed. The parser block applies these rules to the incoming packet to extract the relevant fields and then stores these fields in a metadata register.

The metadata register is a temporary storage location that holds the extracted header fields until they can be processed by the match-action block. The metadata register contains information about the packet, such as the source MAC address, destination MAC address, IP address, and TCP/UDP port number. The parser block can be designed to support various packet formats and protocols. It can extract header fields from including Ethernet, IP, TCP/UDP, VxLAN, VLAN, and MPLS (up to 3 nested).

Furthermore, the parser block can be designed to operate in two modes: full parser mode and bit-based parser mode. In the full parser mode, the parser block parses the entire packet header and extracts all the fields that are specified in the parsing rules. In contrast, the bit-based parser mode only extracts the fields that are specified by the P4 agent. In this mode, the P4 agent specifies which fields should be extracted and how they should be parsed. The bit-based parser mode provides a more flexible approach to packet parsing and allows the Smart NIC to support new protocols without requiring any changes to the hardware design.

The match-action block of the P4 data plane allows for flexible and programmable packet processing. The match-action block works by comparing the fields extracted by the parser against a set of rules defined in the match table. Each rule in the match table specifies a set of conditions that must be met for the action to be taken. The conditions can include matching on the values of one or more header fields, as well as on metadata values. Once the match is made, the match-action block applies the corresponding action to the packet. The actions can be used to modify the packet fields, add or remove headers, forward the packet to a different port, or drop the packet, among other functions. The actions can also modify the metadata values, allowing for more complex processing to be performed in subsequent stages of the data plane.

One important aspect of the match-action block is the ability to handle multiple matches [71] for a single packet. This is achieved using priority levels in the match table, which allow for multiple rules to be evaluated in order of priority. This allows for more complex policies to be implemented, such as traffic shaping or load balancing. The match-action block enables flexible and programmable packet processing based on a set of rules. By allowing for the modification of packet fields and metadata, the match-action block enables the implementation of a wide range of networking policies and functions.

In addition, the matching block described employs a Ternary Content-Addressable Memory (TCAM) [72] to perform fast searching and matching of the searching key and the mask key, resulting in a hit/miss indication of the matching in a specific place. The designed FPGA-based matching functional block can handle up to 660 mask keys. The block can support match state of '1', '0', and 'X'. Furthermore, if a use case requires priority or more than 32 output requirements, the matching block can be cascaded. The implemented matching block utilizing TCAM technology provides efficient and scalable solutions for searching and matching P4 language's converted binary mask file with a vast number of possible matching cases.

The deparser block in a P4-enabled SmartNIC is responsible for assembling the modified packet fields into a new packet format that can be sent out on the network. The deparser uses a set of deparser rules to determine how to assemble the packet fields, and then it sends the new packet out through the appropriate port. The deparser rules specify how the fields extracted from the packet header by the parser block are to be arranged in the outgoing packet. This involves determining the size and position of each field, as well as the order in which they should appear. Once the deparser has assembled the new packet, it hands it off to the output port for transmission. In some cases, the deparser may need to perform additional processing on the packet before it can be sent out on the network. For example, it may need to add a checksum or perform some other kind of packet verification.

Metadata block provides a flexible and programmable way to store and share packet information between different blocks of the P4 pipeline. The metadata can be used to store packet headers, packet state, and other information that needs to be shared between the different blocks of the P4 pipeline. In addition, metadata can also be used to carry additional information such as statistics, timestamps, and flow state. This information can be used to enable advanced network functions such as network telemetry, load balancing, and flow monitoring. Metadata can be accessed and modified by the P4 program using metadata operations, which include reading and writing metadata registers, copying metadata between registers, and performing arithmetic operations on metadata values. The P4 program can also define its own metadata fields, which can be used to store custom packet information that is specific to the application.

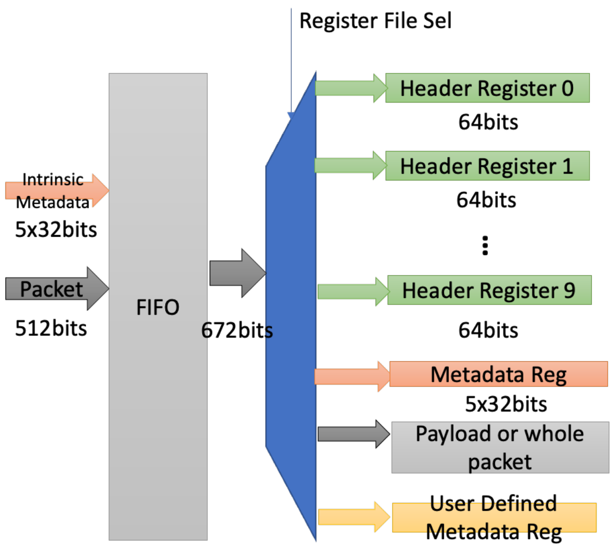


Figure 13:FPGA-based Smart NIC Hearder block

As shown in Figure 13, the Header Register 0-9: 64bits, can be any bits of the Header. The maximum header bits are 640bits. Header Registers/Metadata need to be synchronized with Payload. User Defined metadata register need to include the information of which bits of header are extracted.

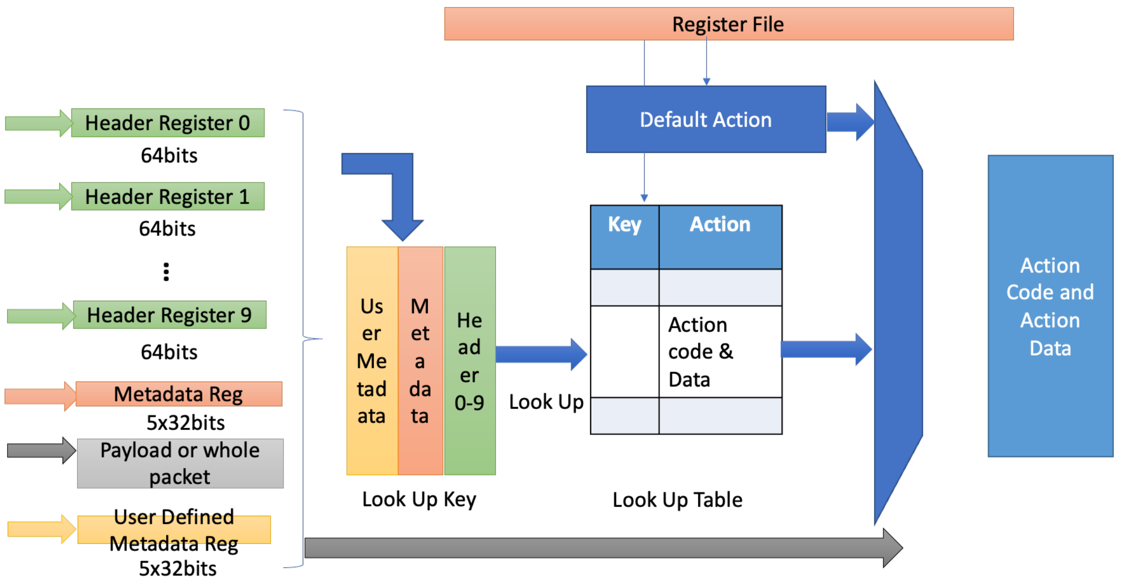


Figure 14:FPGA-based Smart NIC Look Up Table Block

As shown in the Figure 14, the Header Registers and Metadata Registers are set as the Look Up Key that can go through the LUT to the match and action block for the looking up and action.

Packet processing followed a pipelined approach, beginning with packet parsing. During parsing, metadata was extracted and matched against mask and matching tables. Processed packets then proceeded in accordance with the defined action rules, resulting in the creation of new packets for output.

The parser functional block was thoughtfully implemented with two distinct modes: the bit-based mode and the parser mode. On one hand, in the parser mode, the parser could parse the headers to multiple protocols, including Ethernet, IP, TCP/UDP, VxLAN, VLAN, and MPLS (up to 3 nested). On the other hand, the bit-based mode allowed the Smart NIC to receive instructions from the P4 agent bit by bit, specifying the instructions of parsing procession. This mode introduced a remarkable level of protocol independence.

However, for the support of other protocols such as Paxos and Market Data Protocol, the responsibility for protocol extraction fell upon the P4 agent. It was necessary for the P4 agent to manage the extraction process due to hardware limitations.

It's worth noting that the Smart NIC solution utilized the Xilinx hardcore 100G/25G Ethernet MAC, which primarily supported Ethernet-based protocols, owing to hardware constraints.

In summary, the system design harmoniously blends software and FPGA-based elements to attain high-performance networking capabilities. The software component facilitates seamless application support by leveraging open networking projects and established protocols. Conversely, the FPGA-based data plane segment shoulders the bulk of the low-latency and offload responsibilities, ensuring efficient and robust network performance.

3.3.3 FPGA-based Implementation Beyond P4

FPGA is a versatile and powerful tool that is not limited to network processing. In fact, current applications such as OvS offload, 5G UPC offload, machine learning, edge computing, and cryptos have all successfully used FPGA technology [55]. This suggests that FPGAs could be used as the basis for an application-driven server edge processor.

The ability to reconfigure an FPGA is a significant advantage that sets it apart from traditional processors. FPGAs can be programmed to perform a wide range of functions, and their flexibility allows them to be easily adapted to different applications without the need to physically change the hardware. With the ability to fully reconfigure an FPGA, designers can create entirely new functions that were not previously possible. This allows for rapid prototyping and experimentation, as well as the ability to adapt to changing application requirements. For example, if a particular application requires a different algorithm, the designer can simply create a new bit file and download it to the FPGA.

Partial reconfiguration technology elevates flexibility to an advanced level. It enables designers to load partial bit files into an FPGA while the remaining logic remains operational. This dynamic capability allows the FPGA's hardware functionality to be modified on the fly, without disrupting ongoing processes. The capacity to adapt the FPGA's hardware functionality in real-time yields substantial advantages across various applications. For instance, in a network processing scenario, the FPGA can be reconfigured to handle varying types of network traffic based on specific application needs. Similarly, in a machine learning application, the FPGA can be adjusted to optimize its hardware for particular neural network types.

An illustration of partial reconfiguration's utility is its application in crafting a P4 ACTION functional block. This block can be intentionally designed to support partial reconfiguration, thereby enabling the addition of supplementary sets of actions, processing routines, or algorithms to the system as required. Crucially, this enhancement can be accomplished without interrupting the ongoing service, which holds particular significance in situations where downtime is not permissible.

One of the paramount merits of partial reconfiguration is its capacity to empower FPGA designers to optimize resource utilization effectively. By segmenting the FPGA's functionality into smaller, modular components, designers can allocate resources with greater efficiency, culminating in more cost-effective and easily manageable designs. Consequently, the P4 data plane within the Smart NIC serves as a driving force that is application-centric in the Data Center Network (DCN).However, this approach relies on the Xilinx partial reconfiguration technology, the flexible module need to be drawn as a PBlock in the FPGA implementation. In practice, it is difficult to meet the timing especially when taken more than 50% FPGA resources or the design clock is more than 250MHz. Then this method might be a good reasearch outcome, but can't be a very practical product.

Despite these limitations, the use of partial reconfiguration technology remains an important area of research and development in the field of FPGA technology. As advancements are made in this area, it is possible that future designs may overcome some of the limitations associated with this approach.

By implementing the P4-enabled dataplane with the partial reconfiguration technology, I found it was hard to design and implement a general P4-enabled dataplane suitable for most applications. For specific application, such as OvS offload, some application specific twists need to be carried out.

3.3.4 FPGA-based Smart NIC offload engine

The trend towards more intelligent and efficient NICs has gained significant momentum, as organizations look to maximize their network performance and minimize latency. One of the key features that make a NIC intelligent and smart is the offload engine. This offload engine allows the NIC to offload certain tasks that would traditionally be performed by the CPU, such as computing the TCP/IP checksum, and instead handle them directly. This results in significant improvements in network throughput and reduced CPU utilization.

In the context of cloud services, the offload engine is even more crucial, as these services require a high level of agility and scalability. To support the technology and platforms that have already been employed in the DCN, such as SDN, DPDK, OpenStack, and others, the offload engine must be designed to be compatible with these technologies. Therefore, the offload engine that has been considered in the PhD work primarily includes TCP/IP checksum offload[66] and Open vSwitch offload[67]. These offload capabilities allow for faster and more efficient communication between devices on a network, while also ensuring that the NIC can handle the demands of cloud services and the technologies that support them.

3.3.4.1 TCP/IP offload

From Chapter 1, the primary reason for offloading TCP/IP processing to a NIC is to reduce the CPU burden on iterated packet processing. When packets go through the CPU-based TCP/IP stack, it consumes a significant amount of CPU resources, which can affect overall system performance. However, these tasks can be processed efficiently in parallel processing hardware. By offloading the TCP/IP stack[66] to the NIC, the NIC takes over the processing of the entire TCP/IP stack on the hardware level. This means that the NIC handles all packet formation, checksum calculations, and other tasks, while the CPU exchanges blocks of data with the NIC.

This method of offloading TCP/IP processing [73] has been proven to be an efficient and effective way of processing simple network protocols such as IP and ICMP. However, due to the complexity of the TCP/IP protocol, it is challenging to implement the entire protocol in FPGA-based RTL logic. The typical way of implementing TCP/IP protocol in a processor in the FPGA can reduce the CPU burden but significantly increases the latency. Therefore, offloading TCP/IP processing to the NIC can improve network performance without sacrificing latency. This approach can be especially beneficial in high-performance computing environments where low latency and high throughput are critical for efficient operation.

Since the TCP/IP stack can be very complexity based on the use case, to reduce the resource usage and balance the CPU and NIC workload, there is another TCP/IP offload method, called TCP/IP checksum offload. As previously described, this is a method of offloading the TCP/IP/UDP checksums calculation and verification in the NIC, which allows the traffic to be parsed and calculated by the NIC rather than by the host CPU. The FPGA-based NIC process the bits/bytes in parallel, which means, theoretically, it can process thousands of bits in 1 clock cycle (normally in ns); however, for the CPU, depends on the CPU processing width (64bits currently), in each clock cycle, it can process limited bits, and need iterated work for processing the whole packet and calculate the checksum. This is why processing the checksum in the FPGA-based hardware instead of CPU can dramatically decrease the CPU workload.

In some use cases, the TCP/IP stack can be extremely complex, which can cause significant resource usage and imbalance in workload between the CPU and NIC. Thanks to the DPDK, which enables applications to bypass the kernel and interact directly with the NIC by moving the packet processing to user space. Thus, significantly reduces packet processing latency and improves overall network performance. DPDK eliminates the overhead associated with system calls, context switches, and network stacks that are typically involved in kernel-based networking.

Although in certain use cases, such as financial trading, TCP/IP stack offloading improved packet throughput and reduced processing latency. To address this, there is another TCP/IP offload method known as TCP/IP checksum offload. This approach involves offloading the TCP/IP/UDP checksum calculation and verification to the NIC, enabling traffic to be parsed and calculated by the NIC instead of the host CPU.

The FPGA-based NIC processes bits/bytes in parallel, which theoretically allows it to process thousands of bits in a single clock cycle (usually in nanoseconds). In contrast, the CPU's processing width (currently 64 bits) limits the number of bits it can process in each clock cycle, requiring it to perform iterated work to process the entire packet and calculate the checksum. Processing the checksum in FPGA-based hardware instead of the CPU can therefore dramatically reduce the CPU workload.

By offloading the TCP/IP checksum calculation and verification to the NIC, the CPU workload is reduced, enabling it to handle other tasks more efficiently. This method is particularly useful in high-performance computing environments, where low latency and high throughput are critical. In summary, TCP/IP checksum offload is an effective way to balance the workload between the CPU and NIC and reduce the overall resource usage.

For the TCP/IP segmentation offload, the host or CPU don’t need to process the packets, do the segmentation. Instead, the host will send the bulk of data (can be up to 64Kb) to the Smart NIC, which needs to chop the data to the supported length, copy the TCP/IP/UDP headers from the known packets, calculate the checksums and add the checksums to the proper location of the packets.

3.3.4.2 TCP Segmentation Offload

TSO is specifically designed for the Transmission Control Protocol (TCP), which is a connection-oriented protocol that provides reliable delivery of data over a network. TSO offloads the task of breaking large TCP data packets into smaller ones, adding TCP headers to each packet, and calculating the checksums for the headers and data to the Smart NIC. By offloading this task, TSO reduces CPU utilization and can improve network performance in high-bandwidth environments.

When a large amount of data is sent over a network using the TCP, the data is divided into smaller packets and sent over the network. This process of segmentation requires the operating system to break the data into smaller pieces, add TCP headers to each packet, and calculate the checksums for the headers and data. This process requires CPU resources, which can reduce network performance, particularly in high-bandwidth environments.

With TSO, the operating system sends a single large packet to the NIC with instructions on how to segment it. The NIC then performs the segmentation, adds the TCP headers, calculates the checksums, and sends the smaller packets over the network. This offloads the segmentation process from the CPU, reducing CPU utilization and improving network performance.

On the receiving side of a network connection, RSO can offload tasks such as checksum verification, packet reassembly, and buffer management from the host CPU to the NIC or other hardware. This can help to reduce CPU utilization, improve network performance, and enable the host CPU to handle other tasks.

In a typical RSO implementation, the NIC or other hardware is responsible for verifying the integrity of received packets by calculating and comparing checksums, reassembling packets that have been split into multiple segments, and managing buffer memory. This offloads these tasks from the host CPU, allowing it to focus on higher-level processing tasks.

