Technische Universität Dresden

Ultra-reliable Low-latency, Energy-efficient and Computing-centric Software Data Plane for Network Softwarization

Dipl.-Ing.
Zuo Xiang

der Fakultät Elektrotechnik und Informationstechnik der Technischen Universität Dresden

zur Erlangung des akademischen Grades

Doktoringenieur
(Dr.-Ing.)

genehmigte Dissertation

Vorsitzender: Prof. Dr.-Ing. habil. Leon Urbas
Gutachter: Prof. Dr.-Ing. Dr. h. c. Frank Fitzek
Gutachter: Prof. Dr. Thorsten Strufe
Gutachter: Prof. Dr.-Ing. Hans D. Schotten

Tag der Einreichung: 21.03.2022
Tag der Verteidigung: 19.08.2022

Abstract

Network softwarization plays a significantly important role in the development and deployment of the latest communication system for 5G and beyond. A more flexible and intelligent network architecture can be enabled to provide support for agile network management, rapid launch of innovative network services with much reduction in Capital Expense (CAPEX) and Operating Expense (OPEX). Despite these benefits, 5G system also raises unprecedented challenges as emerging machine-to-machine and human-to-machine communication use cases require Ultra-Reliable Low Latency Communication (URLLC). According to empirical measurements performed by the author of this dissertation on a practical testbed, State of the Art (STOA) technologies and systems are not able to achieve the one millisecond end-to-end latency requirement of the 5G standard on Commercial Off-The-Shelf (COTS) servers. This dissertation performs a comprehensive introduction to three innovative approaches that can be used to improve different aspects of the current software-driven network data plane. All three approaches are carefully designed, professionally implemented and rigorously evaluated. According to the measurement results, these novel approaches put forward the research in the design and implementation of ultra-reliable low-latency, energy-efficient and computing-first software data plane for 5G communication system and beyond.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Prof. Dr.-Ing. Dr. h.c. Frank H. P. Fitzek in providing me the best supervision, financial support and working environments in order to support me completing my Ph.D. research. Without his enlightenment, I would never have been able to finish my Ph.D. research.

I would like to thank Prof. Martin Reisslein for his invaluable help in the discussing, editing and revising of all my journal papers. Every successful publication of my journal paper is due to his careful and meticulous help.

Two of the supervisors that I would like to express my appreciation is Dipl.-Inf. Frank Gabriel and Dr. Giang T. Nguyen. While Dipl.-Inf. Frank Gabriel spends a lot of time helping me to solve the problems in the system implementation, Dr. Giang T. Nguyen provides me many insightful comments on the writing of my publications. I would not get these publications without their selfless and patient help.

I would like to thank all my colleagues at the Deutsche Telekom Chair of Communication Networks, especially Dipl.-Ing. Justus Rischke, Dipl.-Ing. Malte Höweler, M.Eng. Vu Nguyen and M.Sc. Alexander Kropp, for their help in my research work as well as the study life at TU Dresden, Germany.

I would like to express my deep gratitude to my family and friends for their encouragement and support. I would especially like to thank my father Jinyuan Xiang and mother Yanmei Zheng for the emotional and financial support they have provided. Finally, I would like to thank my wife, Xuefan Wang, for her greatest and meticulous care for my life. Without her encouragement and concern, I would have given up pursuing this Ph.D. research in the second year. This dissertation could not have been completed without her love.

Statement of authorship

I hereby certify that I have authored this document entitled *Ultra-reliable Low-latency, Energy-efficient and Computing-centric Software Data Plane for Network Softwarization* independently and without undue assistance from third parties. No other than the resources and references indicated in this document have been used. I have marked both literal and accordingly adopted quotations as such. There were no additional persons involved in the intellectual preparation of the present document. I am aware that violations of this declaration may lead to subsequent withdrawal of the academic degree.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Research Motivation

In order to understand the 5G, namely the fifth-generation of cellular communication networks and motivations of this dissertation, it is significantly important to review and understand how communication networks have evolved over time [1]. At the time of this work, communication systems have already evolved to its fourth generation and the emerging 5G system is still open for research and exploring even with some early real-world implementations and deployments. For the very first commercial and widely deployed communication network, namely the telephone network, to the largest global Internet network system, the communication network has evolved from conventional paradigm of circuit switching to the novel, simple, robust and low cost paradigm of packet switching, which is currently mainly based on two fundamental protocols: Internet Protocol (IP) and Transmission Control Protocol (TCP) and one simple policy of *store and forward*. The packet-switched approach used by Internet has already thrived over the years, while the cellular communication systems began as a wireless extension to the Public Switched Telephone Network (PSTN), which focuses only on voice services. The cellular networks can be connected to Internet through the IP protocol.

In last 40 years, the cellular communication system has already evolved from 1G to 4G. While the 1st Generation (1G) of cellular communication network system was based on analog technologies, the 4th Generation (4G) cellular network system already provides full support of IP-based mobile Internet.

Compared to conventional hardware-based implementations, software firstly began to play a significantly important role in the 4G cellular system. In contrast to the conventional hardware-driven network architecture of 1G to 3G, network softwarization is able to realize a much more flexible network architecture which provides support for agile network management, rapid launch of novel and innovative services with much reduction in Capital Expense (CAPEX) as well as Operating Expense (OPEX).

This work comes at a time when the latest 5G cellular network systems are being implemented and also deployed in some countries. According to the analysis per-

formed by Fitzek *et.al.* in [1], compared to previous evolutions, namely 1G to 4G, 5G seems to be a real revolution: (i) Besides humans, 5G also aims at providing services to billions of end devices, namely the so-called Internet of Things (IoT) concept. Some 5G targeted use cases require the support of ultra-reliable low-latency and real-time communication for both data and control messages [1]. (ii) Instead of focusing only in the wireless domain, the 5G cellular system should be a holistic design with joint efforts in both wireless and wired domains. Standardization entities in both domains should be involved for 5G systems, including 3rd Generation Partnership Project (3GPP) and Internet Engineering Task Force (IETF) [1]. The overview of cellular network evolution and the holistic design of 5G is graphically illustrated in Figure 1.1.

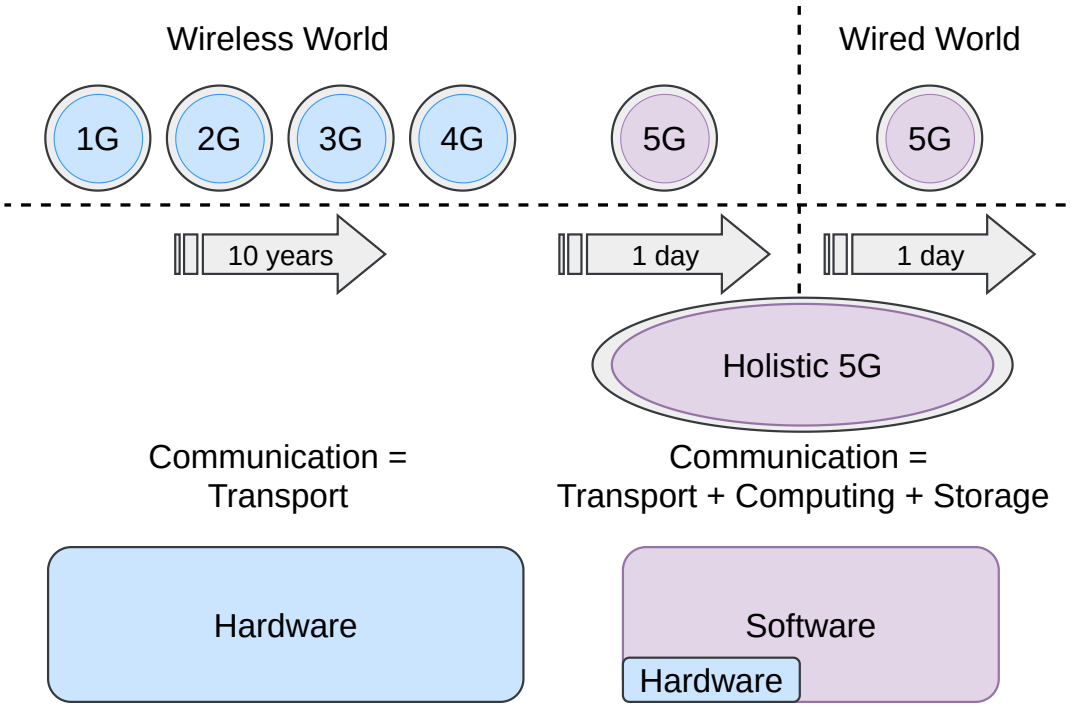


Figure 1.1: The evolution of communication architecture. Reprinted and adapted from the book [1].

As demonstrated in the Figure 1.1, generations before 5G are dominantly hardware-driven, which have relative long innovation and update cycle (i.e. it takes several years) mainly due to the high cost of hardware modification. On the other hand, 5G cellular system are starting to adopt some best practices of IETF approach in the wired domain, which are solely driven by software solutions. Therefore, network softwarization is one of the most important concepts in 5G system which enables innovations and improvements from Radio Access Network (RAN) to mobile core network in a cellular communication system. With this trend of network softwarization for 5G, the conventional network based on the fundamental *store and forward* paradigm (i.e. treat network as a "dumb pipe") is transformed into a new and novel network based on *compute and forward* paradigm, where the data is also stored and processed directly in the network. This innovative and computing-centric *compute and forward* paradigm provided by 5G network system is explored in this work to

improve the latency performance and energy efficiency.

To further understand the latency performance and energy efficiency challenges for the 5G network softwarization, which is the motivation and focus of this dissertation, a more detailed introduction of the holistic 5G communication system should be performed here with the exquisite *5G Atom* concept proposed by Fitzek *et.al* in [1]. The 5G atom is graphically illustrated in Figure 1.2. At the heart of the 5G atom are important use cases that drive the development of 5G. In order to support these use cases, multiple technical and performance requirements built the first tier of the atom, where the latency and energy are given high attention in this dissertation. In order to address these requirements, several novel concepts for 5G are listed in the second tier of atom, including network slicing, Multi-access Edge Computing (MEC) and Information-centric Networking (ICN) etc. Then the third tier consists of promising network softwarization technologies which can enable the realization of the concepts presented in the second tier. This tier is the most significant tier for this dissertation, because the core motivation and contribution of this dissertation is to improve these technologies in three different perspectives, especially those used in the network data plane to improve the latency performance, energy efficiency and intelligence of the State of the Art (STOA) 5G system. The last tier of the atom consists of some representative sample innovative mechanisms or approaches that can be better utilized on 5G system due to the flexibility and programmability provided by the concept of network softwarization. With respect to the emerging network softwarization technologies, two of them are significantly important and highly explored and utilized in this dissertation:

- Network Function Virtualization (NFV): NFV is a novel network architecture concept and technology that utilizes the virtualization technologies provided by Information Technology (IT) domain to virtualize conventional hardware-driven Network Functions (NFs) into software-driven Virtualized Network Functions (VNFs) or Cloud-native Network Function (CNF) that can be deployed and orchestrated on Commercial Off-The-Shelf (COTS) hardware to deliver communication services [2].

- Software-Defined Networking (SDN): SDN technology is a novel mechanism for network management that enables flexible, dynamic and highly programmable network configuration [3]. In SDN, the control plane of network nodes are decoupled from their data plane, which enables control and orchestration of the network from a centralized entity, namely the SDN controller [3].

After the introduction of the most important network softwarization techniques, the stringent real-time latency requirements of 5G need to be further discussed here to better understand why low latency is an unprecedented challenge for current 5G and future communication systems, and why several of the research works included in this dissertation prioritize latency performance.

Communication networks have long been designed, implemented and evaluated only for high bandwidth or throughput performance. This is mainly due to the well-known fact that most typical network-based applications require mainly sufficient

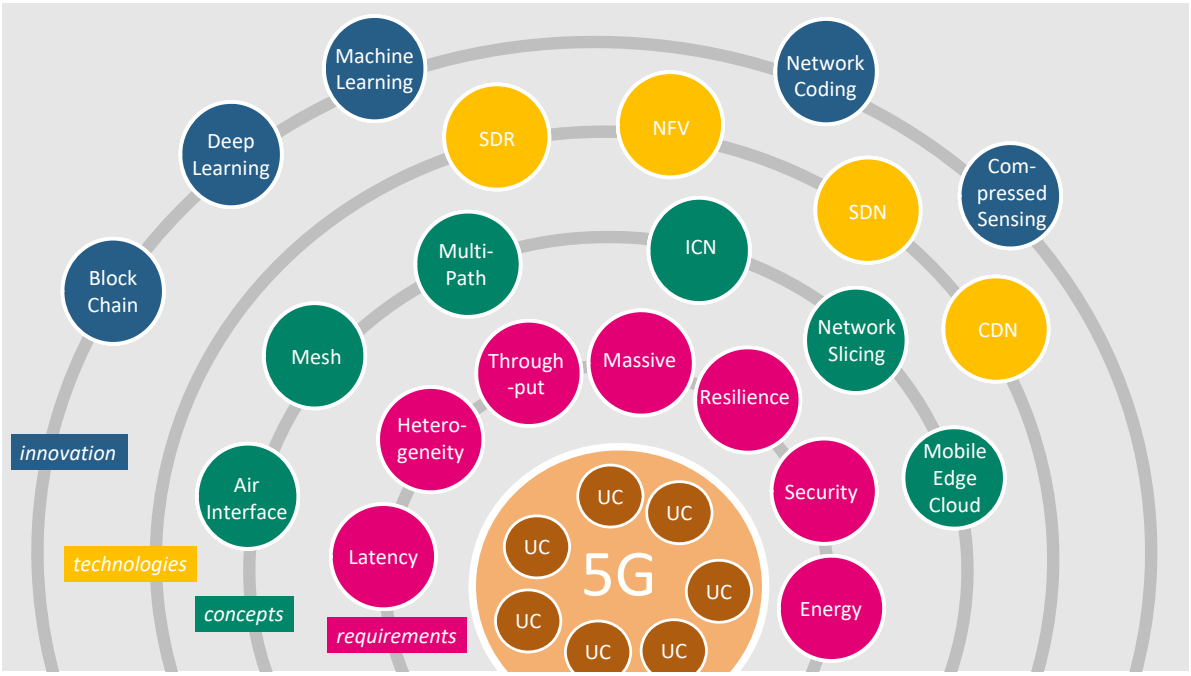


Figure 1.2: The 5G Atom. From the inside out, the first tier presents performance requirements. The second tier presents the novel concepts. The third tier collects emerging and enabling softwarization technologies. The last tier lists the unique innovations. Reprinted from the book [1].

bandwidth, such as web browsing, file transmission or on-demand video streaming. Compared to bandwidth improvements, latency and jitter performance optimization has been mainly ignored in main network research for a long time, as described by Cheshire *et.al.* in already in 1996 [5]. However, in contrast to previous four generations, the latency and jitter performance plays a dominantly important role in 5G system and is listed on the top of the all essential requirements [1]. The main reason of this change is based on the design that 5G and future communication network system targets at the emerging use cases with **integrated control loop**, including Machine to Machine (M2M) and human-to-machine communication [1]. Therefore, end-to-end latency and jitter performance has to be guaranteed for these use cases.