The key benefit of RSO, which is similar to TSO, is that it can improve network performance by reducing the overhead associated with packet processing on the host CPU. By offloading these tasks to the NIC or other hardware, RSO can reduce latency and increase throughput, particularly in high-bandwidth environments.

Implementing TSO in a FPGA requires designing custom hardware circuits that can perform the required packet segmentation, header addition, and checksum calculation functions. The TSO is a mature network technology that has been used in the traditional NIC, the project focused more on the innovated Smart NIC data plane design and implementation, I used a FPGA IP to support the basic TSO/RSO functions

3.3.4.3 PCIe Block

The Smart NIC was connected to the server via a PCIe interface for communication with the CPU. Our main chip, the Xilinx Ultrascale+ FPGA, made use of the Xilinx Ultrascale+ hard IP directly. The Integrated Block for PCIe in UltraScale+ Devices served as a foundational serial interconnect solution, offering remarkable attributes such as high bandwidth, scalability, and reliability. This IP core was meticulously crafted for deployment with UltraScale+ devices, and Xilinx furnished the PCIe integrated block PCIE4 [68] as a key component within this architectural framework. The PCIE4 block, seamlessly integrated into UltraScale+ devices, was adept at supporting

PCIe IP, accommodating configurations ranging from 1-lane to 16-lane setups, and it was compatible with various speeds up to Gen3 (8 GT/s). Notably, this IP core adhered to the PCI Express Base Specification, revision 3.1.

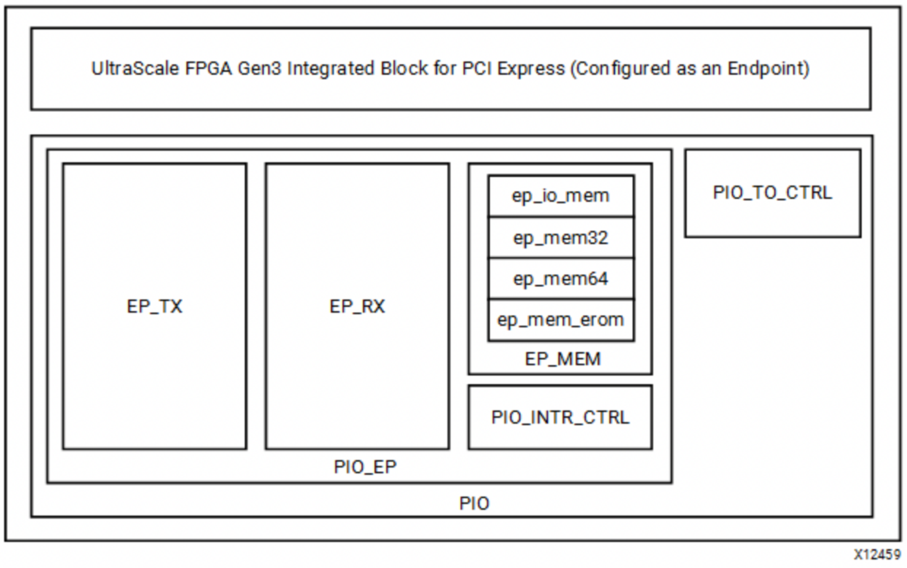


Figure 15:Xilinx PCIE4 Block Example Design [68]

The PCIE4 block stands as an incredibly versatile IP core, offering the ability to tailor it precisely to the specific needs of any given application. It boasts a range of advanced features, including dynamic link-width adjustment, hot-plug capability, advanced error reporting, and MSI-X interrupt support. The example design of the PCIE4 block comes complete with a PCIe Gen3 PHY, supporting both 8b/10b and 128b/130b encoding schemes, thereby enabling data transfer speeds of up to 8 GT/s. This PHY also incorporates vital features like adaptive equalization, receive equalization, and transmitter emphasis, which collectively serve to compensate for signal attenuation and distortion within high-speed serial channels.

In addition to the PCIe Gen3 PHY, the PCIE4 block houses a PCI Express controller responsible for managing the interface connecting the FPGA to the PCIe bus. This controller is equipped with advanced functionalities such as TLP (Transaction Layer Packet) reordering, packet buffering, and credit-based flow control.

For the Smart NIC development project, the PCIE4 block played a pivotal role by facilitating seamless interconnection with a DMA engine. This combination ensures robust and adaptable data transfers between the FPGA and the PCIe bus, contributing to the overall success of the project.

3.3.4.4 FPGA-based DMA design and implementation

DMA is a critical engine of transferring data between an FPGA target and the host computer. Because DMA can access the memory directly and does not need to involve the host processor, therefore it is widely used in today’s NICs. In the design, the DMA is designed and implemented following Intel’s 82599 poll mode DMA [74]. In this mode, the DMA controller does not rely on interrupts to transfer data between the NIC and system memory. Instead, it operates in a polling mode, continuously checking for data that needs to be transferred and handling these transfers without waiting for external interrupts. This approach can result in reduced latency and is often used in high-performance networking scenarios where low-latency data processing is critical.

Nevertheless, for the purpose of accommodating multiple VMs, attaining elevated bandwidth, and minimizing latency, there is a compelling requirement for an upgrade. This upgrade necessitates the transformation into a multi-core MDA (Multi-Core Direct Memory Access) system, which orchestrates data processing across multiple host CPU cores with enhanced efficiency. Consequently, data can be written to distinct host packet buffers, each designated for processing by dedicated CPU cores. This architectural enhancement empowers network security monitoring applications to harness parallel data processing capabilities, ultimately achieving superior levels of speed and operational efficiency.

In addition to drawing inspiration from the poll mode DMA, the design and implementation of the DMA also took cues from Remote Direct Memory Access (RDMA), a prominent feature in high-performance computing interconnects. RDMA introduces a queue-based communication model, enabling two or more computers to exchange data over a network through a meticulously organized queue-based system. These queues offer extensive configurability, with each one tailored to specific interface types and capable of functioning in various modes. Depending on how the DMA descriptors are loaded for a particular queue, messages can be added to or extracted from the queue, delivering a versatile and low-overhead solution for setup and ongoing updates.

A significant advantage of the queue-based design and implementation lies in its adaptability to multiple PCIe Physical Functions (PFs) and Virtual Functions (VFs). This flexibility allows a single DMA core and PCI Express interface to serve a wide array of multifunctional and virtualized application spaces.

The central mechanism employed by the DMA engine for data transfer hinges on the execution of instructions (descriptors) furnished by the host operating system. Utilizing these descriptors, the DMA can facilitate data movement in both the Host to Card (H2C) and Card to Host (C2H) directions. These data transfers occur through the AXI4-Stream interface, ensuring efficient and seamless communication between the host and the card.

DMA is a critical component of data transfer between an FPGA target and a host computer. In addition to take the Intel 82599 poll mode DMA as an example, the design and implementation of DMA also incorporate features from Remote Direct Memory Access (RDMA), which is commonly used in high-performance computing interconnects. RDMA uses a queue-based communication model, which allows multiple computers to communicate over a network using a queue-based system. Each queue can be individually configured by interface type and operates in different modes. DMA descriptors are loaded into a single queue to allow messages to be added or removed from the queue.

An inherent strength of the queue-based design and implementation lies in its remarkable versatility, which extends to the allocation of queues across multiple PFs and VFs. This strategic adaptability empowers the use of a single DMA core and PCIe interface within a diverse spectrum of multifunctional and virtualized application domains. This innovative approach not only streamlines the setup process but also offers continuous update functionality, serving as an efficient mechanism for data transfer while keeping latency to a minimum.

This architectural flexibility opens the door to significant advantages across various applications. By allowing a single DMA core and PCIe interface to cater to a wide range of multifunctional and virtualized scenarios, it optimizes resource utilization and enhances the overall efficiency of data transfer processes. Furthermore, the reduction in setup overhead and the seamless adaptability of queue assignments contribute to a responsive and high-performance data transfer environment, aligning perfectly with the demands of modern, dynamic computing systems.

The primary mechanism for transferring data using DMA is by operating on instructions (descriptors) provided by the host operating system. The descriptors provide information about the data to be transferred, including its source and destination addresses, length, and other relevant details. The DMA engine can move data in both directions, from Host to Card and from Card to Host. The interface used for DMA traffic is the AXI4-Stream interface, which is a high-speed, point-to-point interface designed for streaming data.

By utilizing features from both the Intel 82599 poll mode DMA and RDMA, the DMA design and implementation can support multi-VMs, achieve higher bandwidth, and reduce latency. The use of queue-based communication and descriptors allows for efficient data transfer, while the AXI4-Stream interface provides a high-speed, point-to-point interface for streaming data.

3.3.4.5 Queue-based DMA resource utilization optimization

As described above, the existing high-speed smart NIC's interaction with host data was achieved through the PCIe/DMA interface, which supports 1-4 ports with each port supporting multiple queues. However, when these multi-ports functionalities were implemented in the chip or the FPGA, multiple DMA channels should be implemented to work with multiple channels of multiple ports, which would increase the logical complexity and cost of the chip or the FPGA. Additionally, achieving multi-queue forwarding in high-speed smart NICs would also require multiple DMA channels and logical resources, thus further increased the complexity and cost of the chips/FPGAs. Furthermore, implementing the correspondent multiple queue functionality at the software level would require the software to parse and forward packets, which would consume significant amount of CPU resources.

To address these challenges, the proposed DMA optimization was to use a single channel in the Smart NIC to implement multiple ports and multiple queues, thus saving logic resources and reducing costs of the chip/FPGA.

Once a packet entered the Smart NIC, it was parsed, matched by 5-Tuples and other rules, and then assigned to the corresponding queue. The port and queue number were added at the end of the network packet, after that, the packet was transmitted to the software layer through PCIe/DMA. To avoid accidental duplication of port and queue number and data packet, a metadata format for the port and queue number was defined. The checksum is then calculated to ensure the data integrity.

The metadata format of the port and queue number were defined as follows in Table 2.

[Port ID (8 bits)][Queue ID (16 bits)][Checksum digit (8 bits)]

The check code is calculated by adding the port ID and queue ID and performing a bitwise XOR operation with a predefined constant.

Overall, this innovative approach allowed for efficient data distributing and forwarding in high-speed smart NICs without requiring multiple DMA channels or consuming significant CPU resources. It saved logical resources, reduced cost, and improved the overall performance of the system. The defined metadata format also helped to ensure the integrity of packet information and prevent duplication.

Table 2:Metadata Format								
Byte	0	1	2	3	4	5	6	7
Content（Hex）	aa	bb	cc	Port Number	Queue Number		Random Number	
	Checksum							

The system prepares 256 prime numbers, numbered 0-255. By using the identifier, port number, queue number, 8 bytes of the random number, the system calculated the value in each byte. Take the prime number of the corresponding content, for example, the first byte aa in Table 2's decimal is 170. Then take the corresponding 170th prime number, which was 4bc5d4f3, shift the resulted prime numbers to the left, add the operation by bits:

$$R = (R \ll 1) + V \tag{1}$$

The initial value of R was the first prime number, the initial value of V was the second prime number, and the value of V became the next prime number after each operation. With 8 bytes into account, totally 7 operations were needed.

Two examples of the calculations of the checksum were demonstrated below.

Example 1:

In case: Port number is 01, queue number is 0002, random number is 0305

The first 8 bytes of the added packet are $aabbcc0100020305$

The 8 prime numbers V (in Hex) are

aa (170) corresponds to the prime number V0: 4bc5d4f3

bb (187) corresponds to the prime number V1: 42dff19f

cc (204) corresponds to the prime number V2: 1ee35bb

01 (1) corresponds to the prime number V3: 3a907251

00 (0) corresponds to the prime number V4: 1fb0dfc9

02 (51) corresponds to the prime number V5: 7b720269

03 (03) corresponds to prime number V6: 6850364b

05 (05) corresponds to prime number V7: 02b3b673

According to the Equation 1,

R0=4bc5d4f3

R1 = (R0 << 1) + V1 = (4bc5d4f3<< 1) + 42dff19f = da6b9b85

R2 = (R1 << 1) + V2 = (da6b9b85<< 1) + 1ee35bb = d3c36cc5

R3 = (R2 << 1) + V3 = (d3c36cc5<<1) + 3a907251= e2174bdb

R4 = (R3 << 1) + V4= (e2174bdb << 1) + 1fb0dfc9=e3df777f

R5 = (R4 << 1) + V5= (e3df777f << 1) + 7b720269=4330f167

R6 = (R5 << 1) + V6= (4330f167<< 1) + 6850364b =eeb21919

R7 = (R6 << 1) + V7 = (eeb21919<<1) + 02b3b673=e017e8a5

The checksum is e017e8a5

The metadata is aabbcc0100020305e017e8a5

Example 2:

In case: Port number 02, queue number 0008, random number 0305

The first 16 bytes of the added packet are $aabbcc0200080305$

The 8 prime numbers V (in Hex) are

aa (170) corresponds to the prime number V0: 4bc5d4f3

bb (187) corresponds to the prime number V1: 42dff19f

cc (204) corresponds to the prime number V2: 1eec35bb

02 (2) corresponds to the prime number V3: 56cb4033

00 (0) corresponds to the prime number V4: 1fb0dfc9

08 (8) corresponds to the prime number V5: 45039065

03 (3) corresponds to the prime number V6: 6850364b

05 (5) corresponds to prime number V7: 02b3b673

According to the Equation 1,

R0=4bc5d4f3

R1 = (R0 << 1) + V1 = (4bc5d4f3<< 1) + 42dff19f = da6b9b85

R2 = (R1 << 1) + V2 = (da6b9b85<< 1) + 1eec35bb = d3c36cc5

R3 = (R2 << 1) + V3 = (d3c36cc5<<1) + 56cb4033= fe5219bd

R4 = (R3 << 1) + V4 = (fe5219bd << 1) + 1fb0dfc9=1c551343

R5 = (R4 << 1) + V5 = (1c551343<< 1) + 45039065=7dadb6eb

R6 = (R5 << 1) + V6 = (7dadb6eb << 1) + 6850364b =63aba421

R7 = (R6 << 1) + V7 = (63aba421<<1) + 02b3b673=ca0afeb5

The checksum is CA0AFEB5

The metadata is aabbcc0200080305CA0afeb5

When the driver recognized the identifier (aabbcc), it used the same identifier + port number + queue number + random number to calculate the checksum. After that, it compared the calculated value with the checksum. If the results were identical, then the data was considered the correct data which could be used as the port number and queue data. We could extract the port number and queue number from the data.

According to the method described above, the corresponding PCIe/DMA logic was implemented with Xilinx VU3P FPGA, using the logical resource occupation of single-channel DMA in the thesis and the DMA logical resource occupation of the traditional multi-channel multi-queue mode as shown in Table 3.

Table 3:Optimised Single Channel DMA Resource Utilization vs Normal DMA resource utilization

	CLB LUTs	CLB registers	Block RAM
The Optimised DMA resource utilization	43348	44373	73
Normal DMA resource utilization	76648	68175	159

From Table 3, a comparison of the two methods shows that logical resources CLB LUTs save 43%, CLB registers save 34.9%, and Block RAM save 41.5%.

3.3.4.6 DMA throughput performance Optimization

With the resource utilization optimized Queue-based DMA implemented, for 100GbE transmission, the PCIe Gen3 x 16 had the limitations on the small size packets, such as 64B to 256B. The PCIe throughput was limited by the transmission and reception of the size of the frames. The feature of the Smart NIC or NIC was designed to handle the network packets one by one, even the small one. Therefore, the network performance was limited by the PCIe throughput.

To improve the throughput of the Ethernet frames, some work needed to be done on the PCIe side of the Smart NIC. The optimization was based on the network packet transmission scheme on the PCIe end. We designed an algorithm for optimizing the network packet transmission by accumulating the packets together. In the meantime, to avoid the high latency that couldn’t meet the users’ requirements of the networks, the algorithm was based on the bandwidth and latency requirements of the user. The length of the packets to be transmitted could be calculated under the premise of the required latency and throughput. So that for the same type of network packet, the algorithm provided was able to calculate the number of bytes transmitted by large packets which could meet the required latency and throughput. Therefore, the PCIe bandwidth was used effectively to improve the utilization of high-speed Smart NIC on the PCIe side.

By providing a mask-based match lookup module in the smart NIC, users could configure the requirements of network packets, such as some packets with specific throughput and latency requirements, and some packets with other specific throughput and latency requirements. In this way, through the controller of the smart NIC, the requirements were configured, and the data packets entering the Smart NIC could be classified. After that, with the specific throughput and latency considered, the number of bytes of the packet that needed to be accumulated for a bigger packet was calculated according to Equation 2 and 3 as below. Then, when the packets entered the Smart NIC, the number of bytes of the required packets were accumulated, and when the calculated number of bytes were reached, it was sent out as a whole packet.

$$\mathrm{PS} = (8\mathrm{R} * \mathrm{BW} * (\mathrm{OD} + \mathrm{D} - \mathrm{BD}))/(\mathrm{R} - \mathrm{BW}) \tag{2}$$

$$\mathrm{OD} = 32 * 8/\mathrm{R} \tag{3}$$

Table 4: The symbol meanings in Equation 2,3

Symbol	Description
R	Port bandwidth（bit/s）
BW	The bandwidth required by the application（bit/s）
PS	The payload after getting packets together（byte）
OD	The latency of overhead after assembling packets together (s）
D	The latency required by the application（s）
BD	The latency of the whole smart NIC before getting packets together. （s）

With the Equation 2 and 3, the redundant byte length was also recorded in the FPGA to tell the driver the actual bytes of the packets after multiple data packets of the same application were assembled. To simplify the calculation and complexity of splicing, the redundant byte length after splicing was set to 32.

To identify the accumulated packets, you need to mark them in the DMA description table, and the packets spelled from the network card to the server need to be unpacked on the server. The packets spelled from the server to the NIC need to be unpacked on the NIC side.

Through the above-described method, the throughput and latency requirements could be achieved, the number of transmissions from the Smart NIC on the PCIe end could be reduced, and different throughput rates and latency services can be provided according to different applications.

The test results of the 100GbE Smart NIC were shown in Table 5 and Table 6.

Table 5：100GbE Smart NIC measured performance（Throughput）

Ethernet Frame Size	Without accumulating frames	Accumulating frames to 1024B，adaptable throughput
64B	30.2Gbps	30.2Gbps to75Gbps
128B	34.6Gbps	34.6Gbps to 81.2Gbps
256B	43.1Gbps	43.1Gbps to 91.3Gbps

As shown in Table 6, when the received network packets are packets of 256 bytes or less, this method can effectively improve the throughput rate and effectively adapt to user needs.

Table 6: 100GbE Smart NIC measured performance（Latency）

Ethernet Frame Size	Without accumulating frames, Latency	Accumulating frames to 1024B， adaptable Latency
64B	1.2µs	The value can be any value >=1.2 µs

As can be seen from Table 6, the latency of network packets may fluctuate according to the network environment and internal storage size, and this method can effectively adapt to user needs.

3.4 FPGA-based Smart NIC Evaluation and Results

As described in previous sections, a P4-enabled Smart NIC was designed and implemented on a Xilinx Ultrascale+ VU3P device. For the outcome of this Smart NIC, a test bed was setup as shown in Figure 16 Then the resource utilization and performance were measured.

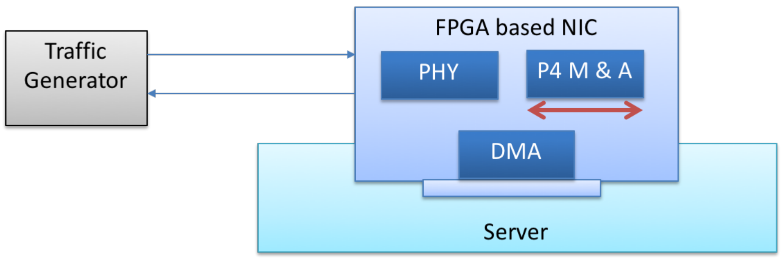


Figure 16:FPGA-based Smart NIC evaluation testbed setup

The P4 data plane block was not restricted to the vendor, which means that the design was not limited to a specific hardware vendor. The testbed was set up with an Anritsu MT1100 generating 100Gbps Ethernet traffic flow, which was fed to the FPGA-based SmartNIC. The data was then moved from the FPGA to the server RAM using DMA, and vice versa. The traffic flow was sent back to the traffic generator for measuring data loss and latency.

The experiments and measurements in this chapter focused on the P4 data plane functional blocks only, which were responsible for packet forwarding and processing. We targeted three metrics: resource utilization, latency, and throughput. Resource utilization refered to the amount of resources consumed by the P4 data plane blocks on the FPGA, including logic cells, memory, and DSP blocks. Latency is the time it takes for a packet to travel through the Smart NIC, while throughput refers to the amount of data that can be processed by the Smart NIC per unit time.

For proving the FPGA-based Smart NIC architecture design and implementation, in this chapter, our experiments and measurements focused on the P4-based data plane functional blocks, and targeting on three metrics: resource utilization, latency, and throughput.

3.4.1 Resource Utilization

The parse-match-action process was the core concept in P4 programming that was used to define the packet processing pipeline for networking devices. The process involved three steps: parsing, matching, and action. In the parsing stage, incoming packets were inspected and broken down into their constituent fields. These fields were then matched against pre-defined rules in the matching stage. Finally, in the action stage, a set of actions was performed on the packet based on the results of the matching process.

Implementing the parse-match-action process in hardware could be challenging, as it involved balancing the timing closure and combinational logic of the design. Timing closure refered to the process of ensuring that all signals in the design met their timing requirements, such as setup and hold times. This was crucial to ensure that the hardware design functions correctly and reliably. However, meeting timing constraints could be difficult, especially in high-speed networking applications.

Combinational logic refers to the logical operations performed on signals in the hardware design. In the context of P4 programming, the combinational logic was used to implement the matching and action stages of the parse-match-action process. Designing the combinational logic to be efficient and performant was critical to achieving high throughput in the hardware implementation.

Balancing timing closure and combinational logic was a common challenge in hardware design, and it was especially relevant in P4-enabled Smart NICs, where high-performance and low-latency packet processing was critical. The authors of the paper focused on optimizing the design of the parse-match-action process to achieve high performance and resource efficiency in the P4-enabled Smart NIC implementation.

Table 7: FPGA-based Smart NIC P4 Functional Blocks Resource Utilization

resource	utilization	available	utilization %
LUT	38855	394080	9.85
LUTRAM	4292	197280	2.17
FF	9560	788160	1.21
BRAM	120.5	720	16.7

In the implementation of the P4-enabled Smart NIC, I was able to achieve a clock frequency of 350MHz while minimizing the utilization of FPGA resources. This was an impressive achievement, as high clock frequencies are critical for achieving high-speed packet processing in networking devices. Table 7 shows the resource utilization of the FPGA for the P4-enabled Smart NIC implementation. The authors used various FPGA resources to implement the parse-match-action process, including BRAMs, FFs, and LUTs.

BRAMs, or block RAMs, were primarily used for the TCAM match and configure block. TCAMs are used for fast table lookups in networking devices. The BRAMs in the FPGA were used to implement the TCAMs and store the pre-defined rules used in the matching stage of the parse-match-action process.

Flip-flops, were used for the parser. The parser was responsible for breaking down the incoming packet into its constituent fields, and FFs were used to store the state information of the parser.

LUTs were used for the action block, which is responsible for performing a set of actions on the packet based on the results of the matching stage. The LUTs were used to implement the combinational logic required for the action block.

The implementation showed its possibility to achieve high performance and resource efficiency in P4-enabled Smart NICs by optimizing the implementation of the parse-match-action process and carefully utilizing FPGA resources such as BRAMs, FFs, and LUTs.

3.4.2 *P4 configuration from controller*

In addition to the hardware implementation of the P4-enabled Smart NIC, a software component to enable the translation of the P4 code into a format that can be understood by the FPGA was designed. This piece of the agent code and the open source P4C were combined and wrapped by using the open source P4C for FPGA-based Smart NIC.

The P4 binary file generated by the compiler was translated to the mask table, the match table, and the action table. These tables were sent to the FPGA-based Smart NIC for operating the P4 functional blocks.

The latency from sending the P4 binary file to the rising edge of the P4 updated ready signal of the P4 block in the FPGA was measured. The latency was found to be below 1 millisecond, which is an impressive result that demonstrates the network manager's ability to change the data plane behaviour in real-time.

This software component is essential for the operation of the P4-enabled Smart NIC as it enables the translation of P4 code into a format that can be understood by the FPGA. By developing their own P4 compiler agent, the authors were able to optimize the translation process for their specific implementation, which likely contributed to the low latency achieved.

3.4.3 *P4 functional block latency and throughput*

The experiments were conducted to measure the latency and throughput of the P4 block in the FPGA using a test bed setup shown in Figure 16. The action was set as 5 stages with the following operations: set Destination MAC address, push MPLS, pop VLAN, tag, and forward to a port.

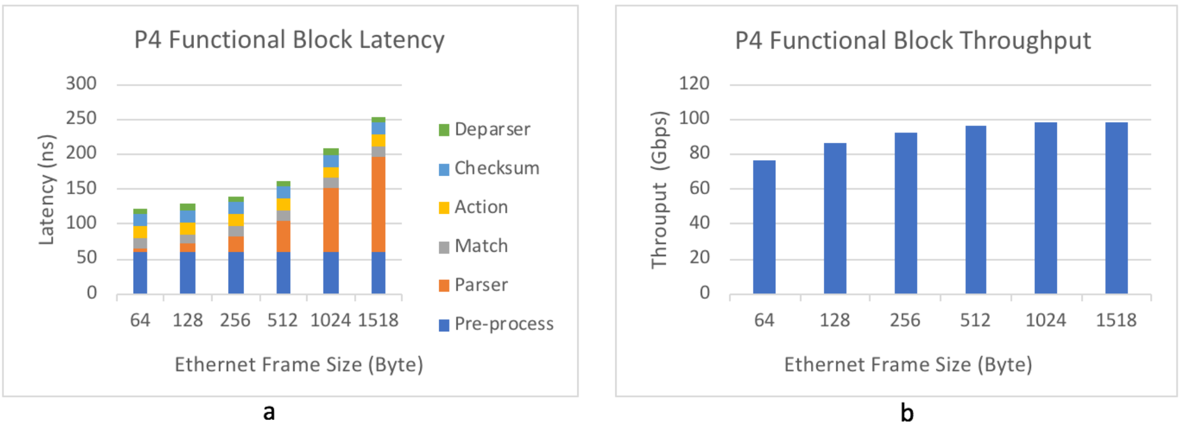


Figure 17:The measurement results of P4 functional: (a) Latency; (b) Throughput

The outcomes of our experimental investigations have been thoughtfully presented in Figure 17, providing a comprehensive insight into the performance characteristics of the system under scrutiny. In particular, our examination aimed to shed light on the intricate intricacies of latency and throughput, which are crucial parameters for understanding the system's overall efficiency.

As showcased in Figure 17 (a), the latency analysis offered a detailed breakdown of latency across various functional blocks. It was discerned that the latency of the parser was predominantly influenced by the length of the Ethernet frames. In contrast, the remaining functional blocks exhibited a relatively stable and consistent latency, regardless of the specific input conditions. This finding illuminates the significance of Ethernet frame length as a critical determinant in the overall latency experienced by the system.

Turning our attention to Figure 17 (b), the throughput analysis revealed that several factors played a pivotal role in influencing the system's performance. Notably, the size of the Ethernet frames emerged as a prominent factor impacting throughput. Moreover, the efficiency of the PCIe/DMA (Peripheral Component Interconnect Express/Direct Memory Access) subsystem significantly affected the throughput of the system. It is imperative to highlight that when the action block operated as a push/add function, there was a discernible reduction in throughput, resulting in a throughput level that was 98% of the baseline measured result.

These results are important for understanding the performance of the P4-enabled SmartNIC and can help guide further optimization of the system. And it helps us to be able to identify the factors that most significantly impact the latency and throughput of the system and improve the system performance in the following chapters for the 5G and cloud use cases.

Chapter 4. P4-ENABLED SMART NIC FOR ACCELERATING END-TO-END INTER-SERVER COMMUNICATION

P4 has become increasingly popular in the networking industry due to its unique ability to enable network operators and developers to customize network behaviour and performance to a degree that was not possible with traditional networking technologies. With P4, network operators can define their own packet processing rules and tailor the behaviour of their networks to their specific needs. This level of flexibility and control had made P4 an attractive option for developers who seek to optimize network performance and improve the efficiency of network resources. However, it was important to keep in mind that P4 was not a silver bullet that would solve all network-related problems. Actually, like any technologies, it had its limitations, and proper implementation and usage were crucial to achieve the desired results. Despite this, when effectively utilized, P4 had the potential to enhance the performance and efficiency of networks, making them better equipped to handle service workloads, met the demands of modern applications, and enabled new and innovative use cases.

With a P4-enabled Smart NIC, network administrators could dynamically adjust network settings and protocols in real-time to respond to changing network conditions and traffic patterns, thus, powered end-to-end programmability in the network for inter-server communication.

In this chapter, the P4-enabled data plane use cases in 5G networks were explored, the focus was on the Smart NIC enabled end-to-end applications.

4.1 Introduction to the P4-enabled data plane technologies in 5G networks

The extensive deployment of 5G networks by leading global operators, coupled with advancements in autonomous vehicles and Internet of Things (IoT), has unveiled significant potential to revolutionize our modes of communication and interaction with technology. These developments had led network innovators to begin discussing what should the next generation of networks would look like, probably to bring significant improvements in terms of speed, latency, capacity, reliability, and flexibility, enabling new use cases and applications that were not possible with previous generations of networks. Some of the key technologies would probably include millimetre-wave communication, massive MIMO, network slicing, edge computing, and artificial intelligence, more of which are the KPIs of the 5G already.

4.1.1 5G UPF

5G UPF (User Plane Function) was a crucial component of 5G networks responsible for processing and forwarding user data packets. In other words, the 5G UPF receives data packets from the 5G Core Network and forwards them to their intended destination, such as a website or application. It played a critical role in ensuring the efficient and reliable transport of data traffic in 5G networks. The 5G UPF was designed to support the high-speed, low-latency, and highly scalable requirements of 5G networks. It was located at the edge of the network, closer to the end-user devices, and can be deployed in multiple locations depending on the network operator's requirements.

In addition to data forwarding, the 5G UPF could perform other functions, such as packet filtering and inspection, policy enforcement, and charging. These functions were critical for ensuring the smooth operation of the 5G network and providing a high-quality user experience.

In a typical 5G network architecture, the 5G UPF was located at the edge of the network, closer to the end-user devices. It was connected to the 5G Core Network (5GC) via the N6 interface and to the Radio Access Network (RAN) via the N3 interface.

When an end-user device sent data traffic over the 5G network, the traffic was initially received by the RAN, which forwarded it to the 5G UPF for further processing and forwarding. The 5G UPF then applied policy rules and filtered the traffic, before forwarding it to its intended destination, which could be a service or application on the internet or another network.

The 5G UPF played a critical role in ensuring the high-speed, low-latency, and reliable transport of data traffic in 5G networks. Its location at the edge of the network allowed it to provide low-latency data transport and support for a large number of connected devices, which were key requirements for many 5G use cases such as autonomous vehicles, smart factories, and virtual reality.

P4 allowed network administrators to define how packets are processed by network devices. By writing P4 code, administrators can customize the behavior of the UPF, which was the key component of 5G network architecture responsible for packet forwarding and routing functionality.

Using P4, administrators could define how the UPF handles various types of traffic, including data traffic, control traffic, and management traffic. For example, they could create rules that specify how the UPF should handle different types of traffic based on their QoS requirements. This allowed administrators to enforce network policies such as traffic management, packet filtering, and load balancing.

P4 could also be used to implement new network services and applications on the UPF. For example, administrators can write P4 code to enable new security services such as intrusion detection and prevention, or to create new network functions such as traffic shaping or packet compression.

4.1.2 Network Slicing in 5G networks

Transport network slicing was one of the critical features in 5G networks and beyond, as it allowed for the creation of virtualized technology frameworks that could tailor the network performance and functionality to meet the specific requirements of tenants such as mobile operators, DC applications, fintech, and more. By slicing the network across multiple layers, including optical, IP, and applications, network slicing allowed for the efficient allocation of network resources, resulting in better network performance and improved quality of service.

Network slicing enabled the forwarding of packets with the ability to dynamically demand the necessary bandwidth and latency. This ensures that the network be able to adapt to the changing demands of different applications and their associated KPIs, ensuring that they were able to perform at their best. With the ability to support different functional splits, network slicing was able to provide a high level of flexibility and scalability to 5G networks, allowing for the creation of tailored network solutions that can meet the unique requirements of different applications and tenants.

Network slicing in 5G networks was a powerful tool that allows network operators to create multiple virtualized networks on top of a shared physical infrastructure. This allowed different types of applications to coexist on the same network, while each application receives the specific network resources it needs to meet its requirements. With network slicing, the network could be divided into separate logical networks, each with its own network functions, control plane, and data plane.

In addition, with the optical enablers, optical network slicing could be used to cooperate with network slicing to separate optical channels on a shared physical infrastructure, allowing different applications to have their own dedicated optical channels. IP network slicing, on the other hand, allows network operators to create virtualized IP networks with their own dedicated network functions, such as routing, security, and QoS.

The dynamic allocation of network resources, such as bandwidth and latency, based on the requirements of specific applications was one of the key benefits of network slicing. With network slicing, resources could be allocated on demand, allowing the network to adapt to changing traffic patterns and ensure that each application receives the necessary resources to operate at peak efficiency. This was especially important in 5G networks, where the number of connected devices and the variety of applications was expected to increase significantly in the coming years.

Segment Routing (SR) emerged as a widely embraced technology in the realm of IP networks, employed to furnish virtualized Virtual Private Networks (VPNs). SR harnessed the power of source routing and represented an extension of Multi-Protocol Label Switching (MPLS). Its design originally revolved around the central controller, which was responsible for managing label assignments and their distribution.

Source routing referred to the process of specifying the complete path of a packet through a network by including the complete sequence of nodes that the packet should traverse in its header. This method allowed for greater flexibility and control over network traffic flows. SR leveraged this concept by using a centrally controlled label stack for packet forwarding.

In SR, a centralized controller assigned labels to traffic flows and distributes them to network devices, such as routers and switches. These labels were then used to direct traffic along a specific path through the network. SR had been shown to be effective in allowing traffic to be sent from servers and traverse the network core all the way to the destination.