According to the 5G standard [6] published by 3GPP, the allowed end-to-end communication latency for ultra-reliable low-latency use cases is limited to only one millisecond (1 ms). Based on this requirement, a typical latency budget for major 5G components involved in a sensor/actuator control loop is illustrated in the Figure 1.3 [4]. In this latency budget, a total of 0.4 ms is consumed by the embedded systems (sensors and actuators) and the wireless network systems. With the 1 ms end-to-

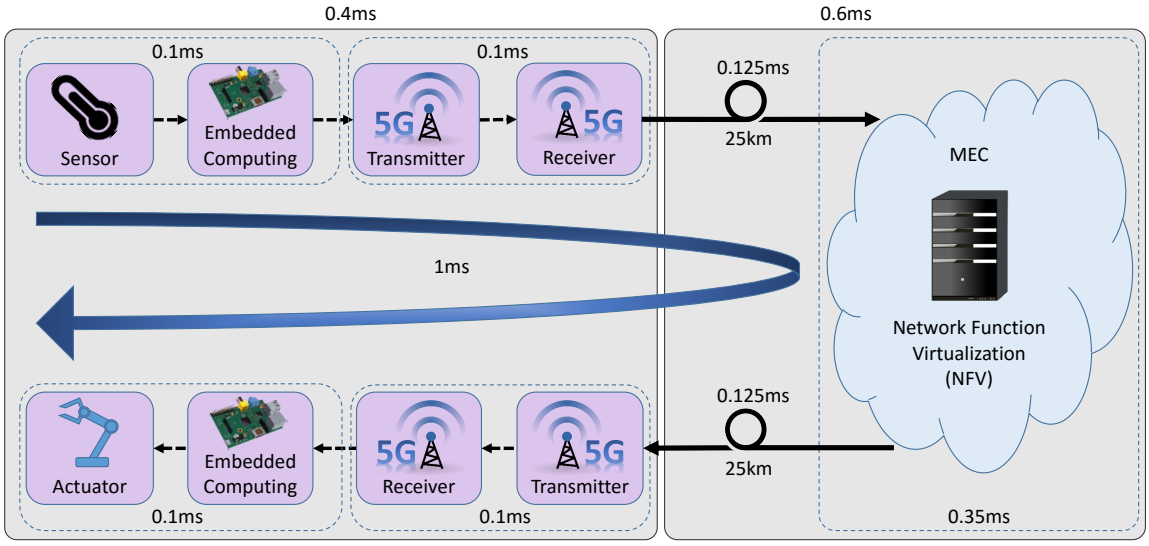


Figure 1.3: Typical latency budget for the sensor-to-actuator remote control loop which meets the 1 ms end-to-end latency requirement of 5G. Reprinted from **my journal paper** [4].

end latency requirement, the wired domain is left only with 0.6 ms. The wire domain consists of two main components. While one latency component is the wired communication over the fiber where the latency is limited by the speed of the light (namely, about 3.34 μs per kilometer), the second component is based on the network and computing nodes in the wired domain. As introduced above, 5G network system utilizes network softwarization technologies to support the *compute and forward* paradigm. These novel softwarization technologies enable the concept of Multi-access Edge Computing (MEC), which enables local network data processing. Assuming the maximum length of the fiber used in the system is 25 km and the speed of the fiber used is 2000000 km/s, 0.25 ms latency is required for this component. Finally, only 0.35 ms, namely 35% of the millisecond, remains for the MEC system.

However, according to my rigorous measurements on State of the Art (STOA) MEC testbed based on OpenStack, which is described in detail in Chapter 2, VNFs implemented with traditional socket Application Programming Interfaces (APIs) provide by the Linux kernel are only able to provide an end-to-end latency performance of several milliseconds even with the simplest elementary forwarding network function. Therefore, in summary, this dissertation aims to address the question: *How to significantly reduce the latency of State of the Art (STOA) softwarization network data plane to meet the 5G stringent end-to-end latency requirement of one millisecond with minimal negative impact or even improvement on other important performance metrics, especially bandwidth and energy consumption?*

1.2 Main Contributions

In order to address the research question introduced in Section 1.1, three innovative approaches published as journals (all as the first author) are concluded in this dissertation to improve the high-performance software data plane in three different directions. These three directions or perspectives are illustrated as a Triforce in Figure 1.4.

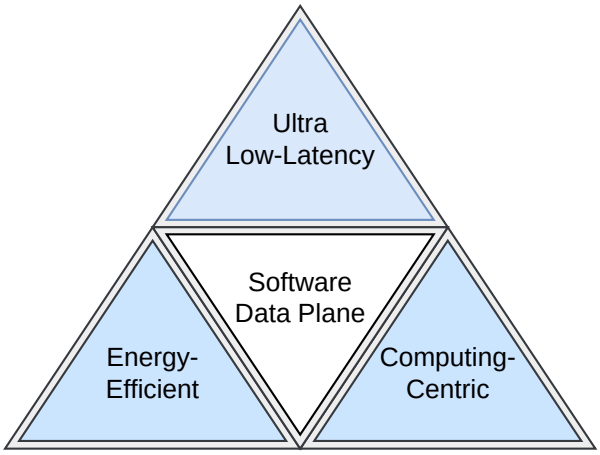


Figure 1.4: Main contributions of this dissertation.

1. Ultra-reliable low-latency: A novel and practical framework called Chain bAsed Low latency VNF ImplemeNtation (CALVIN) was designed, implemented and evaluated in **my journal** [4] to achieve an end-to-end RTT performance on the order of only 0.32 ms on the Commercial Off-The-Shelf (COTS) Multi-access Edge Computing (MEC) platform.

2. Energy-efficient: In **my journal** [7], the novel XDP-Monitoring energy-Adaptive Network functions (X-MAN) framework was designed and implemented to enable non-intrusive and fine-grained traffic workload monitoring and per-CPU core frequency management for energy saving of high-performance softwarization data plane systems. According to extensive measurements and evaluations of the proposed X-MAN system on a physical testbed with support for 10 Gbps Ethernet, X-MAN is able to support two important performance metrics: (i) X-MAN can consistently monitor the workload traffic for four data plane network interfaces with a latency of only 10 μs, while the STOA Hardware Counter (HC) approach [8] requires a latency ranging from 20 to even 80 μs (ii) For the random traffic model described in Section 4.4, X-MAN is able to reduce the energy consumption to less than half of the STOA Code Instruction with Heuristic power management (CIH) approach [9], while has negligible impact on latency performance.

3. Computing-centric: A novel approach named as You Only Look Once, but Compute Twice (YOLO-CT) is proposed in **my journal** [10] which utilizes the

COmputing In Network (COIN) paradigm supported by the softwarized network to significantly reduce the amount of data required to be sent through the network by offloading part of the Convolutional Neural Network (CNN) model directly into the network nodes with computing power and functionalities.

List of Publications

Journals

- Zuo Xiang, Frank Gabriel, Elena Urbano, Giang T. Nguyen, Martin Reisslein, and Frank HP Fitzek. "Reducing latency in virtual machines: Enabling tactile Internet for human-machine co-working." IEEE Journal on Selected Areas in Communications 37, no. 5 (2019): 1098-1116.
- Zuo Xiang, Patrick Seeling, and Frank HP Fitzek. "You only look once, but compute twice: Service function chaining for low-latency object detection in softwarized networks." Applied Sciences 11, no. 5 (2021): 2177.
- Zuo Xiang, Sreekrishna Pandi, Juan Cabrera, Fabrizio Granelli, Patrick Seeling, and Frank HP Fitzek. "An open source testbed for virtualized communication networks." IEEE Communications Magazine 59, no. 2 (2021): 77-83.
- Huanzhuo Wu, Zuo Xiang, Giang T. Nguyen, Yunbin Shen, and Frank HP Fitzek. "Computing Meets Network: COIN-Aware Offloading for Data-Intensive Blind Source Separation." IEEE Network 35, no. 5 (2021): 21-27.
- Zuo Xiang, Malte Höweler, Dongho You, Martin Reisslein, and Frank HP Fitzek. "X-MAN: A Non-intrusive Power Manager for Energy-adaptive Cloud-native Network Functions." IEEE Transactions on Network and Service Management (2021).

Conferences

- Zuo Xiang, Frank Gabriel, Giang T. Nguyen, and Frank HP Fitzek. "Latency measurement of service function chaining on OpenStack platform." In 2018 IEEE 43rd Conference on Local Computer Networks (LCN), pp. 473-476. IEEE, 2018.
- Huanzhuo Wu, Jia He, Máté Tömösközi, Zuo Xiang, and Frank HP Fitzek. "In-Network Processing for Low-Latency Industrial Anomaly Detection in Softwarized Networks." In 2021 IEEE Global Communications Conference (GLOBECOM), (has been accepted, not official published yet), IEEE, 2021.

Book Chapters

- Zuo Xiang, Sreekrishna Pandi, Patrick Seeling, and Frank HP Fitzek. "ComNetsEmu: a lightweight emulator." In Computing in Communication Networks, pp. 245-256. Academic Press, 2020.

- Zuo Xiang, Renbing Zhang, and Patrick Seeling. "Machine learning for object detection." In Computing in Communication Networks, pp. 325-338. Academic Press, 2020.
- Zuo Xiang, and Patrick Seeling. "Mininet: an instant virtual network on your computer." In Computing in Communication Networks, pp. 219-230. Academic Press, 2020.
- Zuo Xiang, Carl Collmann, and Patrick Seeling. "Realizing mobile edge clouds." In Computing in Communication Networks, pp. 277-287. Academic Press, 2020.
- Justus Rischke, and Zuo Xiang. "Network coding for transport." In Computing in Communication Networks, pp. 339-349. Academic Press, 2020.

1.3 Dissertation Organization

This dissertation is organized into seven chapters:

Chapter 1: This chapter presents the research motivation, innovative contributions and the outline of this dissertation.

Chapter 2: This chapter describes my research work on the topic "Latency Measurement of Service Function Chaining on OpenStack Platform" published in [11]. The analysis and measurement evaluations performed in this work highly motivate other works included in this dissertation.

Chapter 3: This chapter presents my research work on the ultra-reliable low-latency perspective of the STOA high-performance software network data plane. The design, implementation and rigorous evaluation of the proposed CALVIN system is comprehensively described in this chapter.

Chapter 4: This chapter describes my research work on the energy-efficient perspective of the STOA high-performance software network data plane. The design, implementation and rigorous evaluation of the proposed X-MAN system is comprehensively described in this chapter.

Chapter 5: This chapter presents my research work on the design and implementation of the novel Communication Networks Emulator (ComNetsEmu), which can be used to simply prototype and evaluate research ideas for network softwarization systems.

Chapter 6: This chapter describes my research work on the computing-centric perspective of the STOA high-performance software network data plane. The design, implementation and rigorous evaluation of the proposed YOLO-CT system is comprehensively described in this chapter.

Chapter 7: This chapter concludes this dissertation by summarizing the research works done with the scope of the introduced research topic. Main contributions of this dissertation is also summarized.

2 Latency Measurement of Service Function Chaining on OpenStack Platform

All contents in this chapter have already been published in **my conference paper** [11]: "Latency measurement of service function chaining on OpenStack platform." In 2018 IEEE 43rd Conference on Local Computer Networks (LCN), pp. 473-476. IEEE, 2018.

2.1 Introduction

In recent years, with the rapid growth of users and network traffic, how to effectively use limited resources to provide services that can meet certain requirements has become an important issue for network operators. The capabilities of modern communication networks are extended by NFV and SDN to address this challenge. The limitation in legacy systems is resolved with NFV by removing the tight integration of middleboxes running NFs from their proprietary hardware platform. NFs are virtualized and implemented with advanced software technologies into flexible VNFs.

Since the actual use case usually requires a complete and complex NF composed of several smaller components called SFs, NFV systems must be able to force packets to traverse them in a specific order. The mechanism for automatically and efficiently instantiating an ordered chain of VNFs and subsequently redirecting traffic through the chain is known as SFC [12]. One of the most critical challenges is to decide on which worker node to launch SFs to satisfy multiple performance objectives, i.e. the problem of the placement of SFs. A large body of research work in the literature has investigated the optimization issue of efficiently placing SFs physical hosts [13, 14]. However, at the time I was researching this topic, to the best of my knowledge, only a few of them were implemented on real-world cloud platforms. In addition, the lack of latency measurements on practical cloud platforms is an open issue.

In order to understand latencies of SFCs on the OpenStack cloud platform and their causes, a measurement campaign is performed in this work. Instead of the de-

fault round-robing placement algorithm, several heuristic algorithms are designed and implemented to minimize SFC latency. Implementations are evaluated with rigorous measurements on the OpenStack platform. Based on the measurement results, the SFC latency performance of STOA technologies on the OpenStack platform is analyzed.

This work elucidates several potential directions for improving latency performance on real-world cloud platforms based on actual measurements. These analyses provide measurement and practical support for further work in this dissertation.

2.2 Background and Related Work

Medhat *et al.* [15] perform a comprehensive survey of STOA SFC frameworks and practical implementations. A service-oriented SDN controller is proposed in [16] to implement dynamic SFCs within the framework of a service overlay network. However, the impact of packet processing is not considered in this work. Zhang *et al.* [17] presents the `StEERING` SFC framework for flexible traffic management using SDN technologies. A heuristic algorithm is also designed to reduce the service latency of deployed SFCs. The latency performance is performed to compare the proposed algorithm with a random placement. Cloud4NFV framework is introduced in [18] to orchestrate SFs in a distributed telco cloud environment. Although the proposed framework supports the flexible management of SFs through SFC, detailed latency performance measurements are not performed in this work.

In summary, the placement problem of SFC related research needs to be further simplified on practical cloud platform. Furthermore, it is important to understand different types of latencies at each step of the SFC deployment process to validate assumptions of theoretical modelling and shed light on ares for further improving the latency of SFC on the practical OpenStack platform.

2.3 SFC-OStack Framework

At the time this work researched on this topic, OpenStack still had very limited support for SFC. The official SFC extension for OpenStack networking only provide basic APIs to build SFC in Neutron without support for APIs to implement different placement algorithms. In order to simplify the development, management and evaluation of SFC placement algorithms, research oriented framework called Service Function Chaining on OpenStack (SFC-OStack) [19] is developed on the OpenStack platform. It is built on top of the official SFC extension (called `networking-sfc`) of OpenStack.

The architecture and main components of the SFC-OStack framework is illustrated in Figure 2.1. The SFC-OStack orchestrator makes use of services provided by OpenStack to render SFCs into an ordered set of virtual compute instances and corresponding port chains in the network data plane. The SFC-OStack orchestrator contains SFC manager, scheduler and monitor modules.

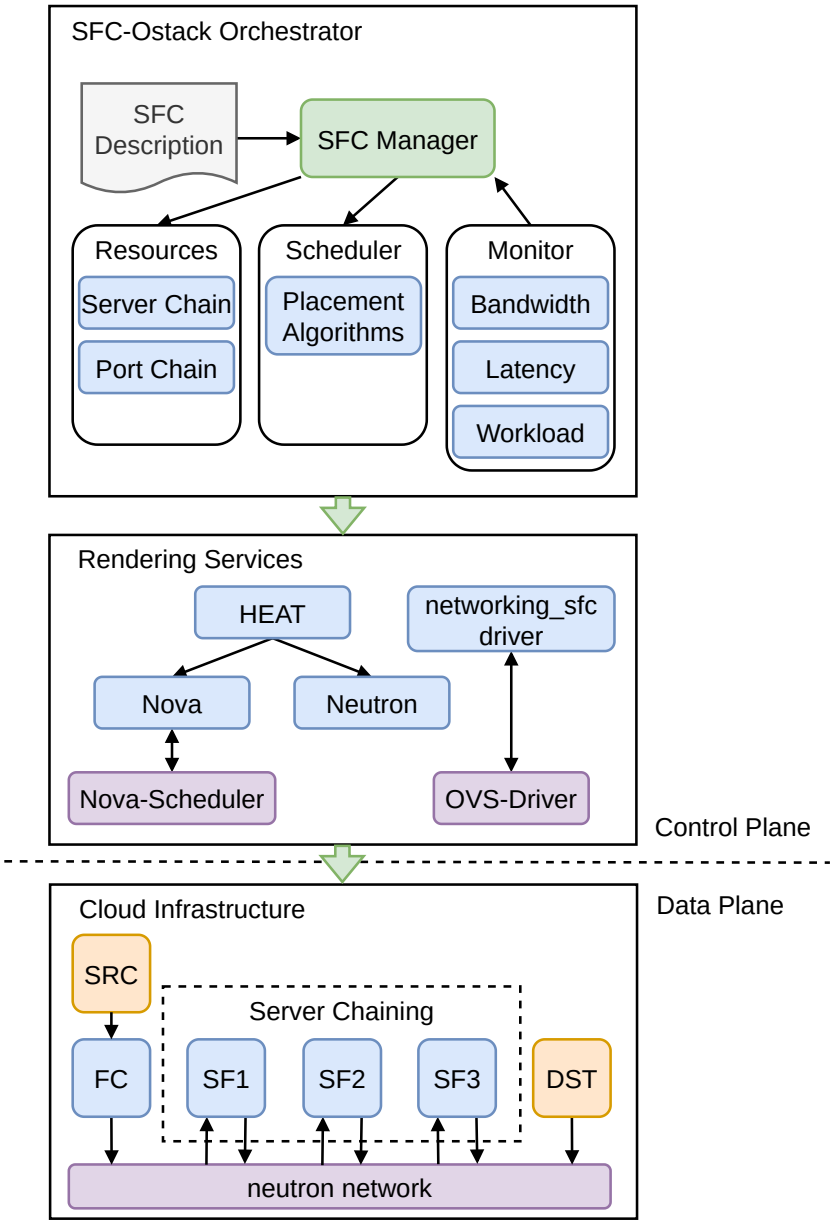


Figure 2.1: SFC-OStack architecture and its main components in control and data plane.