By leveraging SR, network operators could create virtualized VPNs that were tailored to the specific needs of their customers. This enabled them to provide customized services with different QoS requirements, such as low latency or high bandwidth, while maintaining network efficiency and simplicity. Additionally, SR allowed for more efficient use of network resources, reducing the need for expensive dedicated hardware for each VPN.

Optical networking had become an increasingly popular choice for data center interconnectivity (DCI) due to its ability to provide high-speed, high-bandwidth connectivity over longer distances. Various vendors, including web scale internet companies like Facebook, had been developing compact optical transport systems for DCI purposes. These systems were available in compact sizes, with some being as small as pizza boxes, and offered adaptive multi-rate, multi-protocol ports ranging from 1Gbps to 100Gbps.

One of the key features of these optical transport systems was their ability to provide variable bandwidth allocation through a technology called Bandwidth Variable Transponder (BVT). This technology enabled bandwidth allocation towards the core of the network where coherent technologies can take the signal for much longer distances. Coherent technologies utilized a technique called quadrature amplitude modulation (QAM) to increase the amount of data that can be transmitted over long distances while maintaining signal integrity.

P4 could be useful for SR because it allowed for the customization and programming of packet forwarding behavior at the data plane level. With P4, it was possible to program the forwarding plane to implement specific features and functions of SR, such as label assignment and distribution.

Moreover, P4 could enable the integration of SR with other network functions and services by allowing for the creation of custom processing pipelines and the insertion of new functions at various points in the pipeline.

By using P4 to program network devices, it was feasible to create custom forwarding rules that support SR. For example, a P4 program could be used to implement a custom parser that extracted the SR segments from incoming packets, and then forwarded the packet based on the specified segments. P4 could also be used to implement other features of SR, such as label assignment and distribution. For example, a P4 program could be used to implement a central controller that assigns labels to packets and distributes them to network devices.

Furthermore, P4 could help improve the performance of segment routing by enabling network devices to process routing decisions at high speeds with low latency. This is important for real-time network traffic steering and optimization. And another important factor for 5G network was scalability. P4 could help improve the scalability of segment routing by enabling network operators to implement segment routing at different points in their network, such as at the edge, in the core, or in the cloud. This could help operators steer network traffic through complex network topologies and avoid network congestion.

4.1.3 Inband Network Telemetry

Inband Network Telemetry (INT) is a network monitoring technique that provided real-time visibility into the performance and behaviour of network devices and traffic flows. Unlike traditional monitoring techniques, which relied on out-of-band methods such as SNMP, INT embeds telemetry data directly into the data packets as they traversed the network.

The key idea behind INT was to use data packets as a carrier for telemetry information, rather than relying on separate monitoring traffic. This was achieved by inserting additional fields into the packet header to record information about the packet as it traverses the network. This telemetry information can include details such as packet latency, packet loss, queue depth, and other performance metrics.

By embedding telemetry data directly into the packet, INT provided a more granular and accurate view of network performance than traditional monitoring techniques. It allowed network administrators to identify and diagnose performance issues in real-time, and to optimize network resources and traffic flows accordingly.

INT is especially useful in modern network environments where applications and services were highly distributed and dynamic. It enabled administrators to monitor network traffic and performance at the micro-level, providing insights into individual packet flows and identifying issues that might otherwise go unnoticed.

A critical part for INT required end-to-end hardware support in network devices such as switches, routers, and network interface cards (NICs) to insert and extract telemetry data from packets. Therefore, the first step in implementing INT was to ensure that the network devices being used have the necessary hardware support for INT. P4 dataplane with the programmability and extensibility was a good candidate for implementing INT-enabled network devices.

Besides and programmability and extensibility, P4 could help improve the performance of INT by enabling network devices to process telemetry data at high speeds with low latency. This was important for real-time monitoring and analysis of network traffic. Furthermore, P4 could help improve the scalability of INT by enabling network operators to implement INT at different points in their network, such as at the edge, in the core, or in the cloud. This was able to help operators collect telemetry data from different parts of their network and analyze it in a centralized location. With the possibility of user-defined P4 dataplane, the innovation became possible with new telemetry data collection and analysis. This could help operators develop new insights into their network performance and create new services.

4.2 P4-enabled Smart NIC architecture optimization for 5G UPF

The P4 language enabled programming of the network data plane, providing greater flexibility to the user. Within the 3GPP 5G core network system architecture, the User Plane Function (UPF) is a critical component responsible for routing, forwarding, and other related functions for user plane packets. Compared to 4G, 5G required lower latency, larger bandwidth, and higher reliability in multi-user scenarios, making UPF, edge computing, and network slicing pivotal to its success. To meet the high bandwidth forwarding capacity demands of the core network, operators typically expand computing resources, share demand among multi-core processors, or utilize Smart NIC.

The 5G UPF user plane function, whether server- or software-based, relied on multi-core processors and consumes a significant amount of CPU resources. As a result, it might suffer from transmission and reception performance bottlenecks. One solution is to offload the 5G UPF user plane function to the Smart NIC. By doing so, the performance of 5G core network software could be accelerated, improving throughput, and industry-standard APIs interfaces could be used to maintain configuration standardization.

However, implementing the data plane of 5G UPF within the Smart NIC required the development of hardware logic, which presented a significant workload and might consume a large amount of logic resources of the Smart NIC. Despite these challenges, offloading the 5G UPF user plane function to a smart NIC had the potential to address the performance bottlenecks associated with server- and software-based implementations, providing operators with a more efficient and scalable solution for their 5G networks.

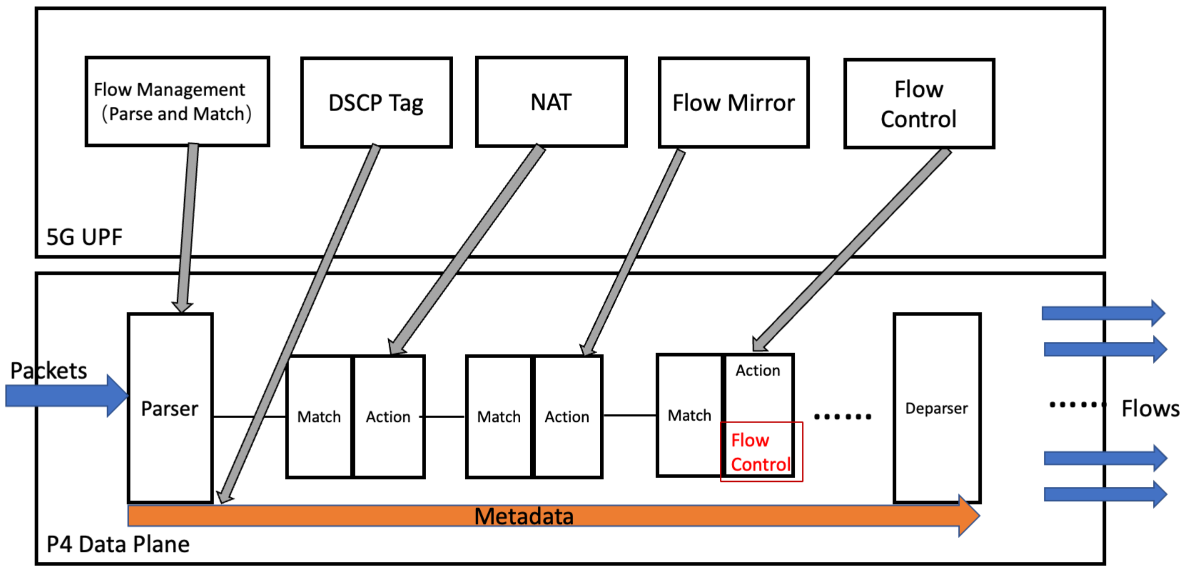


Figure 18:5G UPF matching to P4 data plane

As presented in the upper one of Figure 18, the P4 module, according to the standard of the P4 data plane and the requirements of the Smart NIC, the entire P4 programmable data plane consisted of a parsing module, a multi-level matching and action module, and a deparser module. Among them, the multi-level matching module realized multi-level search matching, and each level included exact matching based on hash operation and mask matching based on three-state content addressing memory.

The main functions of 5G UPF user plane could be decomposed into stream processing, including packet parser, stream matching, flow rate control, DSCP (Differentiated Services Code Point) different service tag codes, NAT network address translation and flow mirroring.

Based on Figure X, the 5G UPF user plane function couldn't be delivered directly to the P4-based data plane because the metadata of the P4 data plane in the smart NIC was required to add and support the on-flow data based on the 5G UPF user plane. The P4 data plane did not support flow rate control and cannot implement the necessary functions of the 5G UPF user plane.

Therefore, for supporting 5G UPF user plane function with existing Smart NIC P4 data plane, as shown in lower figure of Figure X, the packet flow proceeded following a matching process, during which the parameters were linked to the Metadata configurations in the P4 data plane. The variables and parameters of the 5G UPF user plane function was incorporated into the metadata of the P4 data plane. This integration allowed them to be seamlessly carried through the different stages of the pipeline, addressing the issue of attaching the DSCP service tag code to the packet flow.

To enable the integration of the 5G UPF function mapping module and the flow rate control action module, an expansion of the existing P4 action module was necessary to align with the P4 language requirements. This expansion involved incorporating the original meter module from the P4 action module and enhancing it to include cache and timing packet generation capabilities for supporting flow rate control. Simultaneously, the meter module should maintain its original functionality while adding a flow control flag within the Metadata. This flag instructed the meter module whether it should support flow rate control.

The configuration involved mapping and defining the pipeline matching and actions within the P4 data plane according to the 5G UPF user plane function, encompassing the flow table mapping and action execution subsequent to a successful flow table match.

Upon packet ingressed into the Smart NIC, it underwent parsing and subsequent processing in alignment with the specific action associated with its flow within the 5G UPF user plane function pipeline. This processing encompassed not only the seamless termination of the 5G UPF user plane function by the Smart NIC but also the handling and distribution of relevant messages.

In detail, the P4 input file was defined by the 5G UPF user plane function, and the action modules that need to be called include no_op, drop, modify_field, count, meter, clone_ingress_pkt_to_egress, etc. According to the functions to be implemented by the 5G UPF user plane, the Metadata setting of the P4 data plane was carried out first. For example, if the DSCP value of the packet matching destination address 192.168.0.1 is 1, set the DSCP of Metadata to 1, and the Metadata flows with the packet.

Subsequently, the 5G UPF user plane function was employed to perform matching and initial configuration within the P4 data plane pipeline. For instance, when the NAT function was linked to the Modify IP Address action in the P4 action module, incoming packets destined for the IP address 192.168.0.2 were altered to 192.168.1.2 as part of this process.

Then, in accordance with the 5G UPF user plane function, the P4 module underwent further matching and configuration. For instance, in the case of packet flow mirroring, the P4 action module was configured for mirroring. When a packet with a destination address of 192.168.0.3 was identified, it was duplicated to queues 2 and 3, effectively copying the packet with the destination address 192.168.0.3 to both queues 2 and 3.

After that, the 5G UPF user plane function is matched and the third level of the P4 module is set. For example, for the message flow rate control function, set the P4 action module to the new flow rate control module. If the flow rate of the packet matching destination address 192.168.0.4 was controlled at 2Gbps, the flow rate control flag bit was set to 1 in the metadata accompanying the packet flow of 192.168.0.4, and when the meter action of P4 was passed, because the flow rate control flag bit in the metadata was 1, the meter action of P4 was used as the flow control, and the flow control module was called to control the flow rate of 192.168.0.4 at 2Gbps.

Finally, if there were additional 5G UPF user plane functions to be matched, they were incorporated through further matching processes. However, if no additional functions need to be matched, the P4 configuration was considered complete, and incoming packets were then processed based on their respective configurations.

To achieve a three-level P4 data plane, each level of 1K matching was an example, and the resources occupied were shown in the Table 8. Wherein 5G UPF occupies logic resources could be completely released after using the mapping method in the present invention, so about 40% of the logic resources could be saved after mapping.

Table 8：Optimized Design and implementation for 5G UPF resource utilization

FPGA Logic Resources	P4 Dataplane	Optimized for 5G UPF	Resources Saving
LUT	157050	134232	46.08%
Flip Flop	38930	26380	40.50%
BRAM	372	302	44.81%

4.3 P4-enabled Smart NIC for accelerating 5G inter-DC communications

The SRv6 acceleration scenario was selected for P4-enabled Smart NIC use cases in 5G, which was previsouly published in our papers [80] and [81]. The setup architecture depicted in Figure 19 presents a unified fronthaul and backhaul network structure, where virtualization and data plane programmability play pivotal roles. The primary driver for enabling data plane programmability is the FPGA-based Smart NIC with P4 capabilities. Additionally, the Voyager BVT Transponder plays a significant role in facilitating programmability within the optical domain. Furthermore, the 5G edge node UPF has been equipped with an FPGA-based Smart NIC for enhanced functionality.

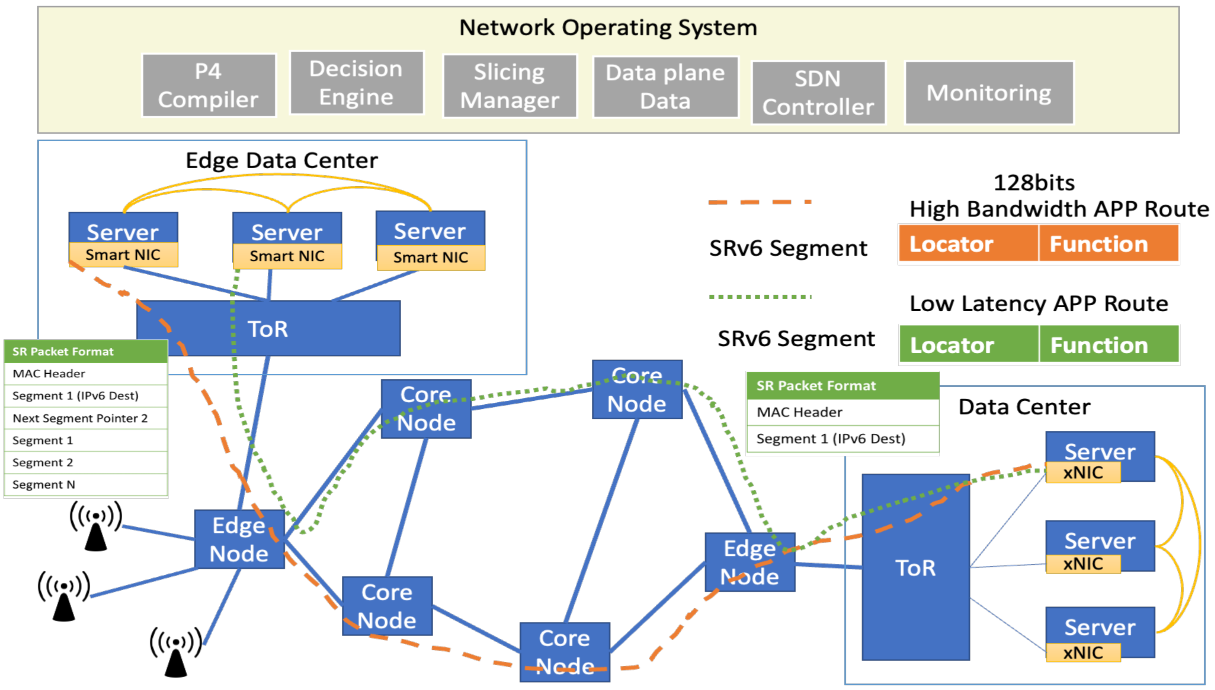


Figure 19:5G SRv6 Scenario Architecture with P4-enabled Smart NIC

Our primary objective was to establish end-to-end network slicing, extending from edge data centers to core data centers, employing segment routing. The solution entailed the direct injection of SRv6 headers [75] and MPLS labels into the Smart NIC. P4 files was written, compiled, and transmitted into the Smart NIC through a socket. This approach significantly enhanced programmability and network performance, while concurrently mitigating CPU utilization, in contrast to the conventional method of inserting SRv6 headers using servers or software.

Figure 19, which is an SRv6 example that showed how packets can be routed through different paths based on the inserted segment identifications. This technique ensures that packets are routed through specific segments to reach high bandwidth or low latency routes, resulting in improved network performance for specific applications and use cases. The solution ensures that the segments are deleted by the end point server Smart NIC when the packets reach their destination.

The approach described in the paragraph provided a more efficient way to manage end-to-end network slicing, delivering optimized performance and reduced CPU utilization. The use of segment routing allowed for traffic to be routed based on specific application requirements, enabling better network performance and lower latency.

4.3.1 *Testbed setup and results*

The optical inter-DC architecture with enabling technologies is shown in the Figure 20, it is based on the 5G smart city testbed setup by High Performance Network Group of Bristol University, they built a city based optical network testbed with latest technologies, and realized a converged 5G fronthaul and backhaul cognitive network. The P4-enabled Smart NICs were inserted in the server for emulating edge data center. The Voyager Transponder was used for the optical side accumulating and transponding.

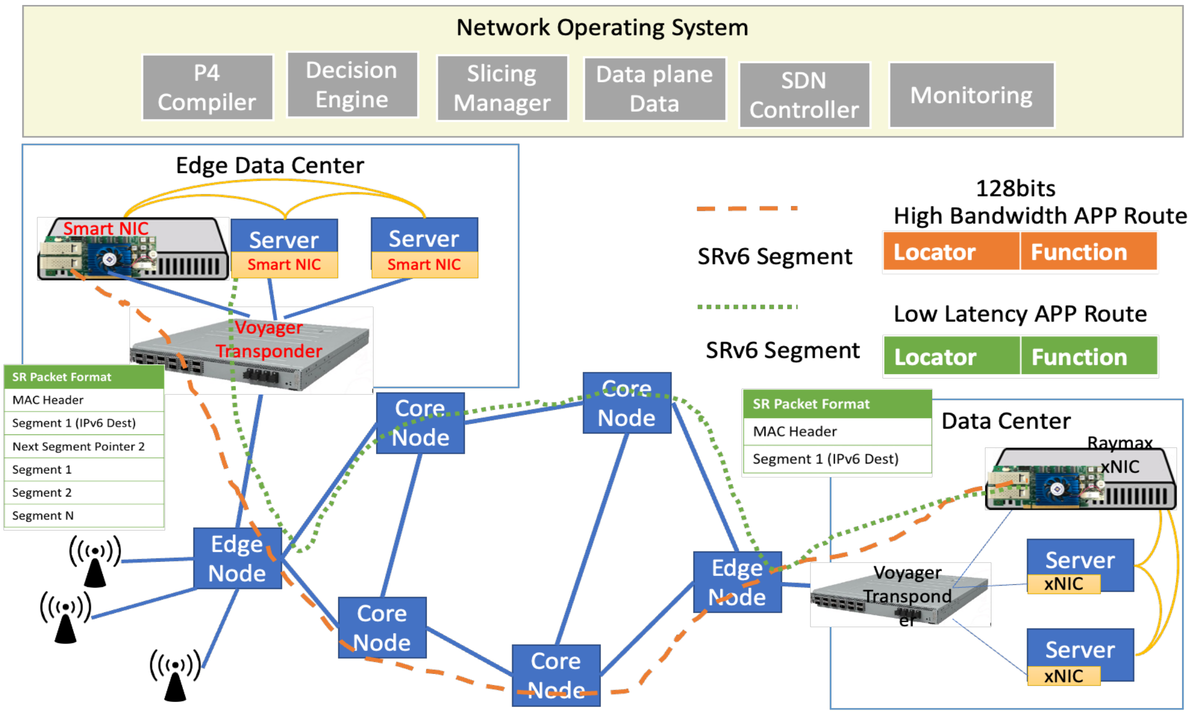


Figure 20: 5G SRv6 Scenario Setup with P4-enabled Smart NIC

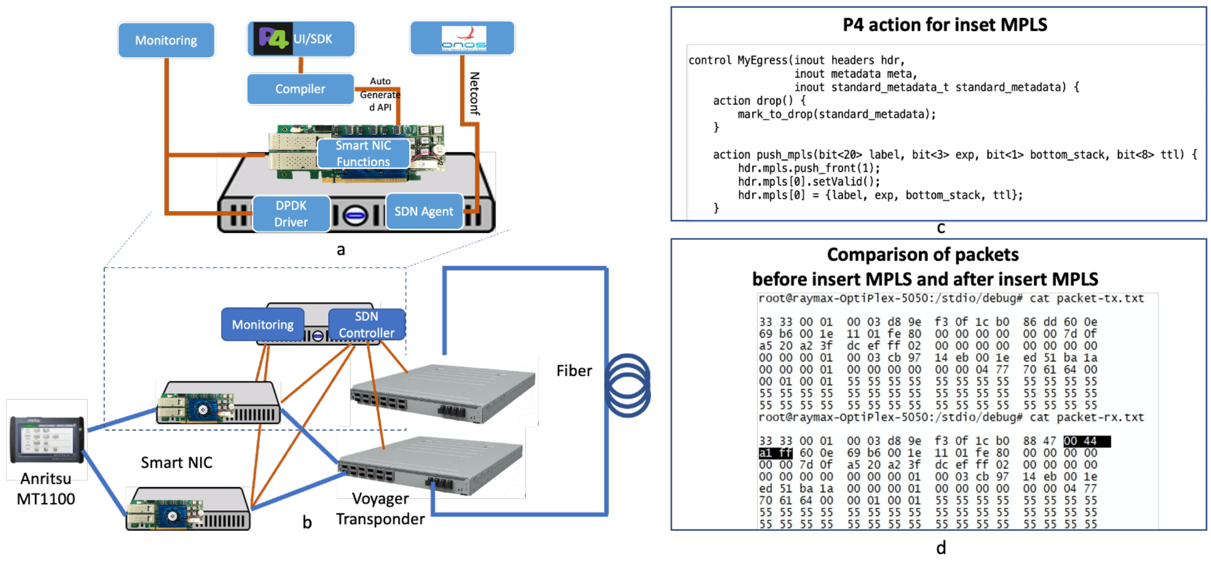
To facilitate the deployment of P4 configurations, I implemented a streamlined process. The written P4 programmes were uploaded through a SDK interface. This interface, in turn, took on the task of compiling and translating the P4 files, ultimately delivering them to the Smart NIC.

The testbed configuration, illustrated in Figure 21, offers a comprehensive environment for experimentation. In Figure 21(a), I established a testbed for testing Smart NIC with both of control plane and data plane components. Here, the Smart NIC was seamlessly integrated into the PCIe slot, and the server itself was equipped with critical software components, including the DPDK driver and an ONOS controller.

A monitoring module was thoughtfully integrated into the testbed to collect, process, and display network statistics and performance metrics from both the Smart NICs and the servers. This ensured comprehensive real-time insight into the network's behaviour and resource utilization.

To further enhance the testbed's capabilities and emulate an edge-DC to DC environment, I incorporated an illustration of IP domain segment routing enhanced with optical domain network slicing. In Figure 21(b), the optical testbed was meticulously set up, featuring Voyager BVT Transponders. In evaluating latency and bandwidth, I harnessed the capabilities of the Anritsu MT1100 traffic analyser for both data generation and analysis.

This comprehensive testbed was designed to assign different wavelengths to SR-MPLS labels, a configuration that facilitates end-to-end network slicing in both the IP and optical domains. The integrated systems offered a versatile environment for extensive experimentation and measurement of network performance across diverse scenarios.



```
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t standard_metadata) {
    action drop() {
        mark_to_drop(standard_metadata);
    }

    action push_mpls(bit<20> label, bit<3> exp, bit<1> bottom_stack, bit<8> ttl) {
        hdr.mpls.push_front(1);
        hdr.mpls[0].setValid();
        hdr.mpls[0] = {label, exp, bottom_stack, ttl};
    }
```

```
root@raymax-OptiPlex-5050:/stdio/debug# cat packet-tx.txt

33 33 00 01  00 03 d8 9e  f3 0f 1c b0  86 dd 60 0e
69 b6 00 1e  11 01 fe 80  00 00 00 00  00 00 7d 0f
a5 20 a2 3f  dc ef ff 02  00 00 00 00  00 00 00 00
00 00 00 01  00 03 cb 97  14 eb 00 1e  ed 51 ba 1a
00 00 00 01  00 00 00 00  00 00 04 77  70 61 64 00
00 01 00 01  55 55 55 55  55 55 55 55  55 55 55 55
55 55 55 55  55 55 55 55  55 55 55 55  55 55 55 55
55 55 55 55  55 55 55 55  55 55 55 55  55 55 55 55
root@raymax-OptiPlex-5050:/stdio/debug# cat packet-rx.txt

33 33 00 01  00 03 d8 9e  f3 0f 1c b0  88 47 00 44
a1 ff 60 0e  69 b6 00 1e  11 01 fe 80  00 00 00 00
00 00 7d 0f  a5 20 a2 3f  dc ef ff 02  00 00 00 00
00 00 00 00  00 00 00 01  00 03 cb 97  14 eb 00 1e
ed 51 ba 1a  00 00 00 01  00 00 00 00  00 00 04 77
70 61 64 00  00 01 00 01  55 55 55 55  55 55 55 55
55 55 55 55  55 55 55 55  55 55 55 55  55 55 55 55
55 55 55 55  55 55 55 55  55 55 55 55  55 55 55 55
```

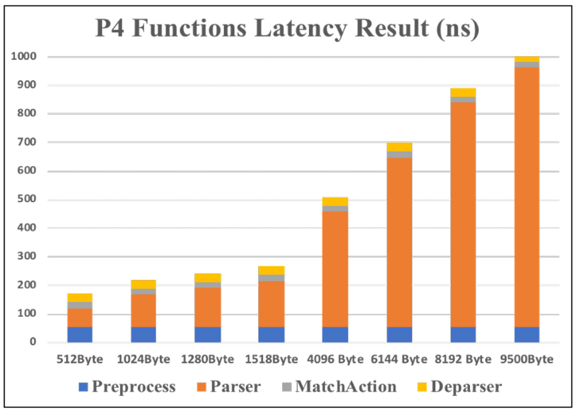
Figure 21:5G use case testbed setup: (a)Server-based testbed setup, (b) Optical testbed setup, (c) P4 action code sample, (d) Comparison of packets.

We wrote the .p4 programmes with the specific functionality of inserting an MPLS-SR header at the ingress and deleting it at the egress stage. In Figure 21 (c), you can observe the P4 action for inserting SR-MPLS, and Figure 21 (d) displays the captured packet after the SR-MPLS insertion.

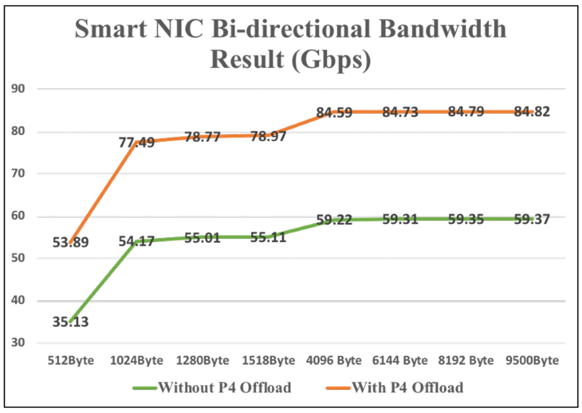
The measurement results, outlined in Figure 22, encompass various aspects. Firstly, I measured the latency of the P4 block within the FPGA, as indicated in Figure 22 (a). The latency breakdown unveiled that the parser's latency was primarily contingent on the Ethernet frame's length, while the latencies of other functional blocks remained relatively consistent.

Moving on to Figure 22 (b), the bandwidth was assessed using a hardware setup consisting of an Intel Core i7-7700K CPU @ 4.20GHz with eight cores and 62.8GiB of memory. Our measurement involved inserting a single SR-MPLS header into the packet and evaluating the maximum bandwidth with and without P4 SR-MPLS offload. The results demonstrated that with offload, the Smart NIC achieved a maximum throughput of 78.97Gbps with 1518 Ethernet frame size, which further increased to 84.82Gbps with jumbo frame size (9500 Bytes). In contrast, without offload, the bandwidth diminished by up to 30%.

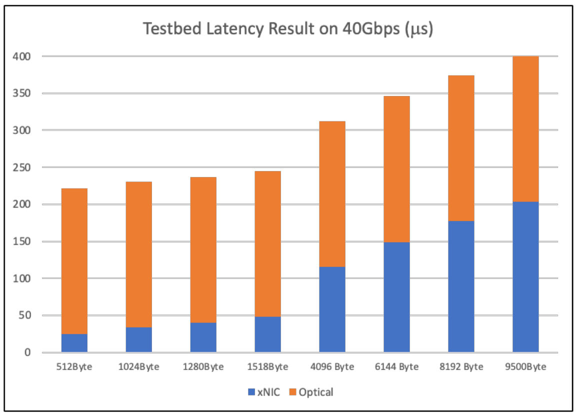
For the comprehensive latency assessment of the entire testbed, I initially planned to employ an Anritsu MT1100. However, the MT1100 being used had only one 100Gbps Ethernet port, whereas the Voyager BVT required 2*100Gbps ports for the setup. Consequently, I conducted separate latency measurements using the MT1100 for Smart NICs, followed by Smart NICs to optical devices and fibers. Figure 22 (c) illustrates the segmented breakdown of the entire testbed's latency, encompassing both Smart NICs and optical devices.



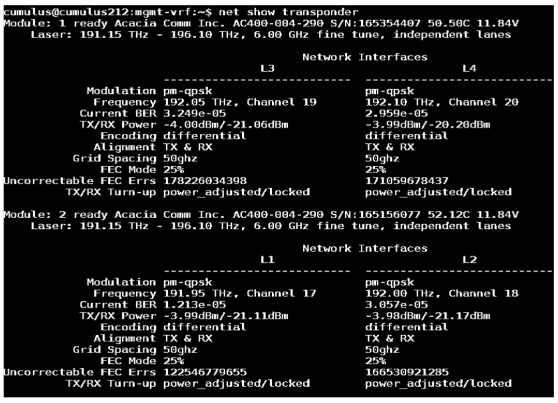
a



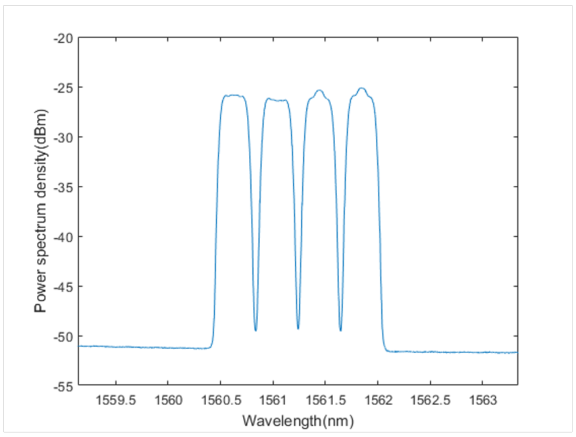
b



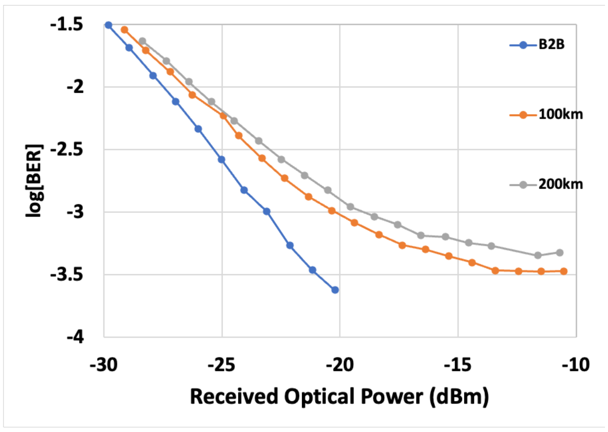
c


```
cumulus@cumulus212:mgmt-vrf:~$ net show transponder
Module: 1 ready Acacia Comm Inc. AC400-004-290 S/N:165354407 50.50C 11.84V
    Laser: 191.15 THz - 196.10 THz, 6.00 GHz fine tune, independent lanes

                                     Network Interfaces
                              L3                              L4
                      ----------------------------  ----------------------------
           Modulation pm-qpsk                       pm-qpsk
            Frequency 192.05 THz, Channel 19        192.10 THz, Channel 20
          Current BER 3.249e-05                     2.959e-05
         TX/RX Power -4.00dBm/-21.06dBm            -3.99dBm/-20.20dBm
             Encoding differential                  differential
            Alignment TX & RX                       TX & RX
         Grid Spacing 50ghz                         50ghz
             FEC Mode 25%                           25%
Uncorrectable FEC Errs 178226034398                  171059678437
        TX/RX Turn-up power_adjusted/locked         power_adjusted/locked

Module: 2 ready Acacia Comm Inc. AC400-004-290 S/N:165156077 52.12C 11.84V
    Laser: 191.15 THz - 196.10 THz, 6.00 GHz fine tune, independent lanes

                                     Network Interfaces
                              L1                              L2
                      ----------------------------  ----------------------------
           Modulation pm-qpsk                       pm-qpsk
            Frequency 191.95 THz, Channel 17        192.00 THz, Channel 18
          Current BER 1.213e-05                     3.057e-05
         TX/RX Power -3.99dBm/-21.11dBm            -3.98dBm/-21.17dBm
             Encoding differential                  differential
            Alignment TX & RX                       TX & RX
         Grid Spacing 50ghz                         50ghz
             FEC Mode 25%                           25%
Uncorrectable FEC Errs 122546779655                  166530921285
        TX/RX Turn-up power_adjusted/locked         power_adjusted/locked
```

d



e



f

Figure 22:The Measurement Results of the P4-enabled Smart NIC: (a) Separated Latency Results of P4 Blocks, (b) With P4 Offload and without P4 offload bandwidth comparison results, (c) On 40Gbps throughput, the measured latency result, (d) The setup details of Voyager BVT,(e) The Power Spectrum density result, (f) The measured Long haul BER.

Figure 22 (d) and (e) present the optical outcomes. We configured the Voyager BVT with a Quadrature Phase Shift Keying (QPSK) modulation format. The frequencies were set at 191.95THz, 192.00THz, 192.05THz, and 192.10THz, each aligning with an MPLS segment to facilitate optical domain slicing. The spectral outcome is depicted in Figure 22 (e). It's worth noting that the Voyager BVT's specifications allow for the adjustment of the modulation format to 8QAM or 16QAM to accommodate longer distances or higher bandwidth requirements, although this was not shown in the figure.

Moving to Figure 22 (f), it illustrates the Bit Error Rate (BER) in a long-haul network scenario. In this context, the worst-case scenario corresponds to the Voyager's modulation of 16QAM. The increased BER observed in the 100km and 200km cases can be attributed to amplifier noise and the dispersion induced by optical fiber transmission.