The SFC manager is responsible for Create, Read, Update and Delete (CRUD) operations and life-cycle management of SFCs. It models the SFC description into resources that are available on OpenStack and its SFC extension. Available OpenStack resources include: (i) Server chain: A set of ordered virtual compute instances on which NF programs are dispatched and executed. Virtual compute instances are grouped into server groups. These groups can be used for the load balancing feature provided by port pair groups of `networking-sfc` extension. Besides compute instances, associated Neutron ports are also created and bounded. These server instances can be launched by all virtualization technologies supported by OpenStack, including bare metal servers, VMs and system containers. In this work, VMs are used

with the default Kernel-based Virtual Machine (KVM) hypervisor. (ii) Port chain (PC): Port chain is a wrapper resource required for traffic steering in a SFC. Each port chain consists of a Flow Classifier (FC) and a list of port pair groups. Port chains can be backed by different networking service providers for service path rendering. In OpenStack version Pike, Open vSwitch (OVS) driver is used as the default service provider. The server chain and port chain are implemented as separate resources in this framework to provide efficient resource allocation and flexible updating of deployed SFCs.

After the orchestrator converts the specification into native OpenStack resources, rendering services are used to build the actual SFCs on the cloud infrastructure. Because the official OpenStack Heat service does not manage `networking-sfc` resources in OpenStack Pike version, SFC-OStack provides its own module to convert customized SFC description files into Heat template files.

In order to deploy and render SFCs on OpenStack Neutron network architecture, the network configuration of compute instances needs to be carefully designed. In SFC-OStack, while the orchestrator is deployed on the Neutron networking node and connected to the external network, SFCs are rendered on multiple compute nodes that are connected to the management and data networks via separate physical interfaces. Each VM in the chain is allocated with three separate virtual interfaces. As two interfaces are used as ingress and egress ports for user plane data traffic, the last interface is for management traffic of the SF. From the view of each VM, all its network interface are attached to a private virtual tenant network. They get internal IP addresses from Neutron's Dynamic Host Configuration Protocol (DHCP) agents and are allocated with a floating IP address for remote access and management from public network.

2.4 Latency-aware Network Function Placement and Chaining

The problem researched by this work is the latency-aware placement and chaining of SFs. All SFs of a logical SFC need to be deployed and then interconnected on the physical cloud infrastructure with multiple resource constraints such as compute, network and storage. The goal of the proposed algorithms is to find a placement that minimizes both rendering and service latency of SFC at the same time. Three simple heuristic algorithms are proposed in this chapter as low-complexity solutions to the SFC placement problem.

In practical deployment, both computational and network aspects need to be considered in order to design heuristic placement algorithms. Therefore, following assumptions are made in this work: (i) Inter-node physical network connections are more costly and time-consuming than intra-node virtual network connections. In contrast to intra-node traffic that can be forwarded directly by local software virtual switch, inter-node traffic requires additional tunnel encapsulation and transmission over the underlying physical network. (ii) Virtual instances and port chains allocated on the same physical compute node share computing and networking re-

sources. When too many SFs are allocated on the same compute node, this compute node can become overwhelmed, which significantly increases computation and network latency. Therefore, two types of latency are considered in following placement strategies: (i) Processing latency: Computational latency required by SFs for packet processing. (ii) Transmission latency: Network latency required to transmit packets between SFs and underlying physical compute nodes if needed.

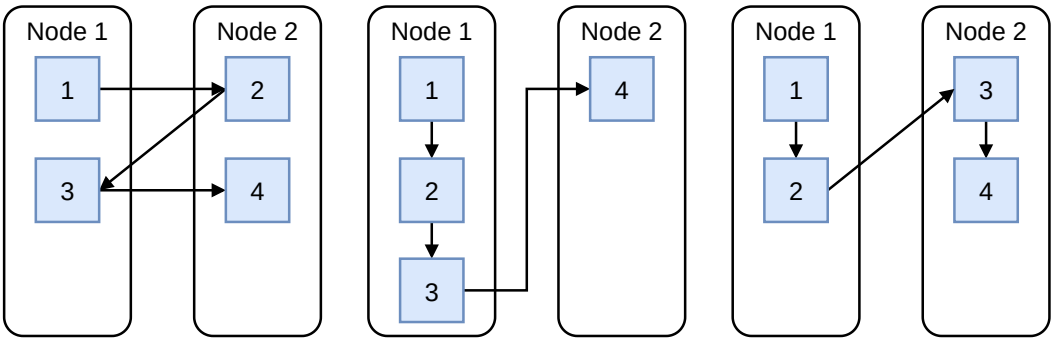


Figure 2.2: An example of three implemented heuristic algorithms for SFC placement: Four SFs are placed on two physical compute nodes. Reprinted and adapted from **my conference paper** [11].

Following heuristic approaches are designed and also implemented: (i) Load Balancing (LB) strategy: This is simply a greedy algorithm that tries to minimize the processing latency without considering the transmission latency. Compute node with minimal computational workload is always selected in each iteration of the SF placement. (ii) Least Connection (LC) strategy: In this approach, the transmission latency is minimized without considering processing latency. Connections between different nodes are minimized. The physical compute node running the service application is firstly used to place new SFs. Then as many SFs as possible are placed on this selected node. When the resources of the node are exhausted, the next node is selected randomly for further placements. (iii) Load Balancing plus Least Connection (LBLC) strategy: This is an optimized version of LB that also takes transmission latency into consideration. LBLC uses the same mechanism of LB to launch SFs. However, a reordering of SFs is performed before chaining them. Reordering here means rearranging the mapping between the virtual SFs in the logical SFC and the VMs spawned on compute nodes. The overhead of this remapping process of virtual SFs is much cheaper than firstly deploy the NF programs and then migrate VMs. The reordering algorithm minimizes connections between different nodes to reduce transmission latency as much as possible. An example using aforementioned three strategies to place four SFs on two compute nodes is presented in Figure 2.2.

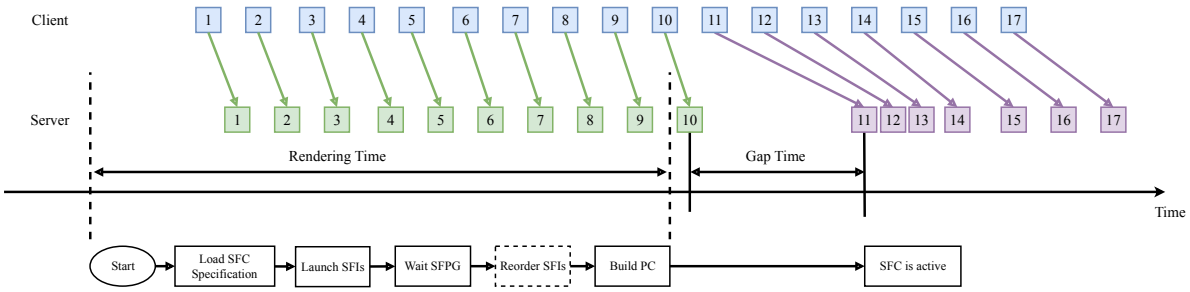


Figure 2.3: SFC startup and service processes. Reprinted from **my conference paper** [11].

2.5 Measurement Campaign and Results Evaluation

Based on analyses of the system, the overall SFC latency contains mainly three different types i.e. rendering latency, gap latency and One-Way Delay (OWD). These delays are graphically illustrated in Figure 2.3. These delays can be used to completely estimate the SFC setup process, from instantiation of a SF chain until the SFC is completely ready for service. These delays are explained as follows: (i) Rendering latency: The duration it takes to instantiate a SF chain with both compute and network resources. (ii) Gap latency: It describes the delay of the first redirected packet to reach its destination through the allocated chain. This gap duration exists because the applied traffic policies and security rules only take effect when the first payload packet arrives at the first payload packet arrival. This delay only happens once for each newly allocated SFC. (iii) OWD: This delay refers to the average delay of packets passing through the allocated SFCs after they are ready for service, i.e. after the rendering process in fully completed and the first packet successfully pass through the chain.

In order to perform the measurement campaign of aforementioned latencies, a combination of both passive and active measurement strategies is applied in this work. The passive strategy is used to measure the rendering delay of the SFC-OStack orchestrator, while the active strategy is used to measure the gap delay and OWD. User Datagram Protocol (UDP) traffic with a fixed packet size is used to send probing traffic in order to have complete control over the probing process without interference from flow and congestion control mechanisms of TCP. In order to measure OWD with high accuracy, clocks of all physical machines in the testbed used in this work are synchronized with Network Time Protocol (NTP).

The UDP probing client marks packet IDs and timestamps for all packets in the probing traffic to measure OWD. In order to measure the gap latency, the probing server has to be able to distinguish between packets with and without SFC processing. Therefore, SFs in the chain modify the original UDP payload.

In order to measure the OWD, sample VNFs need to be launched for packet processing. The processing performance of VNFs is not the focus of this work, so the corresponding overhead should be minimized in all measurements. Only minimal functionalities of VNFs are deployed to make the latency measurements independent of the complex packet processing of the NF itself. The minimal packet pro-

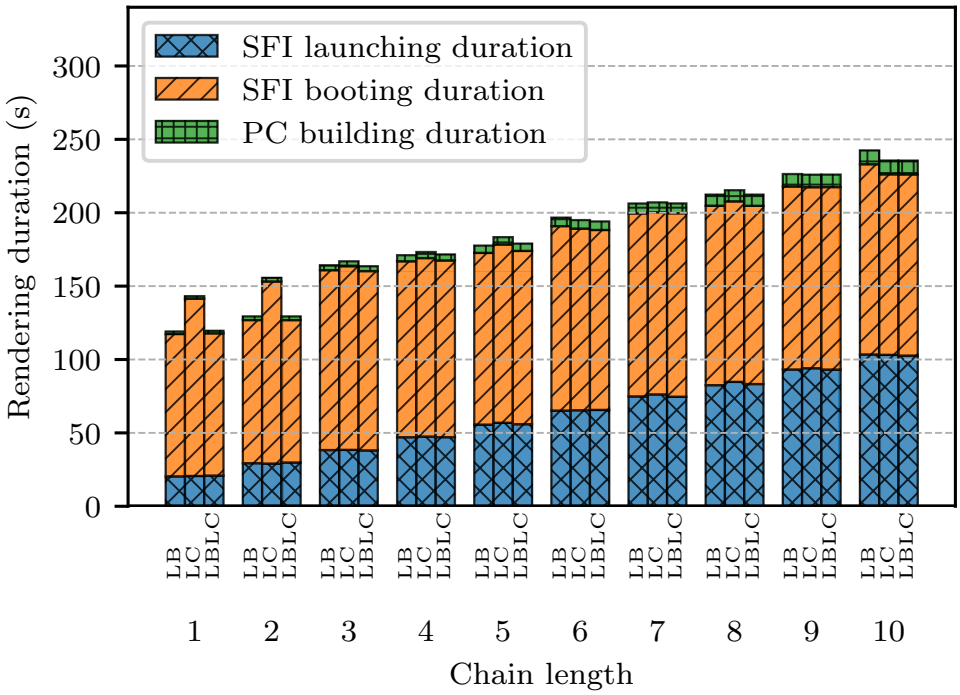
cessing programs launched on VMs are implemented both in Linux kernel and user spaces. At the time of this work, Linux Kernel IP Forwarding (LKF) is launched since it's one of the fastest packet processing provide by Linux kernel. It is chosen in this work as the baseline to evaluate the performance of other NF implementations. Besides LKF, a forwarding function in user space is implemented using Python (PyF) and the Linux packet socket (AF_Packet) API. For all active probing, UDP traffic with a probing interval of four milliseconds and a UDP payload size of 512 bytes is used.

Evaluation measurements are performed on a practical and physical testbed with four COTS servers. Each compute node has four CPU cores (Intel 4^{th} Generation Core i5), 16 GB DDR3 RAM and 128 GB SSD disk. The OpenStack version Pike is deployed and configured on Ubuntu 16.04 Operating System (OS). SFCs with different chain length are created between the probing client and server for latency measurements. For each specific measurement setup, measurements are repeated for 30 times for mean values and 99.9% confidence intervals. All measurement results and figures have been already published in my conference paper [11]. Figures are re-used in this dissertation and the descriptions and analyses are revised. In these figures, each VM running a SF is marked as a Service Function Instance (SFI) [11].

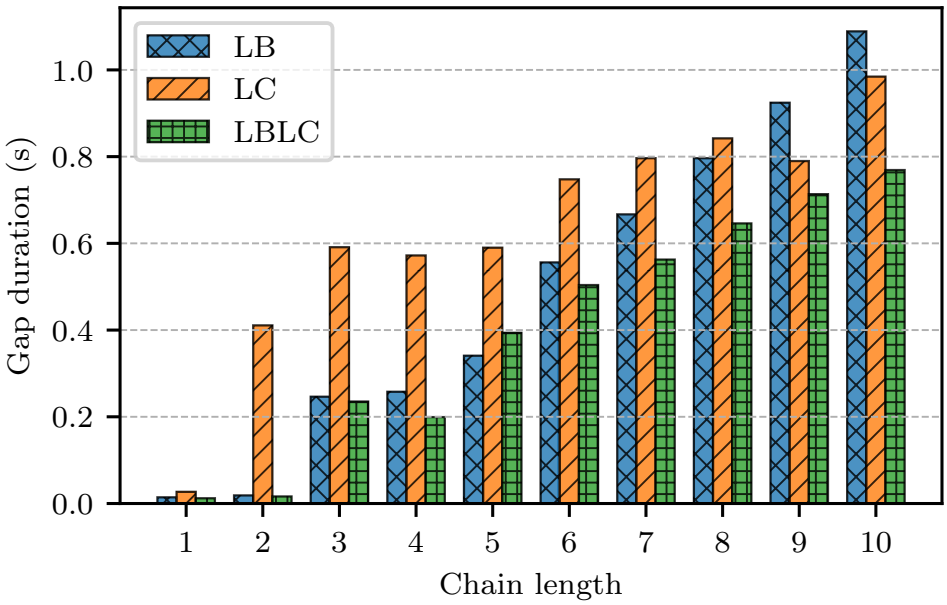
Measurement results of SFC rendering duration are presented in Figure 2.4a. Each bar consists of three parts, namely the latency to launch the VM, to boot the SF and to build the port chain (PC). Although the percentage of the building duration of PC is relatively small, the launching and booting delays of SFs are relative high. As illustrated in the Figure, as the chain length increases, both SFs launching and PC building delays show a linear trend. The booting delays remain unchanged and constitutes for roughly 100 seconds. It can be concluded that distributing SFs over multiple compute nodes with e.g. LB algorithm does not significantly speed up the SFC rendering latency.