Chapter 5. P4-ENABLED SMART NIC FOR ACCELERATING INTRA-SERVER VIRTUAL SWITCHING

The business landscape has seen an increase in demands for higher network bandwidth and lower latency, fueled by the rapid advancements in cloud computing technology and the continuous expansion of cloud scale in recent years. As described in the previous Although the OvS-DPDK approach can accelerate network packet processing in virtual networks and outperforms the OvS kernel datapath, it has certain limitations in terms of performance expansion and cost-effectiveness since OvS is a software-based processing and interaction, it can consume a significant amount of CPU resources and impact network bandwidth. As a result, it may no longer suffice to meet the increasingly high performance demands.

In the industry, the quest for accelerating virtual networks predominantly hinges on hardware-level solutions like ASICs, Smart NICs, DPUs, and other analogous technologies. Smart NICs, in particular, offer distinct advantages due to their proximity to network data. Additionally, this thesis centers its primary objective on the utilization of P4 for data plane programming within the Smart NIC.

While the P4 community has made commendable strides in adapting P4 to increasingly programmable targets, such as those with OVS-like functionalities (e.g., p4c-ubpf), it's noteworthy that these adaptations currently support BPF backend devices that already accommodate OVS offload [76]. There exists a compelling need for further endeavors to render generic P4 data planes adaptable to network offloading applications, including OVS offload.

Within this chapter, I present the diligent efforts undertaken to optimize the P4-enabled data plane architecture, specifically aiming to achieve an OvS data plane and elevate Smart NIC performance through acceleration.

5.1 P4-enabled Smart NIC Dataplane for OvS acceleration

OvS served as a powerful tool for bridging traffic between VMs and the external world. Our core objective was to harness hardware acceleration fully, effectively addressing L2 switching, L3 routing, and overlay network protocols to optimize VM bandwidth and minimize CPU workloads.

As delineated in earlier chapters, both P4 and OVS dataplanes played crucial roles in packet processing and forwarding. P4, as a network dataplane programming language, offered an exceptionally high degree of programmability. It empowered network administrators to tailor packet processing and forwarding according to the specific requirements of applications and network structures. Custom packet forwarding rules and policies could be crafted with precision to accommodate diverse scenarios.

On the other hand, OVS, as virtual switch software, also featured a programmable dataplane, albeit to a lesser extent than P4. OVS came with a set of predefined dataplane processing rules, offering the flexibility for configuration and customization to adapt to different network settings. OVS was designed to complement other SDN technologies like OpenFlow, contributing to a holistic software-defined networking solution.

The high-level architecture for FPGA-based Smart NIC OVS offload was visually depicted in Figure 23. The central challenge revolved around optimizing the FPGA's inherent parallel processing capabilities and making the most of PCIe SR-IOV features. To enable offloading, the FPGA-based OVS dataplane needed to implement parser, match, and action functions, aligning with the software-based OVS dataplane. The number of flow rules that the

FPGA could support was constrained by its storage resources. The hardware-level mapping to the P4-enabled pipeline had been demonstrated in prior chapters.

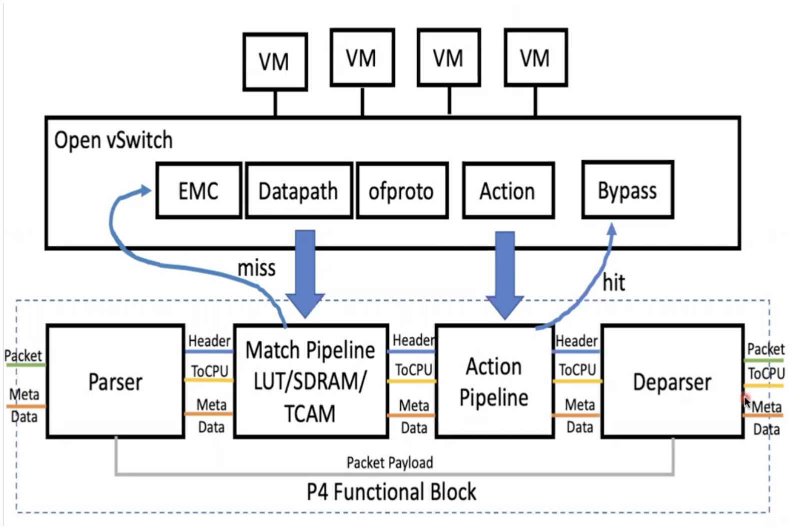


Figure 23:OVS data plane offload matching to the P4 functional block

5.2 *Innovated P4-enabled dataplane architecture with OvS acceleration capability*

The implementation was grounded in OVS-2.11.1, and it leveraged OVS-DPDK, which comprised three tiers of lookup tables and caches: the Exact Match Cache (EMC), the dpcls data path (mega flow cache), and the OpenFlow pipelines ofproto. These three stages of matching rules were to be seamlessly integrated into the FPGA-based Smart NIC.

In this configuration, incoming packets encountered the flow tables, and based on matching rules, they were either directly routed to the corresponding virtual machines (VMs) or redirected to the OVS data path for software-based matching and associated actions.

On the control plane side of OVS, inherent support for hardware offloading was present. To enhance this capability, an "init_flow_api" interface within the "netdev_class dpdk_class" of "netdev-dpdk" was introduced. This addition further strengthened the hardware offload acceleration, contributing to the overall optimization of the system's performance.

The innovation of proposed OvS offloading leveraged the P4 language-based dataplane architecture designed and implemented in Chapter 3. The OvS data plane was implemented by mapping the OvS dataplane to the P4 data plane as a P4-based application, which can realize the resource multiplexing of P4 and OvS data plane. Thus can save up to 40% of logic resources.

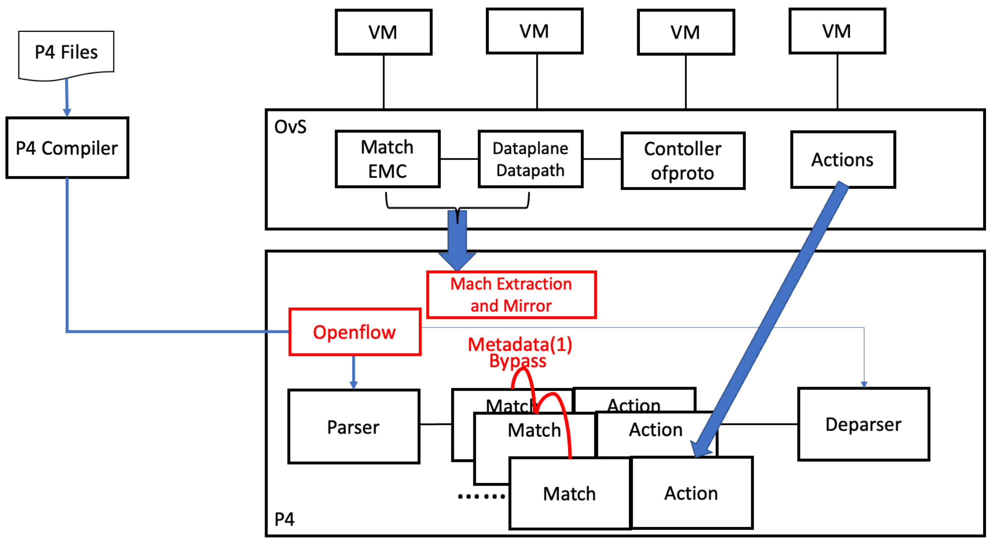


Figure 24:Proposed OvS offloading leveraging the P4 language-based dataplane architecture

The architecture of P4 language-based dataplane is demonstrated in Figure 24. The entire P4 programmable data plane consists of a parse message module, a multi-level matching and action module, and a reassembly message module. Among them, the multi-level matching module realizes multi-level search matching, and each level includes exact matching based on hash operation and fuzzy matching based on three-state content addressing memory.

In order to adapt to the OvS data plane, the present invention adds a bypass (bypass) module when implementing the P4 data plane, and adds a bypass indicator bit in the P4 program user-defined metadata (metadata) to indicate whether the data flow needs to be bypassed. When the application was offloaded for the OvS data plane, the P4 data plane was determined according to the openflow protocol of OvS. Then, the exact matching EMC of OvS was mapped to the first-level lookup match of the P4 data plane, and the dataplane matching Datapath of OvS was mapped to the second-level lookup matching of P4. Map OvS actions to P4's action module.

Whenever there was a corresponding match in any level, the metadata bypass is marked as 1, the next level of bypass matches, and finally the action was done according to the rules, if there was no corresponding match, the metadata bypass was represented as 0, and the next level of matching was done. This could realize the full unloading of the data plane of OvS based on the data plane of P4.

The flow journey followed the steps below:

Firstly, In the P4 data plane-based smart NIC, P4 programs are written according to the Openflow protocol that OvS follows to realize the P4 programmable data plane based on the Openflow protocol. For example, Openflow 1.3 includes a capability representation framework for length and numeric values, as well as support for IPv6 extension headers.

Secondly, map the exact matching EMC of OvS to the first-level lookup matching of the P4 data plane, map the data plane matching of OvS to the second-level lookup matching of the P4 data plane, and map the action items of OvS to the action modules of the P4 data plane.

Thirdly, when the packet ingressed to the P4 first level of matching (the original OvS accurately matched EMC) and matched the corresponding rule, set the Metadata bypass to indicate that the bit was 1. The flow would be notified to bypass the subsequent levels of the Bypass module to find the matching module, enter the action module, and perform actions and corresponding forwarding according to the requirements of the action module. When the packets did not find the flow to be matched, it entered the second level of finding the match (the original OvS data plane matched).

After that, when the packets matched the corresponding rule through the P4 second-level search match (the original OvS data plane match), set the Metadata bypass to indicate that the bit was 1. The flow would be notified to bypass the subsequent levels through the Bypass module to find the matching module, enter the action module, and perform actions and corresponding forwarding according to the requirements of the action module. When the packets did not find a flow to match, it entered the OvS control plane library Ofproto through PCIe/DMA, and the control plane issued forwarding rules.

Finally, the packets went through the first-level search and matching module, matched the corresponding flow table, set the Metadata bypass bit to 1, and bypassed the rest to find the match, found the corresponding action, and was forwarded to the corresponding port.

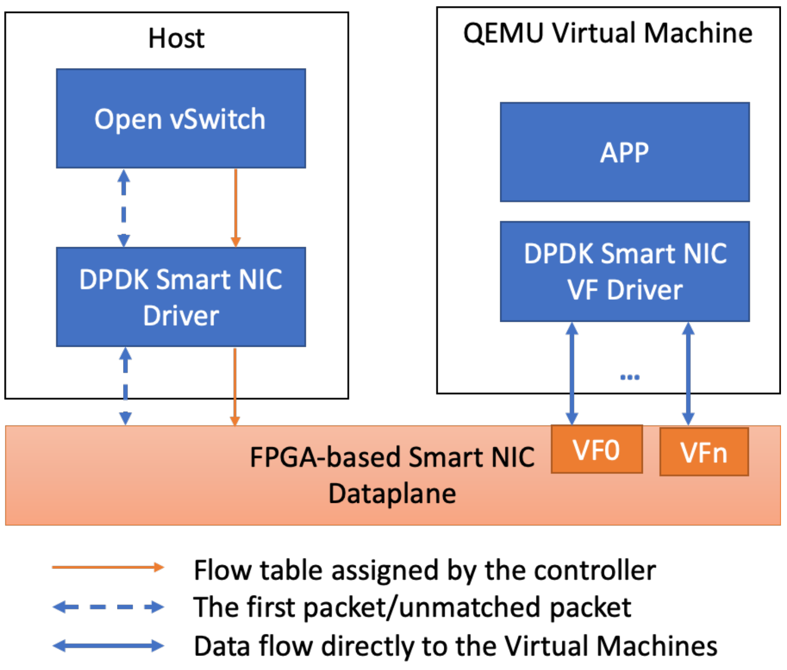

Host
Open vSwitch
DPDK Smart NIC Driver
QEMU Virtual Machine
APP
DPDK Smart NIC VF Driver
...
FPGA-based Smart NIC Dataplane
VF0
VFn
Flow table assigned by the controller
The first packet/unmatched packet
Data flow directly to the Virtual Machines

Figure 25:Host and Virtual Machine Data flow

The Host and Virtual Machine data flow of the packets was shown in Figure 25. The Smart NIC for intra-server virtual switching offloading solution is based on the open source projects Open vSwitch and DPDK, and the overall implementation architecture was shown in Figure 26.

In the virtualization environment, OvS uses DPDK for acceleration, its data planes are implemented in user mode, and QEMU virtual machines are configured in VF passthrough mode using the Smart NIC SR-IOV feature. When the OvS dataplane offloading feature is enabled, the DPDK driver interacts with the Smart NIC PF, and the first packet of the session coming in from the physical port will trigger the flow table Offload process (indicated by the red arrow), and finally all flow table rules will be maintained in the Smart NIC offload module, and subsequent data frames of the session will be directly processed in the network card hardware and forwarded to the specified VF, and the corresponding virtual machine can receive the packet. Session packets sent from virtual machines are processed in the same way.

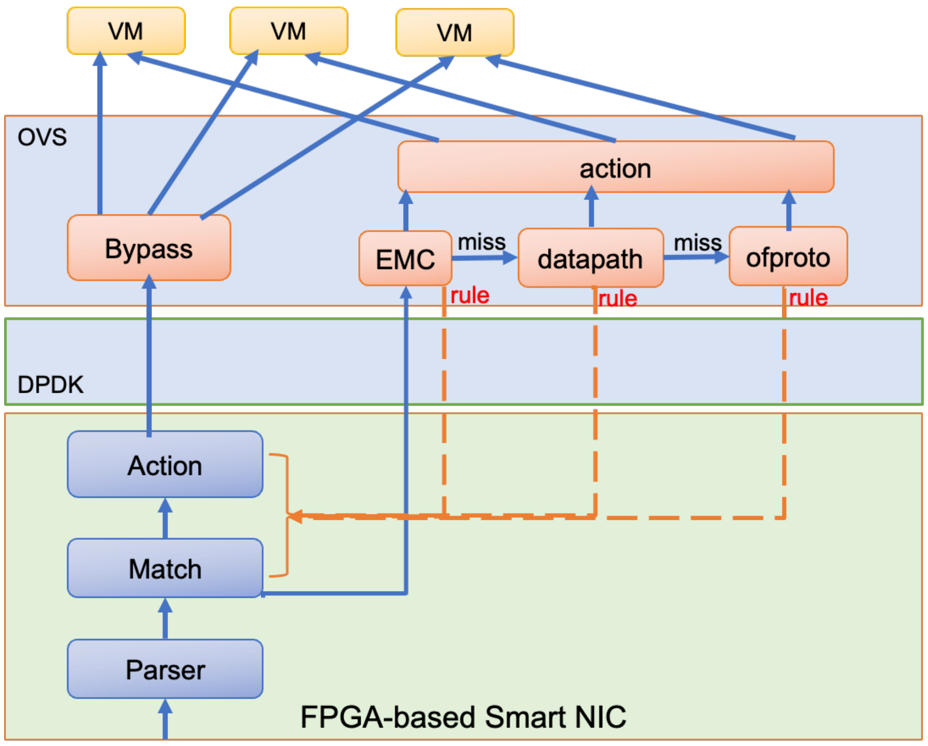


Figure 26:FPGA-based Smart NIC OvS dataplane offload steering flow

The network of QEMU virtual machines in the traditional virtualization environment used VirtIO to communicate with OvS, and the memory copies during data transmission and receiving consumed loads of CPU resources. In scenarios where HOST has multiple virtual machines, the virtual network performance fluctuates due to the load of server system resources (such as memory, bus bandwidth, cache, etc.), and the overall network bandwidth and latency performance are poor. The proposed Smart NIC solution has greatly improved performance compared with traditional methods, and the comparative test analysis is displayed followingly.

5.3 Intra-sever Acceleartion Testbed Setup and Measured Results

In our endeavor to gauge the performance of OVS offload, a dedicated testbed was meticulously configured for an in-depth assessment, which were previous published in our paper [82]. This testbed was furnished with a dual-port 25Gbps Smart NIC, seamlessly integrated into a Supermicro server. The server was no ordinary piece of hardware, boasting the prowess of an Intel Xeon Gold 6146 processor running at a clock speed of 3.2GHz. Furthermore, it featured a total of 12 cores, complemented by a generous 182GB of memory, clocked at an impressive 2666MHz.

To enable OVS data plane support, I engaged the services of the P4 compiler to compile the requisite P4 file. This compiled file was then translated to the FPGA, a crucial step in empowering the FPGA to support the OVS data plane.

The Smart NIC platform being employed in this evaluation bore a close resemblance to the one previously discussed in Chapter 4. It featured the Xilinx Virtex Ultrascale+ VU3P as the underlying implementation platform, which provided the ideal foundation for our comprehensive analysis of OVS offload performance. This rigorous setup allowed us to delve deep into the intricacies of OVS offload performance, providing valuable insights for our assessment.

5.3.1 FPGA-based Data Plane

The traffic flow was initiated from the Anritsu MT1100 traffic generator, routed through the SFP28 transceiver, and directed towards the Smart NIC. In the scenario without OVS offload, as illustrated in Figure 27 (a), the packets followed a sequence. Initially, they were sent to the OVS for matching and action processing, subsequently forwarded to the virtual machine (VM). After being processed by the VM, the packets looped back to the OVS and finally to the NIC before being subjected to measurement by the traffic analyzer.

In contrast, when OVS offload was in operation, as depicted in Figure 27(b), a different path was followed. Rather than undergoing processing by the OVS, the packets were efficiently matched within the Smart NIC. They were then directly transmitted to the VM, and upon completion of VM processing, they returned to the NIC. Ultimately, these packets were measured by the traffic analyzer. This streamlined approach through OVS offload significantly improved efficiency and reduced processing steps.

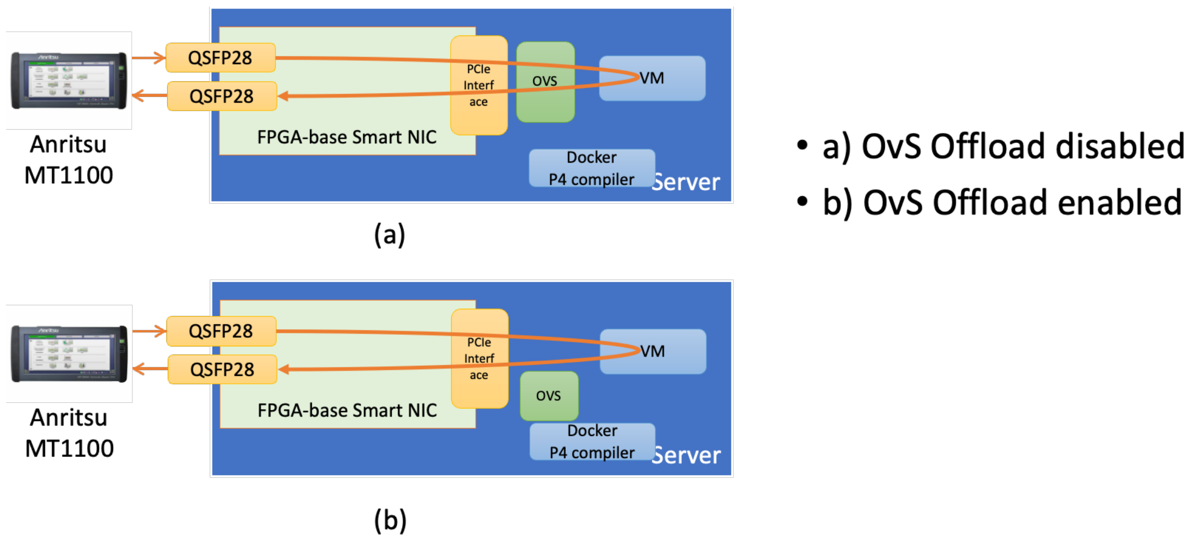
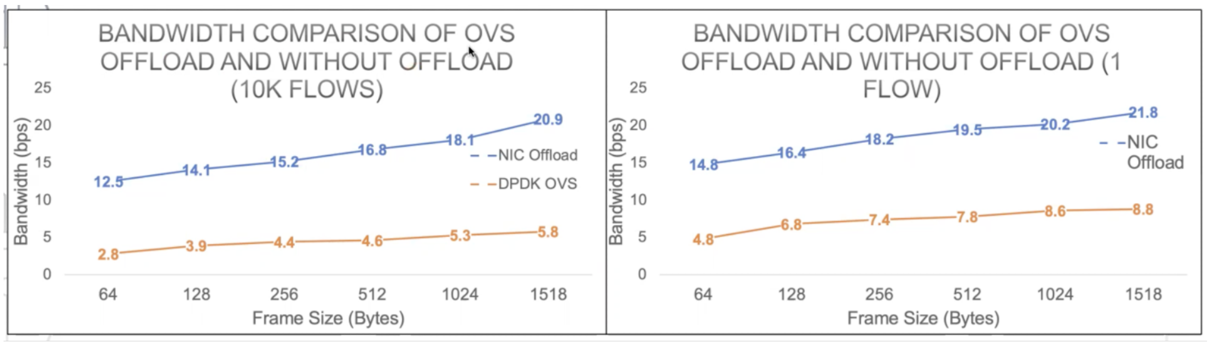


*Figure 27:*OvS data plane offload Test bed setup: (a) Ovs Offload disabled, (b) OvS offload enabled.

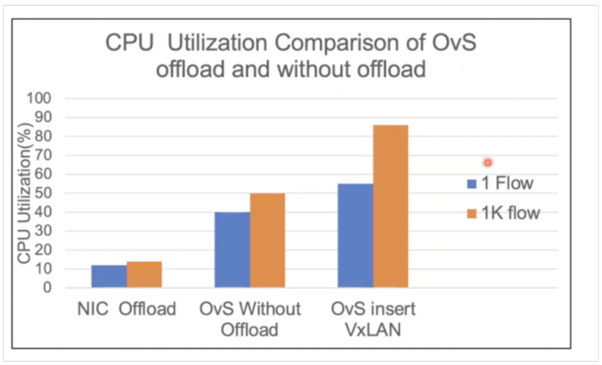
The OVS implementation was designed to accommodate a maximum of 10,000 rules. Our bandwidth measurements were structured around different numbers of traffic flows. In Figure 28(a), the measured bandwidth was presented when comparing OVS offload with its absence, specifically focused on a scenario with only one flow. Meanwhile, Figure 28(b) showcased the measured bandwidth with an emphasis on 10,000 flows. Both sets of results indicated a remarkable upsurge in bandwidth, with Smart NIC's OVS data path offload delivering a substantial increase of 3x to 5x compared to the non-offloaded setup.

Furthermore, in Figure 28(c), I extended our analysis to delve into CPU utilization when processing a single flow and 1,000 flows, comparing scenarios with OVS data path offload to those without. Here, I opted to measure 1,000 flows, as handling more than 1,000 flows necessitated the engagement of more than one CPU core for OVS processing. The findings unveiled a significant reduction in CPU utilization, with OVS offload offering a substantial maximum decrease of 70%.

These insights underscore the substantial benefits of OVS data path offload on both bandwidth and CPU utilization, further accentuating the efficacy of this approach in optimizing network performance.



(a) (b)



(c)

*Figure 28:*Measurement Results based on number of flows: (a) Bandwidth comparison 10K flows, (b) Bandwidth comparison 1K flows, CPU utilization comparison.

5.3.2 Physical Function

The OvS mode without offloading setup is shown in Figure 29 (a). The data received from PF0, forwarded it to Port0 after processed by the OvS dataplane. Figure 29 (b) shows that when the FPGA-based Smart NIC turns on the offloading mode, after the flow table rule is offloaded from OvS to the network card, the data is directly forwarded after the Smart NIC is processed, and no longer enters the OvS software data plane.

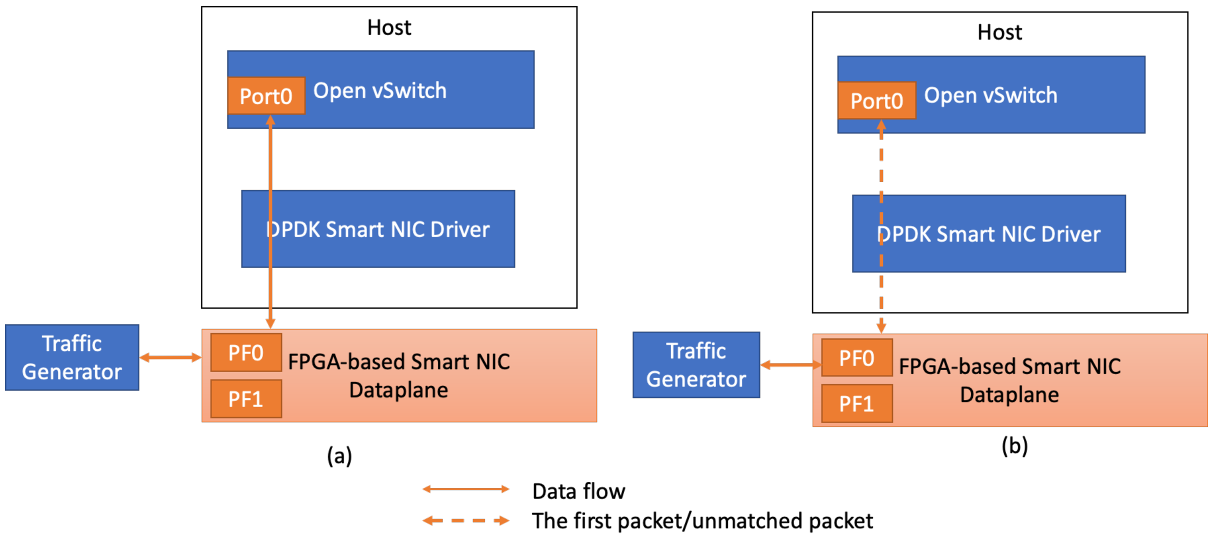


Figure 29:OvS PF mode testbed setup: (a) Without dataplane offloading, (b) With dataplane offloading

After the comparison test as described above, the SmartNIC offload function improved the 64B packet throughput performance by more than 1 times, and the latency was significantly reduced, as shown in Table 9.

Table 9: Smart NIC OvS mode vs Dataplane Offload mode Performance

Type	**Single-Port Bandwidth (Gbps)**	**Dual-Port Bandwidth (Gbps)**	**Latency (µs)**
OvS mode	3.6	6.8	16
OvS Dataplane Offload mode	7.6	14	2.5

5.3.3 Virtual Function

When using VirtIO mode as shown in Figure 30(a) below, port1 is the Vhost-user port, QEMU virtual machine uses the corresponding VirtIO-Net driver, and testpmd uses single-core single-queue mode to forward data.

When the Smart NIC turned on OvS Dataplane offload mode as shown in Figure 30(b) below, the network device of the QEMU virtual machine used the SR-IOV passthrough mode, and the OvS datapath processing was directly implemented in the xSmartNIC network card.

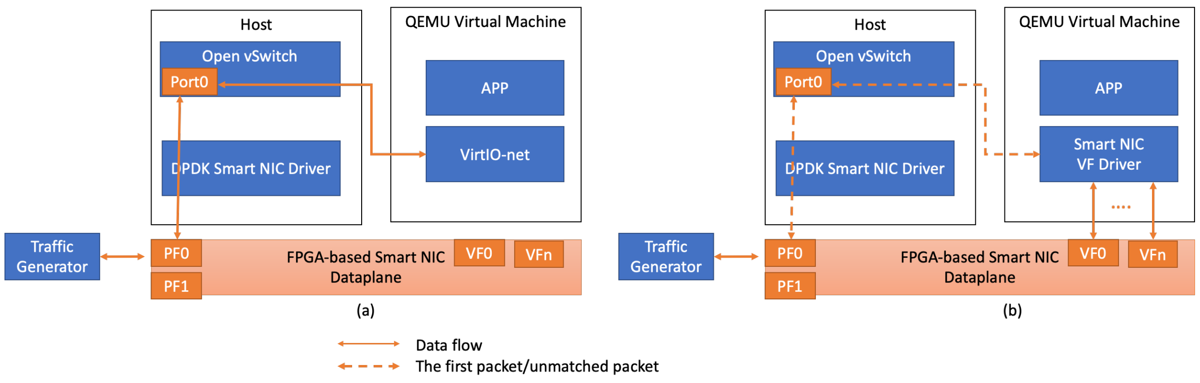


Figure 30: OvS OvS VF mode testbed setup: (a) Without dataplane offloading, (b) With dataplane offloading

According to the above two methods, when PF0 processes four virtual machines (each virtual machine forwards packets independently), the throughput and latency comparison test is shown in Figure 29 below, and the results show that the Smart NIC OvS dataplane offload acceleration function can greatly improve the network card throughput of the virtual machine and reduce the network latency.

When PF0 processes 4 virtual machines and packet traffic passes through each virtual machine in turn, the comparative test results are shown in Figure 31, compared with the previous scenario, when the virtual machine consumes more memory bandwidth resources, the more significant the acceleration performance of Smart NIC OvS offload, the bandwidth of small packets below 512B can be increased by 15~30 times, and the bandwidth of large packets above 512B can be increased by about 5 times, and the delay will be reduced to about 25% of the original.

In addition, under the same network performance, the Smart NIC OvS dataplane offload solution will significantly reduce the consumption of CPU computing power and memory bandwidth compared to Virtio-net.

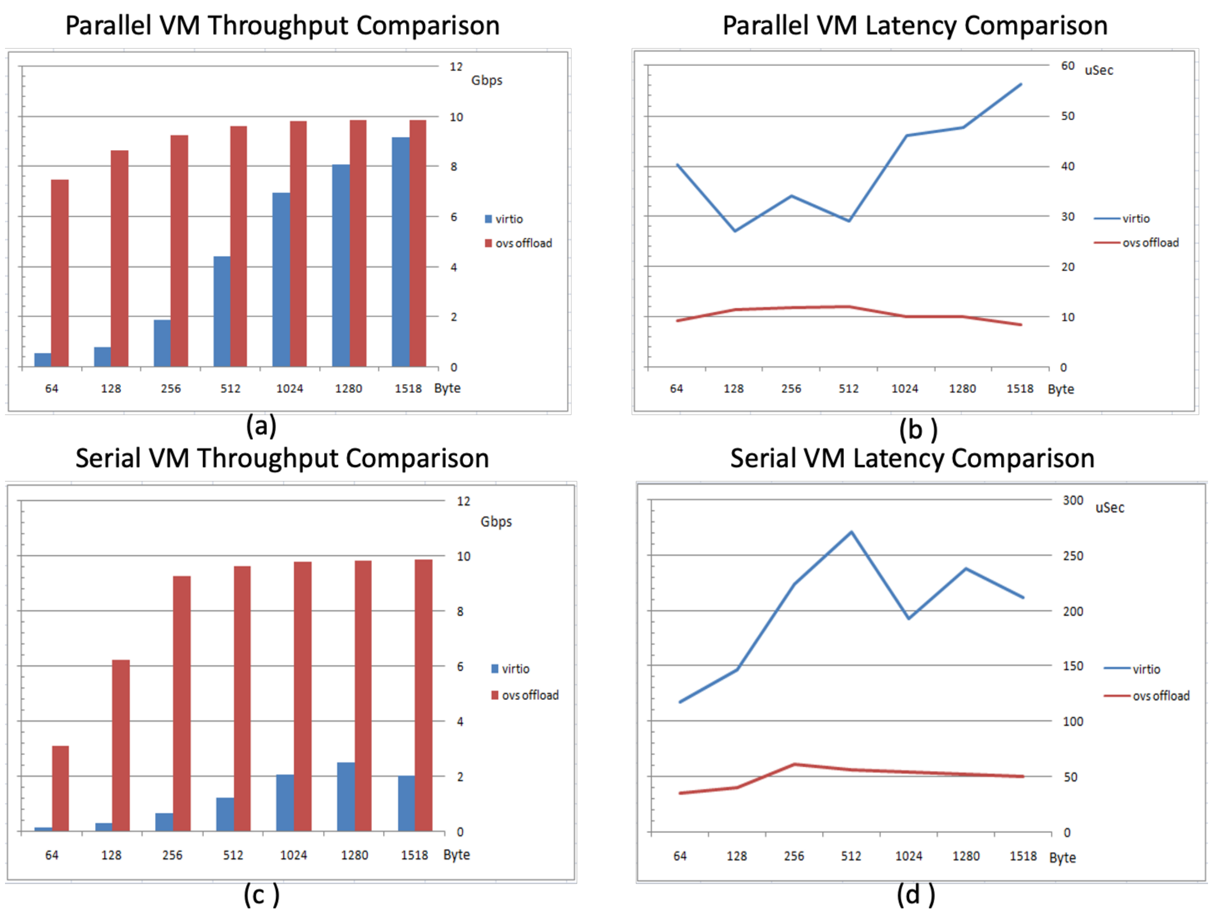


Figure 31:Measurement results: (a)Parallel virtual machine Smart NIC OvS offloading throughput comparison test results; (b)Parallel virtual machine Smart NIC OvS offloading latency comparison test results;(c) Serial Virtual Machine Smart NIC OvS offloading bandwidth comparison test results; (d) Serial Virtual Machine Smart NIC OvS offloading latency comparison test results

Chapter 6. CONCLUSION AND FUTURE WORK

Technologies around us are becoming increasingly smarter, and they demand solutions that can adjust to their greater needs. Smart NICs, providing low latency, broad flexibility on the server edge, offloads processing and calculation burdens from the host server CPUs to achieve better performance and lower cost on the server. Smart NICs could be used in the scenario of Clouds, 5G, network security, AI and so on, for whatever the servers were massively employed.

This thesis supplied the concepts and the technologies developed in the last decade, from SDN to P4, from rigid to Virtualization, from NIC to Smart NIC. We proudly designed and developed a FPGA-based Smart NIC with programmable functional data plane that can accelerate several scenarios such as Cloud and 5G. Several innovative P4-dataplane based design were created for these scenarios for accelerating the network with single FPGA-based data plane design. The implementations were displayed, and the test results were demonstrated showing better throughput and lower latency.

With the explosion of the data today, we are facing a network innovation era. The Data Center network faces challenges more than ever before. In the time of writing the thesis, the ChatGPT has boosted the world with its amazing abilities, however, behind the scene, is the starving demand for CPUs and GPUs. How to make the server network more agile to be service-driven and serve big amount of data in ultra-low latency need efforts from researchers, communities, companies, and etc.

With the evolution of the networks, the dataplane hardware and even the servers themselves, the Smart NICs will face more challenges for supplying more CPU oriented, and easy-to-deploy hardware devices, no matter whether its name is Smart NIC or DPU or IPU or etc. I will continue put my efforts on the smart hardware for faster network!