Measurement results of the gap delays are illustrated in Figure 2.4b. Confidence intervals are not plotted here for readability since variances in results are very small. Although the SFC-OStack orchestrator redirects traffic only after all SFs in the chain report a fully active status, the gap latencies are still in the order of seconds, which is inconsistent with expectations. This is mainly due to the fact that the latency performance of creating and applying network policies into Neutron network is limited for OpenStack Pike version. Compared to the rendering latency, the gap delays of different algorithms show obvious differences. In contrast to the fluctuating trend of the LC algorithm, both LB and LBLC show a linear increase in general. Compared to other approaches, LBLC has the best overall performance. For a SFC with 10 SFs, the gap delay of LBLC is less than 0.8 second compared to the delay about 1.1 second with LB. So the gap delay is reduced by about 29% with LBLC compared to LB.

The OWD measurements results are illustrated in Figure 2.5. The delays of LKF are presented in Figure 2.5a. With the chain length ranging from 1 to 10, OWDs of all three algorithms with LKF show an upward trend. While the LB shows the highest OWD, the LC algorithm shows the best overall OWD performance. OWD results of LBLC are located between the LB and LC approaches. For a chain of 10 SFs, the LC algorithm can reduce the OWD by about 20% compared to the LB. This result is as expected, since LC significantly reduces the transmission delays. And the additional overhead of LKF is small compared to the transmission delays of the testbed used



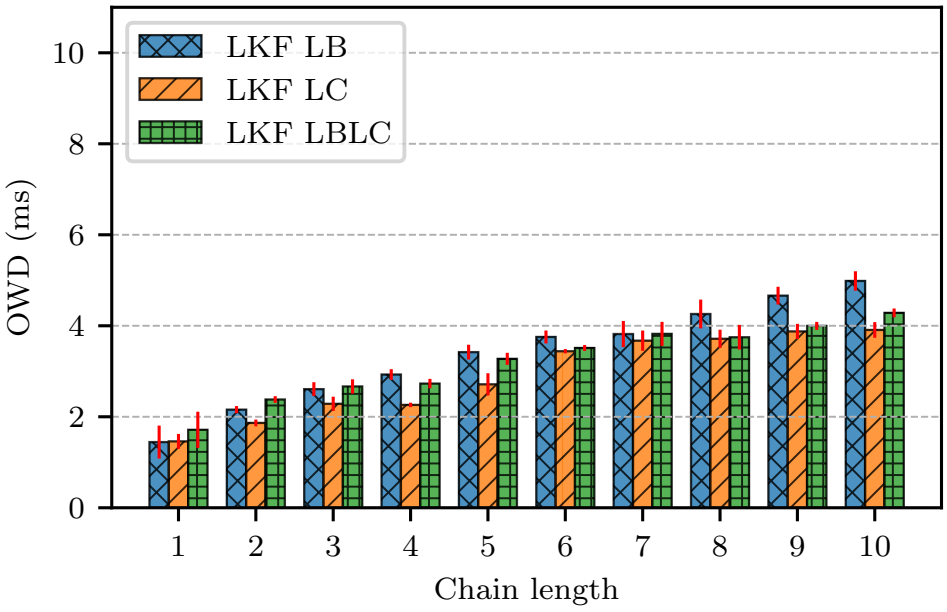
(a) Rendering latency.



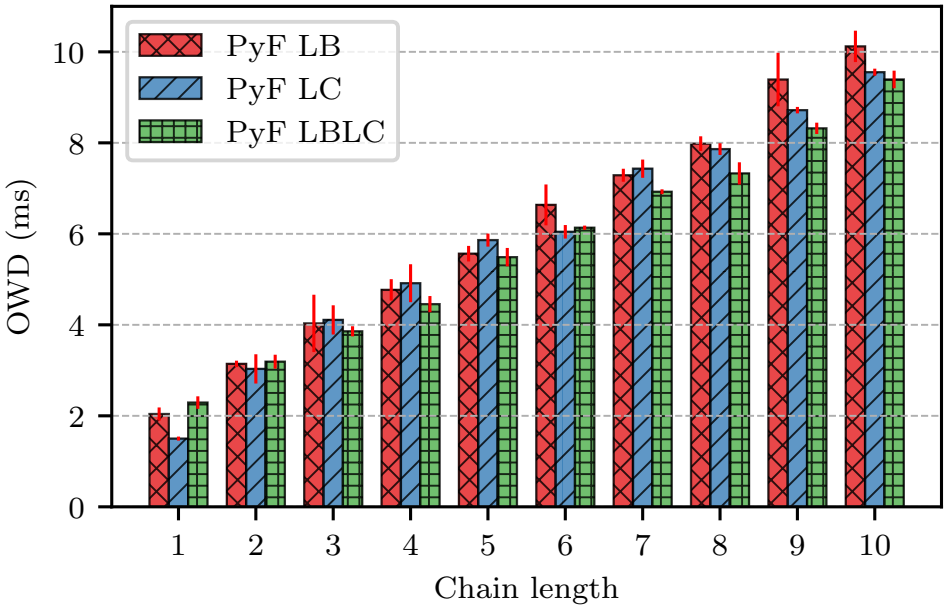
(b) Gap latency.

Figure 2.4: Measurement results of rendering and gap latencies w.r.t. different SFC chain lengths. Reprinted from **my conference paper** [11].

in this work. In contrast to LKF, the user space forwarding with packet socket is expected to require much more packet processing latency because of e.g. additional data copying and context switching. Therefore, the LBLC approach should present the lowest OWD since it consider both transmission and processing latencies. The measurement results shown in Figure 2.5b are clearly in line with the expectations.



(a) Linux kernel forwarding.



(b) Forwarding with Python AF_Packet.

Figure 2.5: Measurement results of OWD w.r.t. different SFC chain lengths. Reprinted from **my conference paper** [11].

When the chain has more than two SFs, LBLC has the best delay performance. When the length of the chain is nine, LBLC presents the minimal delay of 8.3 ms. In comparison, LB shows 9.4 ms and LC shows 8.7 ms. For the SFC with 10 SFs, the LBLC strategy is able to reduce the OWD by about 10% compared to the default LB algorithm. It can be concluded that the proposed LBLC algorithm can provide overall better latency performance when both transmission and processing delays are taken into account.

2.6 Summary

Different types of latencies introduced by SFC on OpenStack cloud platform are researched in this study. Three heuristic algorithms are designed and implemented with the SFC-OStack framework for latency-aware SF placement problem. A rigorous measurement campaign is performed on the OpenStack cloud platform with following important observations: (i) The OWD of probing packets through SFC can already reach several milliseconds, even for the minimal chain length and minimal packet processing operation in Linux kernel. A user space implementation with `AF_Packet` can double the delay with the same processing operation. This latency performance is not sufficient for URLLC use cases target by 5G network systems. This issue is further researched and addressed in the work described in Chapter 3. (ii) The proposed LBLC heuristic can reduce the OWD by about 10% compared to the default LB strategy. (iii) There is a gap delay of hundreds of milliseconds. These conclusions reveal practical challenges in deployment and management of SFC on OpenStack cloud platform with existing technologies.

Due to limited working time and experience in development on OpenStack cloud platform, many components of the SFC-OStack framework can be further improved. For future work, it is also interesting to investigate the root causes of gap latency introduced by SFC-OStack and the underlying OpenStack platform. Furthermore, heuristic algorithms proposed in this work can be extended to take more network performance parameters into consideration besides latencies, such as the available maximal bandwidth of the compute node, the maximal number of network connections and so on.

3 Reducing Latency in Virtual Machines: Enabling Tactile Internet for Human-Machine Co-Working

All contents in this Chapter has been published in my journal paper [4]: "Reducing latency in virtual machines: Enabling tactile Internet for human-machine co-working." IEEE Journal on Selected Areas in Communications 37, no. 5 (2019): 1098-1116.

3.1 Introduction

The core requirement for a tactile Internet that enables human-machine co-working is the low-latency communication [20–24]. Both machines and humans need latencies of less than one millisecond for a wide range of co-working scenarios. For example, for humans working in virtual worlds and interacting with robots or other types of machines, latencies for visual, audio and tactile feedback have to be lower than 15 ms, 3 ms and 1 ms, respectively [25]. To operate in a stable manner, machines based on remote control loops also requires ultra low latencies [26]. As a concrete example, consider a classical inverted pendulum whose remote controller is deployed in the cloud platform. Closing the control loop through the communication network has to introduce additional latencies and packet losses.

The influence of delays between angle sensor and actuator (motor) on the stability of a pendulum is illustrated in Figure 3.1. All results are generated by simulation. The graph on the left shows the angle of the pendulum with different sensor-to-actuator delays, i.e. 50 ms, 40 ms and 1 ms. For these three sensor-to-actuator delays, the inter-packet delay is fixed to 1 ms. For the graph on the right, the sensor-to-actuator delay is fixed to 1 ms while the inter-packet delay is configured to 10 ms, 5 ms and 1 ms.

As presented in Figure 3.1, the pendulum becomes very unstable when the sensor-

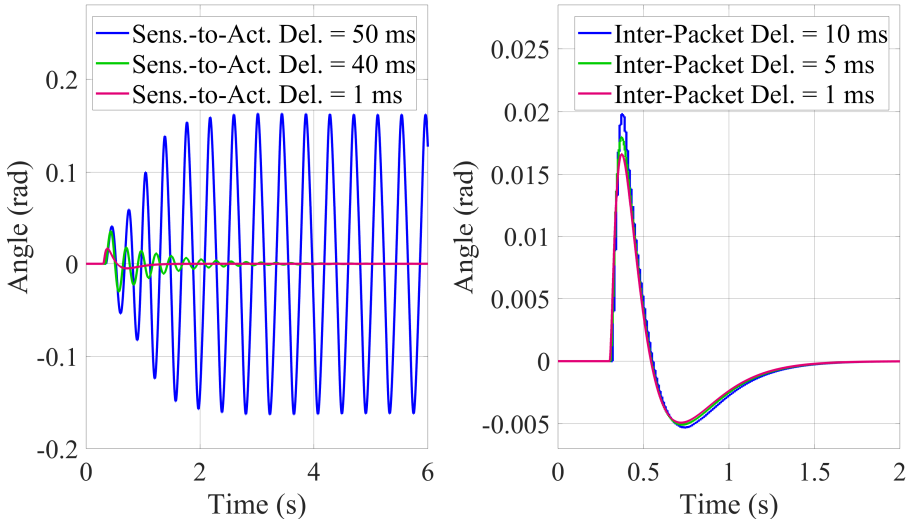


Figure 3.1: Angle of an inverted pendulum. The pendulum tries to reach stability for different sensor-to-actuator latencies and different inter-packet delays. Reprinted from **my journal paper** [4].

to-actuator delay reaches 50 ms. So the pendulum will never reach its stability in the correct position. When sensor-to-actuator delay is reduced to 40 ms, the pendulum can reach the stability after about 3 seconds. Therefore, the sensor-to-actuator delay has a significant impact on the performance of the pendulum and Quality of Service (QoS) of pendulum based applications and systems.

The allowed end-to-end communication latency requirement of one millisecond is defined in the 5G communication standard for automation in vertical domains [6]. As already described in the first chapter of this dissertation, the typical allocation of individual 5G communication network latency budget components is illustrated in Figure 1.3. As showed in the Figure, only 0.6 ms is allocated for the wired domain.

There are two main latency components in the wired domain: (i) Basic communication delay over fiber where the delay is bounded to the speed of light (3.34 μs per kilometer) and fiber characteristics. (ii) Delay introduced by communication nodes. These communication nodes are traditional switches and routers in the conventional *store and forward* network paradigm. However, in the upcoming future communication networks, there is a paradigm shift from *store and forward* to *compute and forward*. All communication nodes can now process and manipulate received network packets, instead of simply forwarding them without any complex computational operations. The new paradigm *compute and forward* can be realized with emerging network softwarization technologies, such as SDN [27] and NFV [2]. These promising technologies also enable the trending concept of MEC. MEC enables local processing of data, which in turn is able to reduce latencies on communication paths. When the maximum distance between sensor/actuator and the MEC platform is 25 km and the speed of fiber used is 2000000 km/s, 0.25 ms is required on the fiber communication. Therefore, only 0.35 ms is left for NFV and SDN processing in the MEC platform.

The summary section 2.6 of measurement results in the previous Chapter 2 and

related works [5, 28] show that achieving low latencies is a well-known challenge in communication networks. While latency types that are proportional to the available transmission bandwidth and data volume can be reduced by increasing transmission capacity and applying better data compression methods, dealing with packet processing delays and its various constant latency contribution poses a significant challenge [29]. Furthermore, recent studies [11, 30] have shown that NFV, for which high flexibility is highly designed, imposes significant packet transmission and processing demands. These demands can introduce relative large latencies that cannot be ignored.

At the time (2018-2019) of this research work on this topic, most virtual switches are already relative fast [31, 32]. However, the STOA VNFs based on conventional Linux kernel networking technologies running on VMs are relative slow, especially the packet Input/Output (IO) and processing operations. Zhang *et.al.* [33] proposed a minimal packet forwarding NF with the centralized approach, which presents an end-to-end delay of more than 2 ms with only one VM running SF. This latency performance is obviously far too bad with regard to the above-mentioned delay budget of 0.35 ms.

Ultra-reliable low-latency NFV and practical general MEC platform built with COTS hardware and open source software are explored in this work. An ultra-reliable low-latency SFC management named Chain bAsed Low latency VNF ImplemeNtation (CALVIN) is designed, implemented and rigorously evaluated in this work. In CALVIN, VNFs are implemented either in the Linux kernel space or in user space and are deployed on each own VM. While the fastest eXpress Data Path (XDP) technology is used for kernel space NFs, high-performance user space packet processing framework Data Plane Development Kit (DPDK) is employed by CALVIN to implement user space NFs. Both of them can achieve the best software packet processing latency currently available. The measurement results of evaluations on practical real-world testbed demonstrate that the proposed CALVIN framework can achieve an end-to-end latency on the order of 0.32 ms for the basic packet forwarding NFs. With CALVIN, it is possible to implement advanced NFs on a MEC platform with COTS hardware and open source software, such as Network Coding (NC) and traffic encryption/decryption with Advanced Encryption Standard (AES).

The proposed CALVIN approach makes it possible to handle more advanced network functions such as NC and traffic encryption with AES in a generic virtualized MEC setup, while meeting the end-to-end 1 ms latency requirement of the tactile Internet.

3.2 Background and Related Work

The main components of a typical NFV based service loop inside the MEC platform is illustrated in Figure 3.2. All packets of a network flow are received by the MEC platform through the ingress network port of a service proxy. Then they are processed by a chain of VNFs and forwarded to the target server running the requested service. After the processing on server, response data is generated by the server

and transmitted back to the client via the egress port of the server proxy. Ingress and egress network ports are endpoints that are exposed to the public network. On OpenStack platform, this network is normally also called the external network. This network is separate from the internal data network mainly for performance and security considerations. So these public endpoints are required for remote clients out of the cloud to access the services provided by the MEC system.

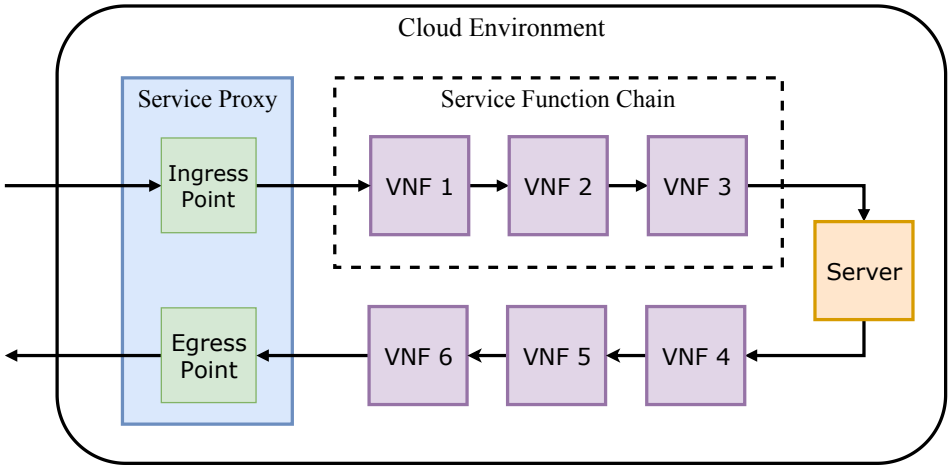


Figure 3.2: The service loop in a MEC cloud environment. Network traffic packets from clients are processed by a SFC consisting of an ordered sequence of VNFs to reach the server running the required application service. Reprinted from **my journal paper** [4].

As presented in Figure 3.2, all ordered VNFs of a SFC work typically as a pipeline. This pipeline is normally implemented on a cloud computing platform, e.g. OpenStack. The cloud computing platform can provide flexible and controllable management over the underlying physical computing, networking and storage resources. In this work, all VNFs and application servers are implemented as software programs running on VMs that are managed by the OpenStack cloud platform.

A typical networking infrastructure setup of the OpenStack cloud platform is presented in Figure 3.3. As plotted in the Figure, multiple VMs are interconnected in a virtualized overlay network. This virtual networking overlay is built on top on the physical networking between compute nodes to enable configurable multi-tenant networking for VMs. For example, two VMs deployed on different compute nodes that are connected to different physical networks (different routing entities) can be located in the same Local Area Network (LAN), namely in the same broadcast domain.

To provide a virtual overlay network on top of the underlying heterogeneous physical network, two software bridges (or switches) on each compute node are used to connect the Virtual Network Interface Controllers (vNICs) to the Physical Network Interface Controllers (pNICs). While the integration bridge is used to connect all VMs running on the same compute node, the tunnel bridge supports encapsulation and transmission of network virtualization tunneling protocols, such as Virtual Extensible LAN (VXLAN) and Generic Routing Encapsulation (GRE).

As illustrated in Figure 3.3, different types of latencies are introduced by compu-

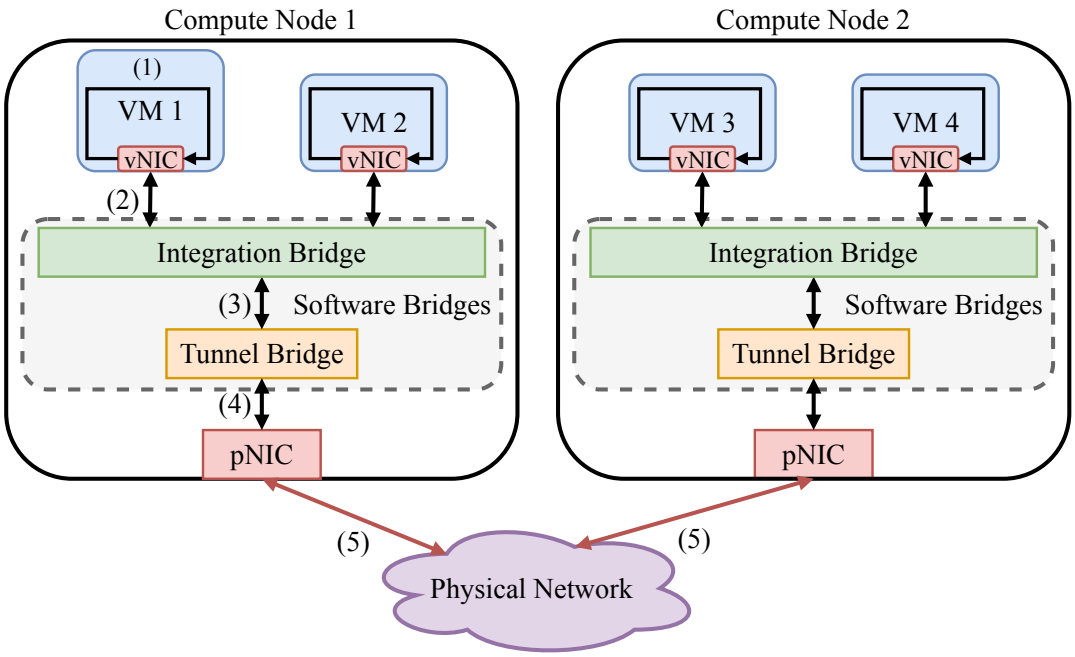


Figure 3.3: A typical cloud computing infrastructure scenario where multiple VMs are connected to a virtualized network overlay. Reprinted from **my journal paper** [4].

tational and network components (main components are marked with numbers 1-5 in the Figure):

- Latency between VM and the integration bridge (1, 2): Minimizing this delay component is the main focus on this work. This delay part mainly consists of two parts:
  - The time required to transmit packets between the VM and integration bridge through the vNIC (2). This part of latency mainly depends on the vNIC implementation and has two main subparts:
    * Time required to transfer packets between virtual switch and the ring buffer of vNIC. This delay can be reduced to the order of only some microseconds [31] with accelerated software switch data planes, such as using Open vSwitch with DPDK Datapath (OVS-DPDK).
    * Time required to transfer packets between vNIC and the VNF running on the VM. If the VNF is implemented with the conventional standard Linux kernel networking API, namely Berkeley sockets, this latency subpart can become a bottleneck in the low latency virtual network overlay.
  - Processing delay inside each VM (1): All packets in a network flow need to be processed by the VNF running inside the VM. The processing latency depends heavily on both algorithms and technologies used to implement the VNF. For ultra-reliable low-latency use cases, such as tactile Internet,

processing operation of each packet should be optimized as much as possible.

- Delay between the integration switch and the pNIC (3, 4): This part depends on the deployed virtual switches, orchestration cloud platform, and underlying physical resources of compute node [34].
- Time required to transmit packets between pNICs (5): This latency part fully depends on the underlying physical networking infrastructure. This part is not the research focus of this work, we try to use the STOA physical networking devices to achieve the best performance obtainable.

Reducing latencies marked as (1) and (2) in the Figure 3.3 is, to the best of my knowledge, still an open research question when I worked on this topic. The CALVIN framework proposed in this work is able to significantly reduce these latency components and provide an end-to-end service latency within the 0.35 ms budget required by 5G URLLC use cases.

Some recent studies [35–38] mainly based on mathematical modelling, analysis and modelling have considered the latencies involved in SFC. Compared to purely mathematical modelling, an experimental research is conducted in this work with empirical and rigorous latency measurements on OpenStack platform. The target of this work is to rigorously investigate the baseline latency of STOA SFC implementation on practical cloud platform. This empirical measurement study complements abovementioned mathematical analysis and simulation studies and provides reference latency values on real cloud platform, which can be used as reference data for future analysis and simulation studies for ultra-reliable low-latency VNFs. The CALVIN framework proposed in this work is able to achieve significantly better end-to-end latencies compared to STOA conventional frameworks.

Several recent works [39–41] have explored the placement problem of VNFs. However, these works mainly works on pure mathematical modelling and optimization. The CALVIN framework proposed in this chapter empirically and complementarily examines all aspects of the VNF placement issue on real cloud system.

Because the proposed CALVIN exploits the possibility to utilize complementary strengths of both in-kernel and kernel bypass technologies for latency reduction, a summary of related works for both in-kernel and kernel bypass packet processing frameworks is presented in this section.

Kernel Space Packet Processing

The XDP high performance data path for packet IO is available in Linux kernel since the version 4.8 [42]. With XDP, a network programmer can attach an extended Berkeley Packet Filter (eBPF) program to the very early hook in the RX path of the Linux kernel to decide the fast processing of the received packet. At the time of this work, XDP is still relative new, few related research studies have been conducted on its latency performance and implementation complexity [22].

eBPF and its features are quantitatively explored in [43]. Several relative strict limitations of eBPF are listed when building complex NFs that require complex packet

and flow processing. For example, eBPF programs have a limited number of instructions and do not allow unbounded loops. Therefore, in CALVIN, not all NFs are implemented with kernel space technologies. According to my preliminary performance evaluation of an XDP based elementary packet forwarding function, XDP can achieve a very low latency when deployed in the VM running the NF. Besides the latency performance, XDP has also multiple advantages over full kernel bypass technologies. For instance, compared to DPDK, XDP does not require dedicated isolated CPU cores and pre-allocated huge pages. XDP is also able to utilize the TCP/IP network stack and other functions already available in Linux kernel and apply the security model already used in Linux kernel.

The InKeV approach published in [44] studies the XDP based network functions for NFV uses cases. In InKeV, only simple NFs are implemented due to limitations of eBPF. However, the performance evaluation of InKeV is only performed on a single physical machine. Compared to it, my evaluation is more rigorous and performed on a multi-node physical cloud testbed.

User Space (Kernel Bypass) Packet Processing

In recent research works related to network softwarization, user space packet processing is much more popular than in-kernel mechanisms [45]. However, most related works put most effort on evaluating throughput instead of latency performance for STOA kernel bypass technologies. According to the survey and evaluation performed in [46], there are three most widely used kernel-bypassing high performance packet IO and processing framework, namely `netmap`, `PF_RING ZC` and Intel DPDK. It can be concluded from the work [46] that DPDK has the best hardware and software driver support, documentation, built-in samples and overall performance. Because of abovementioned advantages, DPDK also becomes now the de facto standard high-performance packet processing framework based on kernel bypassing. So this work also chooses the DPDK framework for user space NF implementation.

Combined Kernel and User Space Packet Processing

Compared to pure in-kernel or user space solutions, more closely related to my approach is the recent works that combine both in-kernel and user space technologies. In [47], the general architecture principles for building a hybrid kernel-user space VNF has been explored. With the combined approaches that are designed to provide the conventional socket API, most legacy VNFs can be ported and deployed with minimal modification. So this combined approach can significantly reduce the implementation complexity for new VNFs.

Recently, the VNF for Network Coding (NC) was implemented in [33] by employing the Kernel Network Interface (KNI) mechanism provided by DPDK library. This approach is referred as the “centralized approach” in this chapter because it aims to pack all VNFs into a single VM to avoid the additional latency introduced by inter-VM packet transmissions. The illustration of this centralized approach is presented in

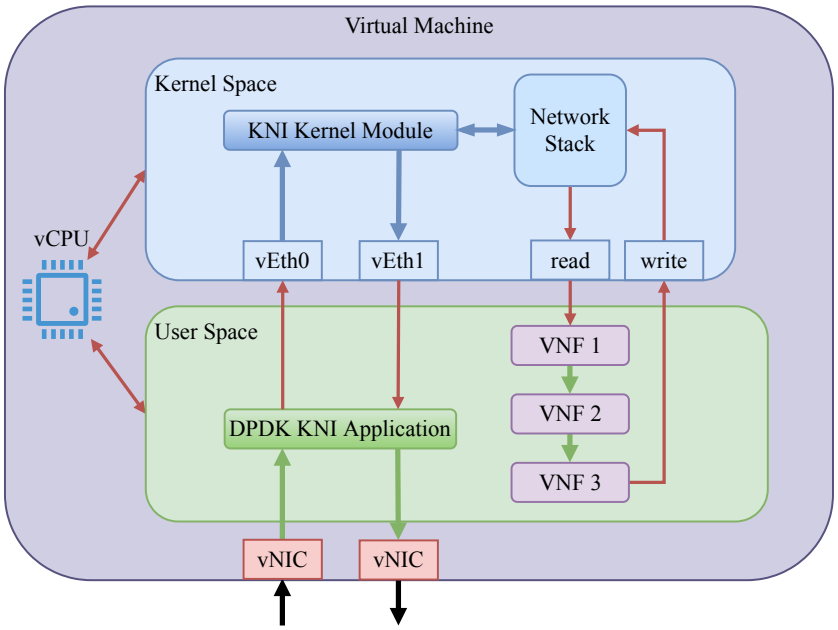


Figure 3.4: Graphical presentation of the centralized combined kernel and user space approach described in [33]. Reprinted from **my journal paper** [4].

Figure 3.4. Both in-kernel and user space mechanisms are applied by the centralized approach to make multiple small VNFs cooperate in a centralized manner.

As illustrated in Figure 3.4, for the centralized approach, each packet in a network flow needs to be exchanged between kernel and user space at least four times, which can be the bottleneck for latency performance. If zero-copy technologies are not available, additional packet data copies must be performed and this can introduce non-negligible latencies for tactile network applications. Furthermore, available physical computation and network resources of underlying compute node have to be shared by all VNFs both in kernel and user space. The scheduling between in kernel and user space VNFs requires context switches. These context switches can introduce unstable latencies and perform negative impact on the cache behavior of CPU. Moreover, latency also cannot be easily reduced by vertical scaling of computation resource. For instance, utilizing two virtual CPUs (vCPUs) does not simply halve the latency. This statement was verified with my measurements of the centralized approach on my testbed. All in all, these negative impacts of centralized approach can result in relative large and unstable latency.

In order to overcome the limitations and drawbacks of the centralized approach described in [33], the CALVIN proposed in this Chapter distributes VNFs over a chain of VMs (namely, to build a SFC) on which Network Functions (NFs) run either completely in the kernel space or completely in the user space.

3.3 Proposed Approach: Chain bAsed Low latency VNF ImplemeNtation (CALVIN)

3.3.1 Overview of CALVIN

The core idea of CALVIN is to access and evaluate the nature of a VNF in complexity when processing packets. CALVIN is designed to take advantage of both in-kernel and user space (i.e. kernel bypass) high performance packet processing frameworks available on Linux. Each VNF is deployed within a separate VMs to build a high-performance SFC. These two practical design choices of CALVIN completely eliminate the context switching and data transmission overhead of packet processing in different spaces, thereby significantly reducing the overall end-to-end service latency.

The design and implementation of CALVIN framework can provide following main advantages: (i) Avoid the non-negligible latency required for context-switching inside VM: Context switching between kernel and user space can introduce high latency [48]. Deploying and running the VNF in a single space can mitigate this negative impact. (ii) Avoid data copying or metadata copying between different spaces: The latency cost of data copying at any location in the data path has to be considered for low latency VNF implementation. Performing packet processing only in a single space can avoid data copying. (iii) The flexibility and scalability is improved. For the centralized approach presented in [33], due to the resource contention on the same VM, the latency performance can not be easily scaled with horizontal scaling. Compared to the centralized approach, CALVIN provides more flexible and horizontal scalability.

3.3.2 Classification of Virtualized Network Functions (VNFs)

The first step in developing high-performance VNFs for CALVIN is to classify them. Depending on the classification of VNFs, a given VNF is either deployed in kernel space or in user space. In CALVIN, VNFs are classified into three main groups:

- Elementary or Skeleton Functions: This is the minimal and fundamental functionalities required for all VNFs: (i) Receive packets from the ingress physical or virtual network interface. (ii) Create data structures and other required resources to store received packets for further processing. (iii) Send processed packets through the physical and virtual egress network interface. Both kernel and user space technologies must support these functions.

- Basic Functions: The main features of the basic functions are listed as follows: (i) Processing are performed only on the packet headers, not on the packet payload. So these NFs are typically stateless. Packet headers also have relative small sizes. Therefore, most basic NFs can be implemented without unbounded loops, which are currently not supported by the XDP technology. (ii) The computational complexity of basic NFs is relative low so that an acceptable latency performance can be achieved without applying some acceleration

technologies that currently only available in user space, such as Single Instruction Multiple Data (SIMD) and CPU cache prefetching. (iii) The VNF implementation has no strict requirements on specific hardware or software runtime (software environments), which are not available or accessible in the Linux kernel. With these characteristics, basic functions are suitable to be implemented and deployed directly inside the Linux kernel space, such as IP router, IP load balancer, stateless firewall and Network Address Translator (NAT).