BIBLIOGRAPHY

[1] Cicsco, “Cisco Annual Internet Report （2018–2023）White Paper” , 2020.

[2] Lockee, B.B., “Online education in the post-COVID era.”, Nat Electron 4, 5–6, https://doi.org/10.1038/s41928-020-00534-0, 2021.

[3] Ed Lee, Philip Bues, Frank Dickson,“Worldwide cloud workload security forecast,2023-2027: Complexity and Resiliency Fuel Growth,” Market Forecast, Jun 2023.

[4] Gartner,”Gartner Forecasts Worldwide 5G Network Infrastructure Revenue”, https://www.gartner.com/en/newsroom/press-releases/2021-08-04-gartner-forecasts-worldwide-5g-network-infrastrucutre-revenue-to-grow-39pc-in-2021, 2021.

[5] Varun Gupta, Sandun Pereara,“Managing surges in online demand using bandwidth throttling: An optimal strategy amid the COVID-19 pandemic,” Transportation Research Part E: Logisctics and Transportation Review, July 2021

[6] Moore’s law, http://www.mooreslaw.org, 2023.

[7] Claus Pahl, Antonio Brogi, Jacopo Soldani, and Pooyan Jamshidi, “Cloud Container Technologies: a State-of-the-Art Review,” IEEE Transactions on Cloud Computing, 2017.

[8] Open Networking Foundation, “OpenFlow Switch Specification,” 2014.

[9] ETSI, “NFV – Update White Paper”, Washington University in St. Louis, 2013.

[10] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard et al., “Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN,” ACM SIGCOMM,2013.

[11] Open Compute Project,“OCP is reimagining hardware”, https://www.opencompute.org/, 2021.

[12] Bijan R. Rofoee, Georgios Zervas, Yan Yan, Dimitra Simeonidou “Griffin: Programmable Optical DataCenter with SDN Enabled Function Planninga nd Virtualisation,” Journal of Lightwave Technology, 2016.

[13] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford et al., “P4: Programming Protocol-independent Packet Processors,” SIGCOMM Comput. Commun. Rev. 44, 3, July 2014.

[14] Barefoot, “P4-programmable Ethernet switch ASICs”, https://barefootnetworks.com/products/brief-tofino/ , 2020.

[15] Xilinx, “P4-SDNET user guide”, 2018.

[16] Broadcom,”High-Capacity StrataXGS Trident 3 Ethernet Switch Series”, https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56870-series/, 2021.

[17] Jamsvinder s., “Introduction the the Data Plane Development Kit (DPDK) Packet Framework”, Intel, 2017.

[18] DPDK, “Data Plane Development Kit”, http://dpdk.org/, 2021.

[19] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, and Pravin Shelar, “The Design and Implementation of Open vSwitch”, In 12th USENIX

Symposium on Networked Systems Design and Implementation (NSDI 15), pages 117–130, Oakland, CA, 2015.

[20] Ben Pfaff, Justin Pettit, Teemu Koponen, Keith Amidon, Martin Casado, Scott Shenker, "Extending Networking into the Virtualization Layer", Hotnets, 2009.

[21] SDX Central Studios, "What is Open vSwitch", https://www.sdxcentral.com/cloud/open-source/definitions/what-is-open-vswitch, 2022.

[22] FD.io, "The fast data project", https://fd.io, 2022.

[23] OPNFV, "Open Platform for NFV", https://www.opnfv.org , 2022.

[24] Open Networking Foundation, "L4-L7 Service Function Chaining Solution Architecture", Version 1.0, ONF TS-027, 14 June 2015.

[25] Light Reading, "Validating Cisco's NFV Infrastructure", https://www.lightreading.com/nfv/nfv-tests-and-trials/validating-ciscos-nfv-infrastructure-pt-1/d/d-id/718684?page_number=8, 2021.

[26] Microsoft, "TCP/IP offload", https://docs.microsoft.com/en-us/windows-hardware/drivers/network/tcp-ip-offload, 2021.

[27] Clifford B. Melzer, Jonathan Rosen, Robert O'Gorman, Paul A. Wood, Mark C. Drummond, Dean Hiller, IP checksum offload, Patent, US5898713A

[28] Vmware, "TCP segmentation offload", https://docs.vmware.com/en/VMware-vSphere/6.0/com.vmware.vsphere.networking.doc/GUID-E105A601-9331-496C-A213-F76EA3863E31.html, 2021.

[29] Microsoft, "Overview of Single Root I/O Virtualization", https://docs.microsoft.com/en-us/windows-hardware/drivers/network/overview-of-single-root-i-o-virtualization--sr-iov , 2021.

[30] A. Caulfield, P. Costa, M. Ghobadi, "Beyond SmartNICs: Towards a Fully Programmable Cloud," In Proc. of IEEE HPSR, 2018.

[31] Daniel Firestone, Andrew Putnam Sambhrama Mundkur Derek Chiou Alireza Dabagh, Mike Andrewartha et al., "Azure Accelerated Networking: SmartNICs in the Public Cloud," in NSDI'18, 2018.

[32] Netronome, "Netronome and P4: A brief history and a roadmap", https://www.netronome.com/blog/netronome-and-p4-a-brief-history-and-a-roadmap/ , 2021.

[33] P4, "P4 to NetFPGA", https://p4.org/p4/p4-netfpga-a-low-cost-solution-for-testing-p4-programs-in-hardware.html, 2021

[34] Netcope, "Netcope P4", https://www.netcope.com/en/products/netcopep4, 2019.

[35] Netronome, "eBPF Introduction", https://www.netronome.com/technology/ebpf/, 2022.

[36] Han Wang, Robert Soule ́, Huynh Tu Dang, Ki Suh Lee, Vishal Shrivastav, Nate Foster, and Hakim Weatherspoon. 2017. P4FPGA : A Rapid Prototyping Framework for P4. In Proceedings of ACM Symposium on SDN Research conference, Santa Clara, California USA, April 2017.

[37] Netmap, "The Netmap project, the fast packet I/O framework", http://info.iet.unipi.it/~luigi/netmap , 2020.

[38] Y. Yan, G. Zervas, B. R. Rofoee, D. Simeonidou, "High performance and flexible FPGA-based time shared optical network (TSON) metro node", ECOC, 2012.

[39] Y. Yan, G. Zervas, Y. Qin, B.R. Rofoee & D. Simeonidou, 'High performance and flexible FPGA-based time shared optical network (TSON) metro node'. Optics Express, vol 21., pp. 5499-5504, 2013.

[40] Y. Yan, G. Zervas, B. R. Rofoee, D. Simeonidou, "FPGA-based Optical Network Function Programmable Node", OFC, 2014.

[41] Y. Yan, Y. Shu, G. Saridis, B. R. Rofoee, G. Zervas, D. Simeonidou, "FPGA-based optical Programmable Switch and Interface Card for Disaggregated OPS/OCS Data Centre Networks" ECOC, 2015.

[42] Y. Yan, G. M. Saridis, Y. Shu, B. R. Rofoee, S. Yan, M. Arslan, T. Bradley, N. V. Wheeler, N. H. L. Wong, F. Poletti, M. N. Petrovich, D. J. Richardson, S. Poole, G. Zervas, D. Simeonidou, "All-Optical Programmable Disaggregated Data Centre Network realized by FPGA-based Switch and Interface Card," Submitted and accepted in Journal of Light wave Technology, 2016.

[43] Rofoee, B. R., Zervas, G., Yan, Y., Amaya Gonzalez, N. & Simeonidou, D. "Griffin: Programmable Optical DataCenter With SDN Enabled Function Planning and Virtualisation", Journal of Lightwave Technology.volume,33 Issue,24, 2016

[44] G. M. Saridis et al., "Lightness: A Function-Virtualizable Software Defined Data Center Network With All-Optical Circuit/Packet Switching," in Journal of Lightwave Technology, vol. 34, no. 7, pp. 1618-1627, 1 April1, 2016.

[45] S. Y. Yan, E. Hugues Salas, V.J.F Rancano,Y. Shu, G. Saridis, B. R. Rofoee, Y. Yan, A. E. Peters, S. Jain, T. May-Smith, P. Petropoulos, D. J. Richardson, G. Zervas, D. Simeonidou "Archon: A Function Programmable Optical Interconnect Architecture for Transparent Intra and Inter Data Center SDM/TDM/WDM Networking" Journal of Lightwave Technology. 33, 8, p. 1586-1595 10 p, 2015.

[46] Bijan Rahimzadeh Rofoee, George Zervas, Yan Yan, and Dimitra Simeonidou, "Griffin: Programmable Optical DataCenter with SDN enabled Function Planning and Virtualisation" Submitted to Journal of Light wave Technology (JLT), 2015.

[47] S. Y. Yan, Y. Yan, B. R. Rofoee, Y. Shu, E. Hugues Salas, G. Zervas, D. Simeonidou, "Real-Time Ethernet to Software-Defined Sliceable Superchannel Transponder" Journal of Lightwave Technology. 33, 8, p. 1571-1577 7 p. , 2015.

[48] Bijan Rahimzadeh Rofoee, Georgios Zervas, Yan Yan, Markos Anastasopoulos, Anna Tzanakaki, Shuping Peng, Reza Nejabati, and Dimitra Simeonidou, "Hardware Virtualized Flexible Network for Wireless-DataCenter Optical Interconnects", JOCN, 2014.

[49] Rofoee, B. R., Zervas, G., Yan, Y., Amaya Gonzalez, N. & Simeonidou, D. "All Programmable and Synthetic Optical Network: Architecture and Implementation" , Journal of optical communications and networking. 5, 9, p. 1096-1110 15 p., 2013.

[50] Rahimzadeh Rofoee, B., Zervas, G., Yan, Y., Simeonidou, D., Bernini, G., Carrozzo, G., Ciulli, N., Levins, J., Basham, M., Dunne, J., Georgiades, M., Belovidov, A., Andreou, L., Sanchez, D., Aracil, J., Lopez, V. & Fernandez-Palacios, J. "Demonstration of low latency Intra/Inter Data-Centre heterogeneous optical Sub-wavelength network using extended GMPLS-PCE control-plane", Optics Express. 21, 5, p. 5463-5474, 2013.

[51] Rahimzadeh Rofoee, B., Zervas, G., Yan, Y., Amaya Gonzalez, N., Qin, Y. & Simeonidou, D. "Programmable on-chip and off-chip network architecture on demand for flexible optical intra-Datacenters", Optics Express. 21, 5, p. 5475-5480, 2013.

[52] G. M. Saridis, Y. Yan, Y.Shu, S. Yan, M. Arslan, T. Bradley, N. V. Wheeler, N.H.L. Wong, F. Poletti, M.N. Petrovich, D.J. Richardson, S. Poole, G. Zervas, D. Simeonidou "EVROS: All-Optical Programmable Disaggregated Data Centre Interconnect Utilizing Hollow-Core Bandgap Fibre" ECOC, 2015.

[53] Saridis, G.M., Peng, S. ,Yan, Y. , Aguado, A. , Guo, B. , Arslan, M. , Jackson, C. , Miao, W. , Calabretta, N. , Agraz, F. , Spadaro, S. , Bernini, G. , Ciulli, N. , Zervas, G. , Nejabati, R. , Simeonidou, D., "LIGHTNESS: A Deeply-Programmable SDN-enabled Data Centre Network with OCS/OPS Multicast/Unicast Switch-over" ECOC Postdeadline, 2014.

[54] S.Y. Yan, S. Peng, Y. Yan, B. R. Rofee, Y. Shu, E. Hugues Salas, G. Zervas, D. Simeonidou, L. Nishihara, R. Okabe, T. Tanaka, T. Takahara, J. C. Rasmussen, M. Svaluto Moreolo, J. Fabrega, L. Nadal, C. Kottke, M. Schlosser, Y. Yoshida, P.J. Argibay-Losada, K. Kitayama, F. Jimenez, and V. Lopez "100G Beyond Ethernet Transport for Inter- and Intra-DCN communication with Solutions and Optical Enabling Technologies in the ICT STRAUSS Project" EuCNC , Paris, 2015.

[55] G. Saridis, E. Hugues Salas, Y. Yan, S. Y. Yan, S. Poole, G. Zervas, D. Simeonidou "DORIOS: Demonstration of an All-Optical Distributed CPU, Memory, Storage Intra DCN Interconnect" OFC (2015)

[56] Bijan Rahimzadeh Rofoee, Kostas Katsalis, Yan Yan, Yi Shu, Thanasis Korakis, Leandros Tassiulas, Anna Tzanakaki, Georgios Zervas, Dimitra Simeonidou, "First Demonstration of Service-Differentiated Converged Optical Sub-Wavelength and LTE/WiFi Networks over GEANT", OFC, 2015.

[57] S. Y. Yan, Y. Yan, B. R. Rofoee, Y. Shu, E. Hugues Salas, G. Zervas, D. Simeonidou, "Demonstration of Real-Time Ethernet to Reconfigurable Superchannel Data Transport over Elastic Optical Network" , ECOC Cannes, France, 2014.

[58] S. Y. Yan, E. Hugues Salas, V. J. F. Rancaňo, Y. Shu, G. Saridis, B. R. Rofoee, Y. Yan, A. E. Peters, A. E., May-Smith, P. Petropoulos, D. Richardson, G. Zervas, D. Simeonidou, "First Demonstration of All-Optical Programmable SDM/TDM Intra Data Centre and WDM Inter-DCN Communication", ECOC. Cannes, France, 3 p. PD 1.2, 2014.

[59] Bijan Rahimzadeh Rofoee, Georgios Zervas, Yan Yan, Markos Anastasopoulos, Anna Tzanakaki, Shuping Peng, Reza Nejabati, and Dimitra Simeonidou, "Wireless-DataCenter Backhaul over Hardware Virtualized Flexible Optical Network", OFC, 2014.

[60] Y. Yoshida, A. Maruta, K. Kitayama, M. Nishihara, T. Tanaka, T. Takahara, J.C Rasmussen, N. Yoshikane, T. Tsuritani, I. Morita, S.Y. Yan, Y. Shu, M. Channegowda, Y. Yan, B. R. Rofoee, E. Hugues Salas, G. Saridis, G. Zervas , R. Nejabati, D. Simeonidou, "First international SDN-based Network Orchestration of Variable-capacity OPS over Programmable Flexi-grid EON", OFC Postdeadline 2014. San Francisco, 3 p. Th5A.2, 2014.

[61] Cisco, "Introduction to Segment Routing", Cisco IOS XE Release 3S,2020.

[62] Changhoon Kim, Parag Bhide, Ed Doe, Hugh Holbrook, Anoop Ghanwani, Dan Daly, Mukesh Hira, Bruce Davie, "In-band Network Telemetry (INT) ", https://www.netronome.com/documents/92/INT-current-spec.pdf , June 2016.

[63] Thomas F Herbert, "An Comparison of Fd.io and OVS/DPDK", DPDK User Space summit, 2015.

[64] Christoforos Kachris and Dimitrios Soudris, “A survey on recon gurable accelerators for cloud computing,” In Proceedings of the International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2016.

[65] NVIDIA MELLANOX BLUEFIELD-2 DPU, Nhttps://network.nvidia.com/files/doc-2020/pb-bluefield-2-dpu.pdf , 2022

[66] Henning Stubbe, “P4 Compiler & Interpreter: A Survey”, Seminars FI/IITM WS 16/17, Network Architectures and Services, May 2017.

[67] Han Wang, Robert Soulé, Huynh Tu Dang, Ki Suh Lee, Vishal Shrivastav, Nate Foster, Hakim Weatherspoon, “P4FPGA: A Rapid Prototyping Framework for P4,” The Symposium on SDN Research, 2017.

[68] P4FPGA Project,“P4 Bluespec Compiler”, https://github.com/hanw/p4fpga, 2016.

[69] Sandor Laki, “High-Speed Forwarding: A P4 Compiler with a Hardware Abstraction Library for Intel DPDK”, http://p4.elte.hu/publications/p4-ws-2016.pdf , 2016.

[70] Mihai Budiu, Chris Doss,”The architecture of the P4-16 compiler”, P4 workshop, Stanford, 2017.

[71] Jose Rolim, “Parallel and Distributed Processing: 11th IPPS/SPDP'99 Workshops Held in Conjunction with the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing,” San Juan, Puerto Rico, UxSA, April 12-16, 1999.

[72] “Ternary Content Addressable Memory (TCAM) Search IP for SDNet” Xilinx, 2017.

[73] Dinigroup, “TCP Offload Engine IP-128 sessions (TOE128)”, https://www.dinigroup.com/web/TOE128.php, 2020.

[74] Intel, “Intel 82599 10GbE Controller Datasheet”,2020.

[75] D. Lebrun, S. Previdi, C. Filsfils, and O. Bonaventure, “Design and Implementation of IPv6 Segment Routing,” Tech. Rep., 2016.

[76] Openstack, “Open vSwitch hardware offloading”, https://docs.openstack.org/neutron/queens/admin/config-ovs-offload.html, 2020.

[77] Nvidea, “An introduction to smartNICs and their benefits”, https://developer.nvidia.com/blog/choosing-the-best-dpu-based-smartnic/ , 2021.

[78] Doyle Research, “Choosing the Best SmartNIC”, https://www.techtarget.com/searchnetworking/tip/An-introduction-to-smart-NICs-and-their-benefits, 2023.

[79] Miano, Sebastiano, et al. "Introducing smartnics in server-based data plane processing: The ddos mitigation use case." IEEE Access 7 (2019): 107161-107170.