- Advanced Functions: Advanced functions involve relative complex and computationally intensive operations compared to the basic functions. This group of VNFs has serveral features: (i) Besides packet headers, the payload data also needs to be processed. (ii) Because of the complexity involved, acceleration mechanisms that are only available in user space have to be applied for low latency performance. (iii) The implementation normally requires specific hardware or software runtime. For instance, the most widely used open source Random Linear Network Coding (RLNC) library `Kodo` currently requires the C++ runtime [49]. The C++ runtime is not available in the Linux kernel yet, so NC network functions using `Kodo` library can not be directly implemented in Linux kernel. Therefore, based on abovementioned features, advanced network functions, such as NC, data encryption, data compression, should be implemented in user space.

3.3.3 VNF Implementations Selection for VNF Classes

Compared to many available kernel bypass technologies, such as `Netmap`, `PF_RING`, and DPDK, relatively few in-kernel VNF frameworks are developed and available. Based on my literature review, eXpress Data Path (XDP) is the fastest in-kernel programmable network data plane framework available, providing fast packet processing at the lowest available software hook in the Linux kernel software stack [42]. Therefore, XDP is selected by CALVIN to implement VNFs running in the Linux kernel space.

Based on my literature review covered [45, 46, 50, 51], and preliminary measurements, DPDK is selected in CALVIN to implement kernel bypass VNFs in user space. Main reasons are listed as follows: (i) High throughput and low latency packet processing with COTS hardware: According to the performance comparison perform in [46], DPDK demonstrate the overall best bandwidth and latency performance among most widely used kernel bypass frameworks. (ii) Open source with comprehensive documentation: DPDK provides full control of nearly all aspects involved in the packet IO and processing. It also has very detailed documentation and extensive built-in examples that describe the best available practices to implement high-performance and efficient VNFs. (iii) DPDK becomes the de facto standard with wide support and related references: A wide range of both physical and software Network Interface Cards (NICs) are supported by DPDK with highly optimized drivers. Furthermore, most high performance software switches, such as OVS and Vector Packet Processing (VPP), have support for DPDK based fast path [51, 52]. OpenStack pike version supports OVS with DPDK data path out-of-the-box.

Although DPDK has the overall best performance, it (the version 18.02) also has some disadvantages: (i) Most drivers of DPDK only support the polling mode. Polling mode can be inefficient for energy consumption. (ii) DPDK currently only provides network protocol stack only up to the IP layer. The support of transport and upper layer network protocols requires third-party libraries or frameworks. (iii) User space frameworks bypass the relative mature security models provided by Linux kernel. User space frameworks currently lack of standard and verified security model, which is a hot research area.

3.3.4 CALVIN Architecture Design and Workflow

The architectural design of CALVIN is presented in Figure 3.5. CALVIN is built on top of the research-oriented SFC framework SFC-OStack introduced in the Chapter 2 and my conference paper [11]. The SFC-OStack framework is extended and improved both in control and data plane for ultra-reliable low-latency SFC system:

- CALVIN Control Plane: A classifier for different VNF groups is added that classify VNFs based on their description and specification into basic and advanced network functions. The processing pipeline of the VNFs is then converted into a functional chain of VMs with their computational and network configuration. Lifecyle management of all instances in the SFC description is handled by the SFC manager which renders and orchestrate SFC with underlying OpenStack services.
- CALVIN Data Plane: Multiple VMs are launched to run VNFs on several physical compute nodes in the data plane. In each VM, the VNF is located in either Linux kernel or user space. Traffic packets are received from the vNIC, processed by the VNF, and then transmitted through the egress vNIC. All VMs running on the same compute node are interconnected with OVS-DPDK.

CALVIN related operations that handle basic and advanced network functions differently require various configurations for both hardware and software used by the OpenStack cloud platform.

Accelerated Configuration of Virtual Network Infrastructure

- Each compute node must be equipped with pNICs that supports DPDK [53]. This is required to deploy OVS-DPDK as the integration and tunneling bridges on OpenStack.
- Dedicated physical CPU cores have to be assigned on each compute node for OVS-DPDK. In order to avoid any interruptions from other processes for low latency, polling mode and CPU core pinning are required by OVS-DPDK to achieve the lowest accessible latency. Dedicated CPU cores are isolated from the Linux kernel scheduler with the `isolcpus` configuration provided by the Linux kernel.

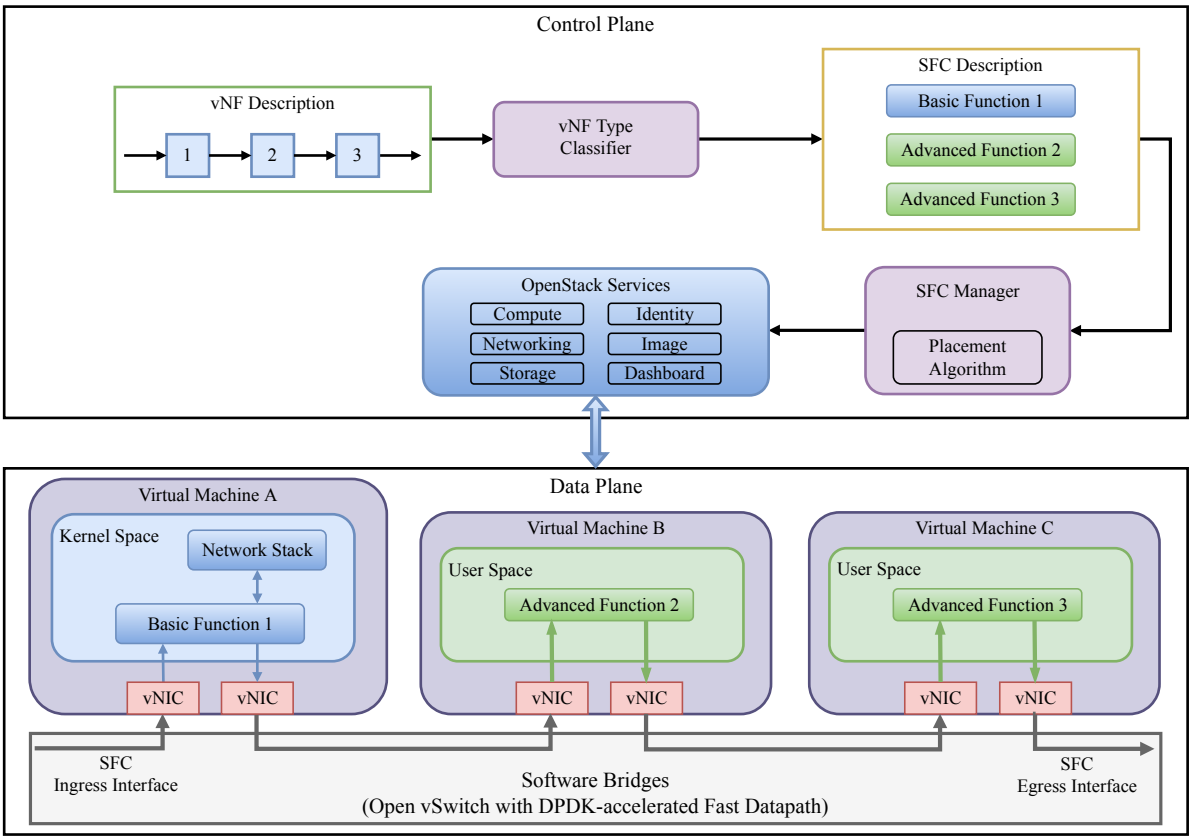


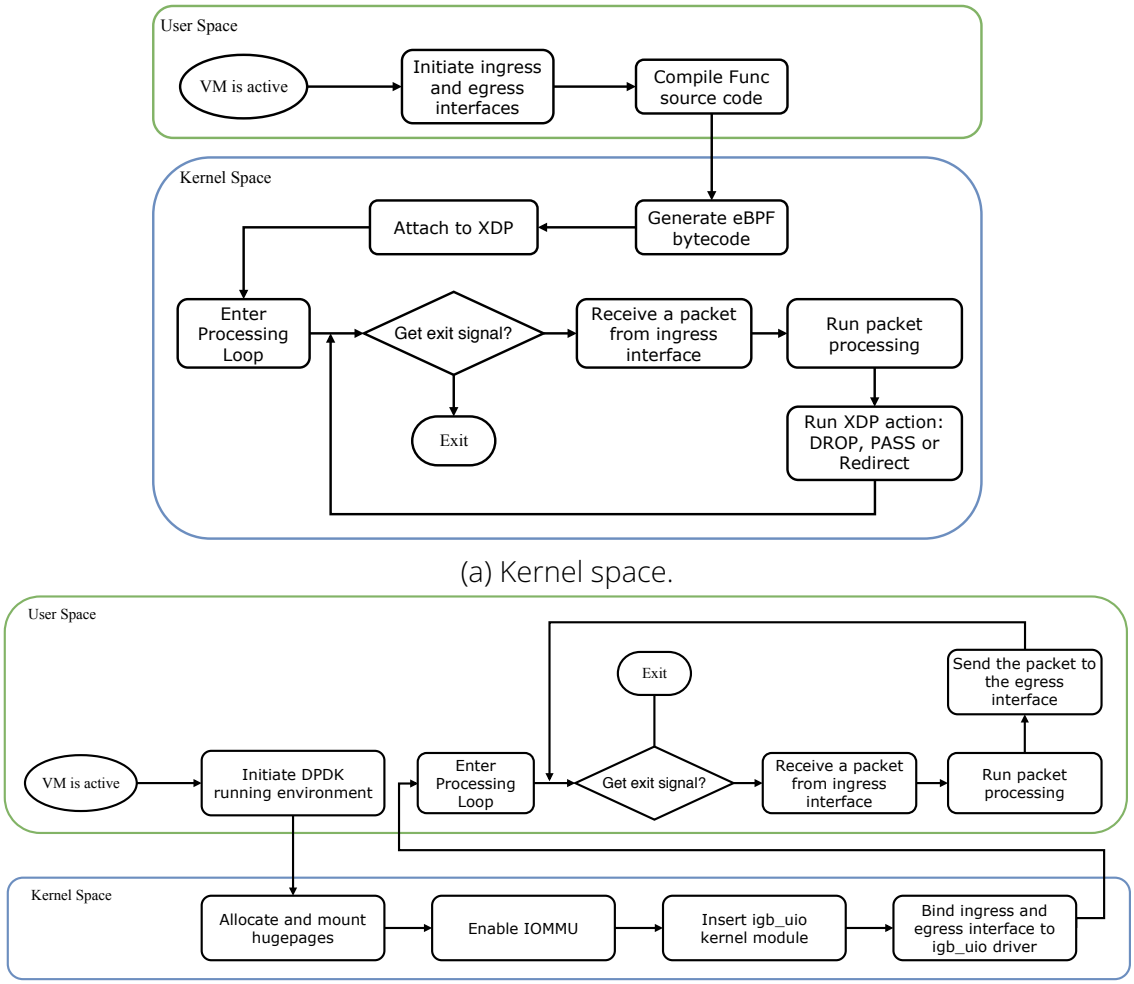
Figure 3.5: The architecture design of CALVIN. Fundamental elements in both control and data plane are illustrated. Reprinted from **my journal paper** [4].

- The Input-Output Memory management Unit (IOMMU) support should be enabled to allow guest VMs to access the pNIC through Direct Memory Access (DMA).
- Enough memory should be reserved to allocate hugepages for OVS-DPDK and other DPDK-based VNFs running in VMs on each compute node.

Configuration for VNF Processing

The Figure 3.6 illustrates the workflows of running basic and advanced network functions in Linux kernel and user space. Because the KVM is the default VM hypervisor on OpenStack (Version: Rocky), following configuration are optimized for the KVM hypervisor.

Kernel Space Because of the current requirements of XDP, The vNIC of each VM should support the assignment of a dedicated transport queue (TX queue). This feature can be enabled by applying the `virtio_net` patch to OpenStack (with version Rocky) Nova component. In order to run XDP programs with sufficient feature support, the linux kernel of the guest OS running within the VM should be updated to at least version 4.8. To compile, attach/detach and manage XDP programs, the BPF



(a) Kernel space.

(b) User space.

Figure 3.6: CALVIN workflows for basic and advanced network functions running in different spaces. Reprinted from **my journal paper** [4].

Compiler Collection (BCC) framework [54] or the libxdp library [55] can be installed to simplify the development process.

User Space The IOMMU should be available for vNICs for minimal latency cost. Sufficient memory spaces should be allocated for hugepages. Sufficient hugepages are required for both OVS-DPDK and all DPDK based advanced network functions. The DPDK kernel module `igb_uio` needs to be loaded for Poll Mode Driver (PMD) driver.

When all VNFs are running in user space, the latency overhead and complexity of the mechanism to exchange packets is non-negligible. Inter-Process Communication (IPC) mechanisms such as Unix Domain Socket (UDS) or shared memory are normally used for data exchange among running VNFs. IPC mechanisms are normally provided the OS kernel, which introduce additional non-negligible latency overhead due to context switching and data exchange. To avoid this overhead, CALVIN

maps one VNF per VM for SFC deployment. Compared to CALVIN, the centralized approach [33] injects packets from user space to kernel space, which can become a critical latency bottleneck according to my preliminary measurements. In order to achieve the best available performance, all advanced VNFs are implemented from scratch to utilize the high-performance facilities provided by DPDK.

3.4 Performance Evaluation of Elementary and Basic Network Functions

3.4.1 Measurement Setup for Elementary and Basic Network Functions

Main components of the measurement setup for end-to-end RTT are presented in Figure 3.7. Two compute nodes of a OpenStack cluster are used to deploy VMs and perform latency measurements with active probing UDP traffic. The UDP traffic is used for active probing since UDP and UDP-based protocols are normally used for latency sensitive applications. Probing with UDP can also avoid the non-negligible complexity of flow and congestion control of TCP for latency measurements.

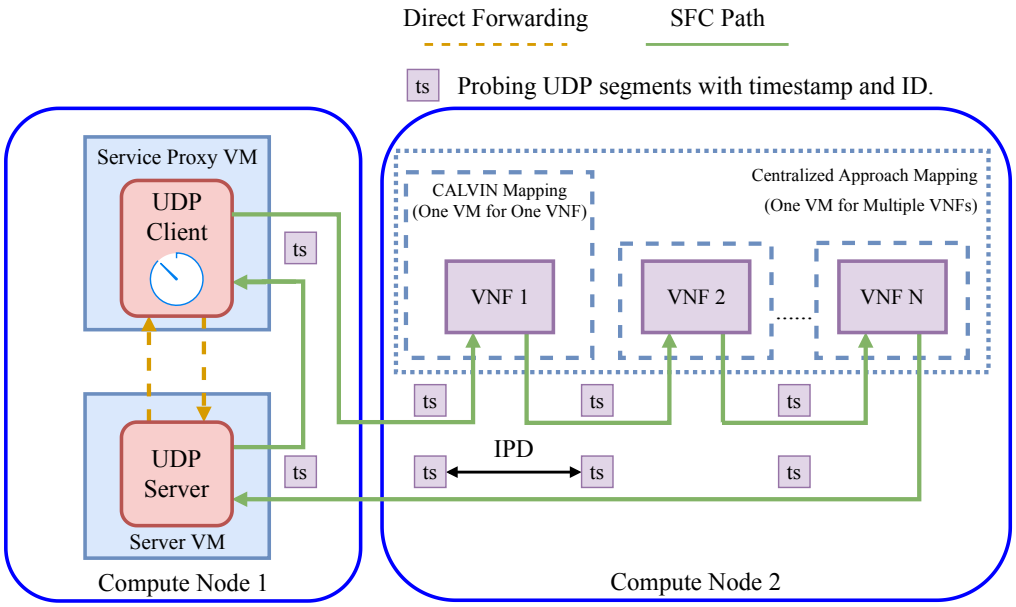


Figure 3.7: RTT measurement setup for CALVIN. Reprinted from **my journal paper** [4].

Architecture The measurement setup is aligned to the basic service loop presented in Figure 3.2. The UDP probing client is deployed on the service proxy. Probing client sends UDP packets to the server located on the same compute node, namely node 1. The deployed server simply bounces all received UDP packets back to the client as

quickly as possible, namely without any payload processing. In addition, as plotted in the Figure, probing UDP packets are forwarded directly to the server without leaving compute node to measure the direct forwarding latency of the underlying virtual Neutron network infrastructure (without any processing in the SFC).

To reflect realistic practical real-world network scenarios, both VMs running the probing client and server do not apply high-performance packet IO technologies such as DPDK or XDP. Both probing client and server work at the network layer, while all VNFs deployed in this evaluation run at the data link layer. Therefore, both probing client and server are implemented based on the conventional socket API provided by the Linux kernel. In contrast to the centralized approach proposed in [33], the components of the above introduced measurement architecture are distributed on two different physical compute nodes.

For each packet in the probing traffic, a Timestamp (ts) and an identification Number (ID) are added before the payload. While the ts is used to measure the RTT, the ID is used to identify lost or out-of-order packets. In measurements of this work, the workload of the probing traffic is tuned to avoid any losses and out-of-order of packets.

Testbed All performance measurements are performed on my practical NFV testbed, which consists of COTS servers connected via two independent Gigabit Ethernet connections. Compute and network node of the OpenStack cloud platform are equipped with 4 CPU cores (Intel 4^{th} Core i5), 16 GB RAM, 128 GB SSD and two Gigabit NICs (Intel 9301CT Gigabit CT). The OpenStack cloud platform (Version Pike) is deployed on multiple physical nodes running the Ubuntu Server (16.04 LTS). For each compute node, while one NIC is used for management and external traffic, another separate NIC is used for the internal data network for all compute nodes. This separation avoid the impact of management traffic on the latency measurements in the data plane. In addition to the OpenStack's standard (minimal) compute, networking, identification, and storage services, the official Neutron SFC plugin, SFC-OStack framework and Neutron OVS-DPDK plugin are also installed. The `Virtio` technology is used for the vNIC and OVS-DPDK for high-performance packet IO. KVM is used as the hypervisor for the management of VMs and the Ubuntu cloud images are used to implement and deploy different VNFs.

Elementary and Basic VNFs The elementary network function and two different basic network functions are implemented and measured in this work.

- FWD - Elementary Forwarding: Packets are received from virtual ingress vNIC and directly forwarded to the virtual egress vNIC without any processing.
- ATS - Appending Time Stamp: The timestamp of the current VNF receiving and sending a particular packet is appended at the end of the UDP payload, just before the packet is transmitted. Because the ATS function changes the size of the UDP payload, the checksums of both IP and UDP headers must be recalculated and updated. To ensure the relative fair latency comparison, the check-

sum calculations are implemented purely in software (instead of using hardware checksum offloading). For IPv4, UDP checksum can be disabled when use all-zeros and the IPv4 checksum can be calculated only with the header data. Thus, the ATS function can be used to estimate the latency caused by a relative trivial processing on packet header.

- XOR - XORing UDP Payload: This network function performs an XOR operation with the same static key on all bytes of the UDP payload. Compared to the above introduced ATS function, the XOR function can be used to estimate the additional latency caused by relative non-trivial computation operations on the packet payload.

Metrics

- Latency: The RTT of each single UDP packet is used to estimate the end-to-end latency performance. This RTT is selected because of following reasons: (i) Both forward and backward paths are included in the RTT measurements. (ii) Measurements of RTT do not require time synchronization between VMs running VNFs. According to the experiments performed in [56], performing time synchronization at the VM level is highly prone to errors due to the interferences of clocks among VMs running on the same compute node.

- Bandwidth: The maximum achievable throughput is measured in this work with the `Iperf` tool [57]. The UDP mode of `Iperf` is used for bandwidth measurements. `Iperf` is not used for RTT measurements because `Iperf` currently does not provide per-packet RTT measurements for both TCP and UDP traffic. In this work, a home-made tool is developed for RTT measurements of each packet in the probing traffic. By manually adjusting the `Iperf` target bandwidth experimentally in steps of 10 kbit/s for 5 minutes each, the maximum bandwidth is determined such that no lost or out-of-order packets are detected by the `Iperf` client.

Active Probing Parameters Two main parameters need to be configured for active probing traffic, namely the Interpacket Gap (IPG) and the UDP payload size. Based on preliminary measurements on my testbed, consistent RTT values can be obtained for relative small IPGs on the order of a few milliseconds. Relative small IPGs can result in additional queuing inside VNFs and other network components in the testbed, which is theoretically analyzed in [58]. My latency measurements focus on the end-to-end latency with relative light workload without additional packet queuing. Increasing the IPG can also lead to increased RTT values. According to my analysis, this behavior should be mainly due to the batching mechanisms of both in-kernel and user space packet processing frameworks which are activated for better throughput performance when the network traffic has low workload. In order to avoid both significant queuing and batching latencies, the IPG is configured to 5 ms in all measurements in this work based on preliminary measurements and calibration.

For UDP payload size, 256 and 1400 bytes are selected as the lower and upper bounds. Based on my preliminary evaluations, UDP packets with payload size smaller than 256 bytes cannot be handled correctly by the current version of XDP. Meanwhile, the official SFC plugin of OpenStack Neutron of version Pike does not currently support jumbo frames. The maximum available UDP payload size depends on the Maximum Transmission Unit (MTU) of the underlying physical and virtual Ethernet. Assume the MTU of the Ethernet is 1500 bytes, the maximum UDP payload size is limited to 1472 bytes. The upper bound of 1400 bytes is selected to reserve enough free spaces for optional IP header options or other tunneling protocols.

For both latency and bandwidth measurements, for each scenario, the measurements are repeated for 50 times. For each probing scenario in the latency measurements, the probing client sends 500 UDP packets.

Measurement Scenarios

Performance Comparison of different VNF technologies As introduced in the previous section, in CALVIN, XDP is chosen to implement basic network functions and DPDK is used to implement advanced network functions. In order to benchmark the performance of selected technologies, both of them are benchmarked. Besides them, the LKF approach introduced in 2.5 is also considered as a reference for traditional packet processing technologies.

- XDP: Because of following listed limitations of XDP, only FWD and XOR functions are implemented: (i) At the time of this work, the maximum number of instructions per XDP programs is limited to 4096 eBPF instructions. This significantly limit the complexity of the computational operations and amount of data that can be processed by a single XDP program. (ii) The available memory space for XDP processing per packet is limited by the size of the original received packet. Operations outside of this memory range are currently prohibited. So the ATS function cannot be implemented with a single XDP program.
- DPDK: Thanks to the high flexibility and programmability provided by DPDK, all three network functions can be implemented as DPDK applications. By default, all DPDK applications run in polling mode and always consume 100% of the available CPU resources. To minimize IO and processing latency as much as possible, the number of packets in a processing batch (burst) is configured to one.
- LKF: It is a built-in Linux kernel feature for packet forwarding at the network layer. Due to its trivial operation, the LKF is one of the fastest running features in kernel space. Because LKF does not provide any programmability, LKF cannot be used to implement different VNFs in kernel space.

Comparison of Centralized Approach [33] and CALVIN The proposed CALVIN approach with the STOA centralized approach [33], which was one of the first approaches studied to implement advanced network functions such as RLNC as a VNF.

The centralized approach has to be re-implemented and evaluated on the testbed used in this work because its source code is not publicly available [33]. To examine the distributed VNF aspect of CALVIN, two FWD VNFs on two separate VMs are implemented for CALVIN approach, while for a clear comparison, only one FWD VNF is implemented for the centralized approach. The centralized approach is allocated with a VM with twice the computational resources (both vCPU and memory) compared to each individual VM used by CALVIN. Therefore, the comparison is actually between two FWD VNFs in CALVIN and one FWD VNF in the centralized approach, which have the same available computational resources. More specifically, for the centralized approach (presented on the left side of Figure 3.8), the packet enters the VM through the left ingress vNIC, traverses the FWD VNF and exits through the right vNIC. For CALVIN, as illustrated on the right side of Figure 3.8, packets enter the left vNIC of VM 1, traverse the XDP FWD, exit through the right vNIC of VM 1, enter the left vNIC of VM 2, traverse the DPDK FWD, and exit through the right vNIC of VM 2.

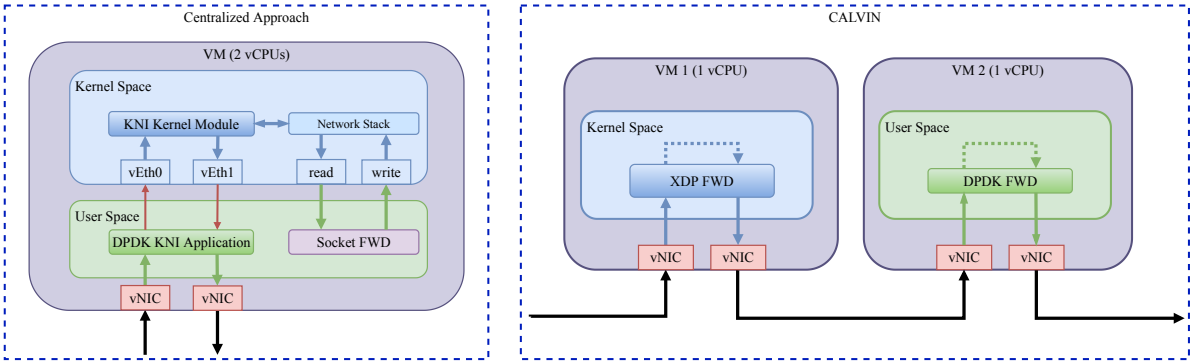


Figure 3.8: Illustration of measurement setup for the RTT comparison between centralized approach and CALVIN. Reprinted from **my journal paper** [4].

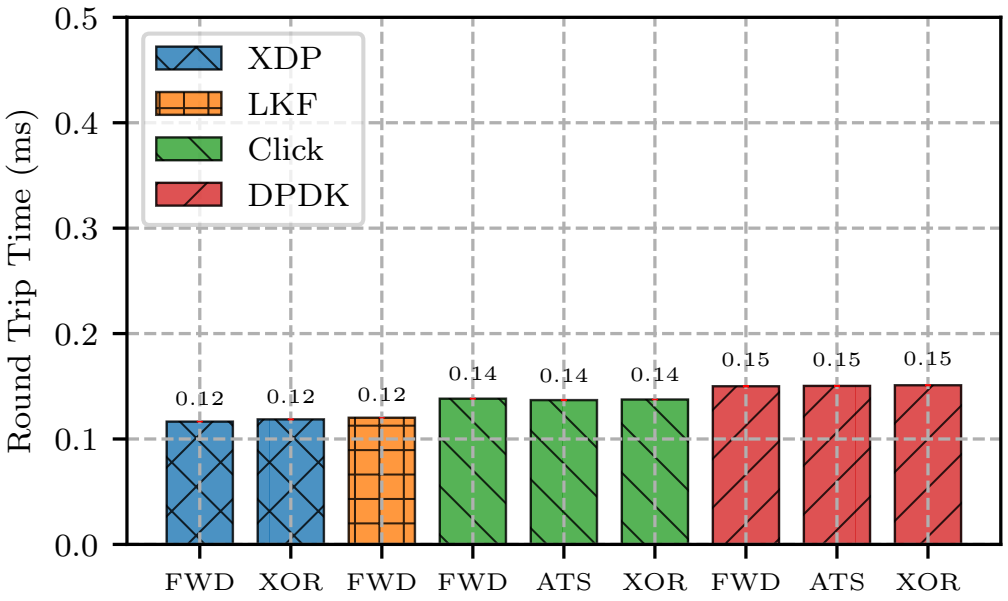
3.4.2 Measurement Results and Evaluation for Elementary and Basic Network Functions

RTT Measurements of Elementary and Basic VNFs for Different Technologies

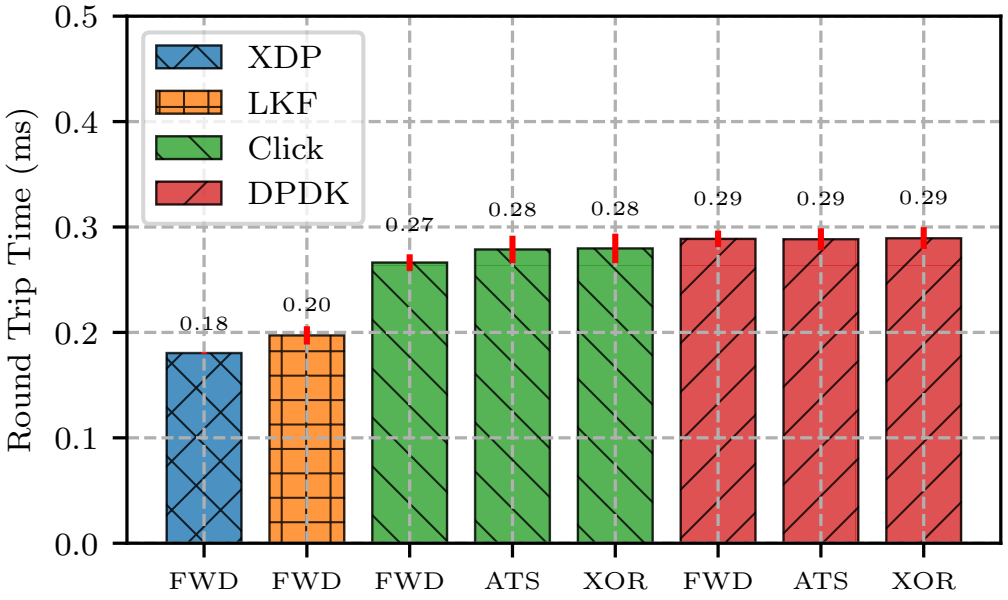
Figure 3.9 illustrates the mean values and 95% confidence intervals of the RTTs of elementary and basic network (FWD, ATS and XOR) functions implemented with different candidate technologies. It can be observed from Figure 3.9 that the RTTs of the two basic VNFs (i.e. ATS and XOR) are comparable to the respective RTTs of the basic FWD VNFs of all selected packet processing technologies. It can be concluded from this result that the additional delay in payload processing for the basic ATS and XOR VNFs is relatively negligible compared to the elementary FWD latency.

Examining the in-kernel technologies closely, it can be observed from Figure 3.9 that although the XDP and LKF have the similar RTT performance for small packets, the XDP FWD is about 10% faster than the LKF FWD for large packets of 1400 bytes.

Furthermore, it can be observed from Figure 3.9 that for small packets of 256 bytes, the RTTs for in-kernel processing (LKF and XDP) and user-space processing (DPDK)



(a) Payload size: 256 bytes.



(b) Payload size: 1400 bytes.

Figure 3.9: Means and 95% confidence intervals for RTT of different VNF technologies in kernel space and user space. The 95% confidence intervals for the 256 byte payload size are very tight and barely visible in this plot. Reprinted from my journal paper [4].

are very similar. Additionally, for large packets of 1400 bytes, the RTTs for user-space processing are much longer (about 50%) than for in-kernel processing.

Observed latency performance differences seem to be mainly caused by two types

of non-negligible overhead introduced by evaluated basic network functions: (i) The overhead of copying packet frames from the virtual ring buffer of vNIC to the VM memory [59]. (ii) The overhead of additional metadata processing of packet frames. In the LKF, like most conventional in-kernel technologies, the standard data structure `sk_buff` containing a lot of metadata must be allocated for each received packet. The latency overhead of extracting metadata and allocating `sk_buff` can be very high [42] for ultra-reliable low-latency scenarios. In contrast, XDP processes packets at the lowest point of the Linux software network stack, without allocating `sk_buff` data structures and without any parsing and pre-processing of packets [42]. For traditional LKF, metadata structures must be created and the Linux kernel needs to check the routing table to update the appropriate Media Access Control (MAC) and IP addresses. As a result, LKF is only slightly slower than XDP for large packets, because LKF needs more time for metadata processing.

Comparison between Centralized Approach and CALVIN

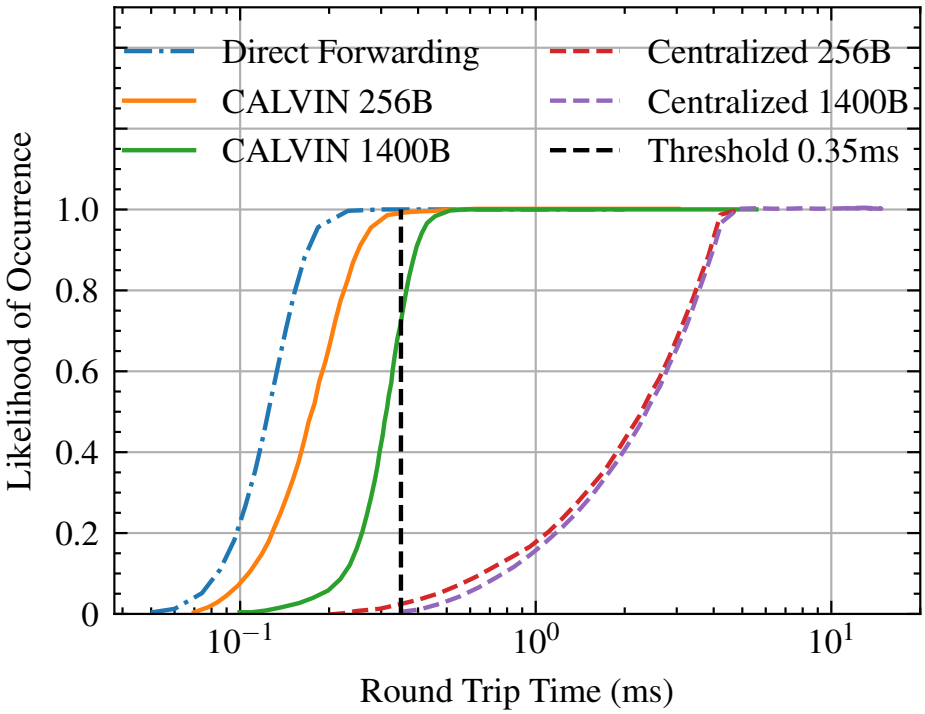


Figure 3.10: The RTT performance comparison of FWD VNF between centralized approach and CALVIN. Reprinted from **my journal paper** [4].

RTT Results Figure 3.10 shows the RTT measurements for the basic FWD VNF. As described in Subsection 3.4.2, direct forwarding is the baseline for transmission latencies introduced by the underlying virtual network infrastructure of OpenStack platform. It can be observed from Figure 3.10 that the centralized approach exceeds the delay threshold of 0.35 ms in the best case, while the RTT of the proposed CALVIN is below the 0.35 ms threshold with a probability of about 70% for 1400 bytes pack-

ets and close to 100% for 256 bytes packets. The measured average of 256 bytes and 1400 bytes packets RTT are listed as follows: CALVIN: 0.19 ms and 0.32 ms, respectively; centralized approach: 2.30 ms and 2.39 ms, respectively. Therefore, it can be concluded from these results that CALVIN can meet the strict ultra-reliable low-latency requirements described in Section 3.1. In CALVIN, when assuming a latency budget of 0.35 ms, there is also extra time available for more complex data processing.

According to the measurements perform in [60] without using STOA accelerated packet processing frameworks, the compact mapping, which is used by the centralized approach, has lower latency than the distributed mapping of VNFs, which is used by CALVIN. However, for compute-intensive VNFs, distribute mapping can provide better latency performance. The RTT measurements above demonstrate that the distributed mapping selected by the CALVIN can achieve ultra-reliable low-latency requirement of tactile Internet for elementary and basic network functions. Distributed mapping used by CALVIN allows each VNF to focus on its processing tasks in a single space (kernel or user space) and leaves the packet transmission to the underlying software integration bridge, which already has proven very good low-latency performance. Allocating a dedicated VM to each VNF with distributed mapping may be considered a waste of resources. However, each VM can be highly optimized for its special purpose, and distributed mapping is more consistent with the emerging concept of unikernel [61].

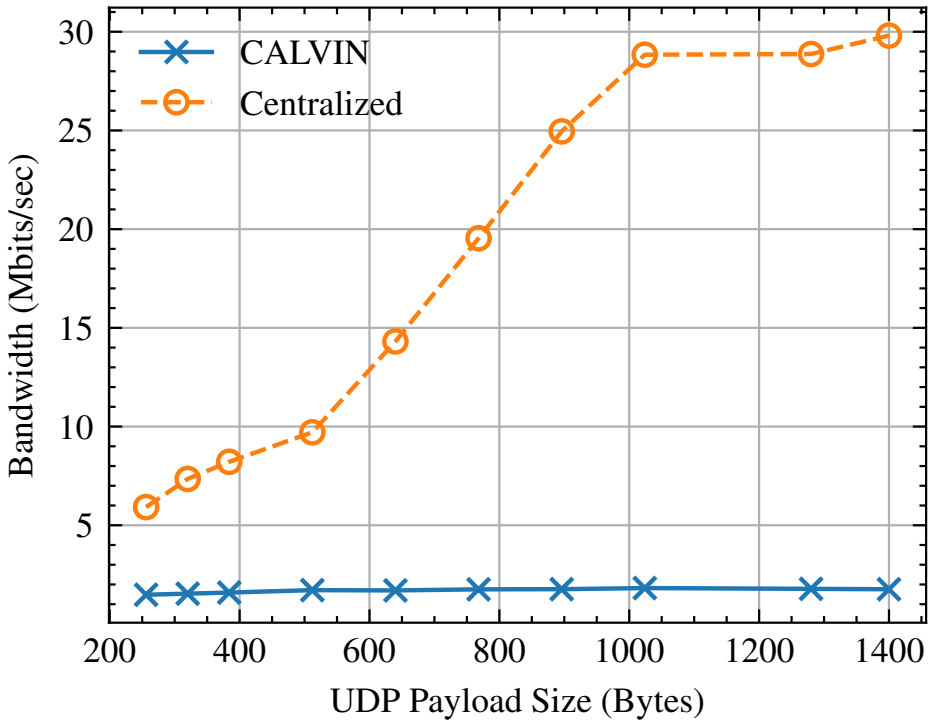


Figure 3.11: The bandwidth performance comparison of FWD VNF between centralized approach and CALVIN. Reprinted from **my journal paper** [4].

Bandwidth Results The bandwidth measurements for UDP payload sizes ranging from 256 to 1400 bytes are illustrated in Figure 3.11. It can be observed from Figure 3.11 that the bandwidth supported by the centralized approach is substantially higher than that of CALVIN, especially for large packets (larger than 512 bytes). For UDP packets with the payload of 1400 bytes, the available bandwidth of the centralized approach is more than 15 times higher than that of the CALVIN. Compared to the minimum bandwidth of about 6 Mbits/s supported by the centralized approach (with UDP packets with the payload of 256 bytes), the maximum bandwidth of CALVIN is only between 1.4 and 1.7 Mbits/s.

Latency Bandwidth Trade-off All in all, the RTT results in Figure 3.10 and the bandwidth results in Figure 3.11 show the trade-off between per-packet latency and bandwidth in a real and practical VNF implementation. The centralized approach uses the conventional Linux socket API, which is primarily designed for high-throughput best-effort service applications, such as file transfers, which typically have bursty traffic. Therefore, the kernel network stack includes a number of mechanisms (normally enabled by default) to improve bandwidth performance, such as batch processing. Batch processing collects multiple packets and then processes the batch once with accelerated methods, for example, with SIMD instructions or enhanced CPU caching. Batch processing can increase the latency of each packet because the first packets in a batch must wait for subsequent packets to fill a batch before they can be further processed.

Even if the batch processing is not used in DPDK user space applications, batch processing in the Linux network stack can significantly slow down the centralized approach. This is because the KNI mechanism injects all packets into the normal Linux kernel network stack, where batch processing is employed and cannot be easily avoided. To the best of my knowledge, the default batch processing in the kernel stack cannot be avoided without modifying the Linux kernel source code. In the design of CALVIN, the modifying of source codes of the underlying infrastructure software should be avoided.

In proposed CALVIN, this batch processing is avoided in Linux kernel space by implementing in-kernel VNFs using the latest fast packet IO techniques such as XDP. Because CALVIN prioritize per-packet latency performance over overall bandwidth, CALVIN tries to avoid batch processing in all VNF implementations. All VNF implementations in CALVIN run in the Run-To-Completion (RTC) mode, i.e. they receive a single packet, then directly process it, and transmit it out as fast as possible. Thus, as clearly shown in Figure 3.10 and 3.11, CALVIN reduces the per-packet delay at the cost of supporting relative lower available bandwidth.

As introduced in Section 3.1, for most use cases of tactile Internet for human-machine co-working, low per-packet latency is typically much more important than support for high bandwidth. The human-machine co-working packet traffic, e.g. the control messages of a robot arm are typically very small so that support for low bandwidths is already sufficient. As illustrated in Figure 3.1, the 5 ms IPG used in my measurements is sufficient for a typical pendulum application. At the same time, control messages delayed by batch processing can profoundly disrupt tactile human-

machine collaboration. Therefore, CALVIN only supports low bandwidth in exchange for significantly reduced per-packet latency. The bandwidth supported by CALVIN can be improved in future work by bandwidth management mechanisms, such as a load balancer that distributes packet flows to a set of duplicate VNFs to enable parallel packet flow processing.

Table 3.1: CPU usage of the physical compute node. Reprinted from my journal paper [4].

Approach	User (%)	Sys (%)	Guest (%)	IDLE (%)
Centralized Approach	25.2	47.2	2.9	24.7
CALVIN	25.2	23.0	2.2	49.6

CPU Resource Usage The usage of the 4 cores of the physical CPUs of compute node 2 plotted in Figure 3.7 is measured using the common tool `mpstat`. Since CPU resource scheduling for all running VMs is managed by the OpenStack compute service (namely Nova), instead of individual CPU, the global average utilization of all cores are measured in this evaluation. For each scenario, a period of 10 minutes measurement is performed with a sampling period of 1 second. The CPU usage levels of the centralized approach and CALVIN at user level (User), kernel level (Sys), and for a niced guest (Guest) are all listed in Table 3.1. It can be observed from Table 3.1 that the centralized approach consumes twice as much CPU resources at the kernel (Sys) level compared to CALVIN. KVM uses the Linux kernel of the host OS as the hypervisor and uses Portable Operating System Interface (POSIX) threads for the vCPU of the guest OS. Therefore, the Sys CPU usage in the table reflects the vCPU of the VM running VNF usage. By avoiding the overhead of context switching and metadata processing for each vCPU, CALVIN greatly reduces the use of the host OS's kernel CPU time. As a result, CALVIN allows a much larger percentage of time for physical CPUs to be idle.

3.5 Performance Evaluation of Advanced Network Functions

The evaluation in Section 3.4 shows that CALVIN is able to complete the basic VNF with an end-to-end RTT of 0.32 ms. Therefore, considering the MEC latency budget of 0.35 ms as plotted in Figure 1.3, there is still a residual latency budget of about 0.02 ms for advanced packet processing functions. This section evaluates NC and data encryption as two practical examples of advanced VNFs with relatively high computational requirements. The practical applications and relevance of these two advanced VNFs are firstly described. Then the evaluation of the processing latency incurred by these advanced VNFs is performed using the same setup used for elementary and basic network functions. The purpose of this evaluation is to evaluate

whether the RTT reduction of the CALVIN implementation of the basic VNFs is sufficient to allow for practical advanced VNFs within the latency requirements of the tactile Internet.

3.5.1 Random Linear Network Coding (RLNC) Network Function

Network Coding (NC) linearly combines several original packets with coding coefficients to form coded packets that are transmitted through the network [62]. In RLNC, which is a simple but powerful NC scheme, the coding coefficients are generated randomly [63]. The main advantages of RLNC include: (i) The ability to recode received packets at all middle nodes in the network without the need for coordination, thus suitable for distributed environments [63]. (ii) A generic coding matrix that allows sparsity (judicious addition of zero values) to significantly reduce computational complexity [64]. (iii) Sufficient support for low-latency communication because of on-the-fly coding mechanisms [65]. (iv) Heterogeneous field sizes that support heterogeneous communicating entities, increasing flexibility in practical heterogeneous environments [66]. (v) Relative small overhead between storage and transport layers, since the same code scheme can also be used for distributed storage [67].

Because of above listed advantages, several variants of RLNC schemes have been already proposed in NC research. Two main types of RLNC are focused by this work, namely, systematic block coding and convolutional sliding window coding. Block-based RLNCs were introduced to significantly reduce the computational requirements and control of NC. To further improve performance, instead of coding each packet in a flow, systematic coding firstly sends the original packets [68]. Packets built from linear combinations are then sent between the original packets or at the end of the coding block [69].

Sliding window NC has been introduced to reduce the in-order latency of coded network communication [70]. For systematic coding with a constraint coding window, sliding window RLNC has a shorter transmission latency compared to block coding, which usually requires comparable computational resources [71].

Despite extensive research on NC in recent years, practical deployment of high-performance RLNC in real-world networks is still rare. One of the most challenging difficulties to deploying RLNC is the limited programmable computing resources on heterogeneous network nodes, which are currently mostly used only for switching and routing decisions. NFV and SDN provide new flexibility for deploying advanced and innovative features in the network [72]. With the help of NFV technologies, RLNC can be implemented as software programs running in VMs or containers that can be instantiated on any NFV-capable network node. In addition, SDN technologies can re-direct the packets flow to VNFs running RLNC network functions and orchestrate them in the SFC. However, to the best of my knowledge, the latency of RLNC as a high-performance VNF in a practical MEC system has not been studied, measured and evaluated in detail.

Per-packet RTT latency measurements in this work considers NC encoding (computationally equivalent to recoding in middle network nodes) with a Galois Field (GF) size of $GF(2^8)$ and a redundancy of 25%. For block coding, the block size of 32 pack-

ets is used. The window size of 8 packets is considered in this evaluation.

3.5.2 Advanced Encryption Standard (AES) Encryption

Data encryption and decryption are key security components of modern communication to ensure the confidentiality and integrity of the data. At the time of this work, more than 40% of the World Wide Web (WEB) traffic is transmitted in encrypted form over Hypertext Transfer Protocol Secure (HTTPS), and this trend is increasing [73]. Therefore, numerous network functions require encryption and decryption functionalities, such as caching and Deep Packet Inspection (DPI). As with RLNC network functions, data encryption requires processing of the entire packet payload, which is a non-negligible computational workload. AES is focused by this work, a widely used encryption standard commonly used for data transmission and storage. According to my preliminary measurements, CALVIN approach enables the AES encryption of small packets within a 20 μs latency budget on a general-purpose MEC platform.

3.5.3 Measurement Setup of Advanced Network Functions

The following additional factors need to be considered when evaluating advanced functions compared to measurement setup for elementary and basic functions describe in Subsection 3.4.1.

VNF Implementation

Because of in-kernel technology limitations described in Subsection 3.3.3, DPDK is chosen by CALVIN for the implementation of all advanced VNFs. RLNC and AES encryption VNFs are implemented on top of the elementary DPDK L2 FWD application. The RLNC network functions is implemented with the Network Coding Kernel Library (NCKernel), which is built on top of the widely used `Kodo` library [49] to support different common network coding communication variants including the sliding window. The lightweight and portable `Tiny-AES-C` library is used to build AES applications.

Multiple VNFs in parallel is implemented to evaluate the scalability of the proposed CALVIN framework. Scalability is a key performance metric for practical MEC platform, as a key aspect of virtualization is enabling running multiple virtual instances on shared and limited hardware resources.

Metric

Because the RTT performance for basic forwarding network function has already been evaluated in Section 3.4, the measurements for the advanced VNFs in this Section focus only on the packet processing latency. The processing latency is defined as the delay it takes for a VNF to fully process a received packet. For VNFs capable of generating redundant packets, such as RLNC encoders or recoders, this processing latency also includes the time required to create all redundant packets.

Methodology

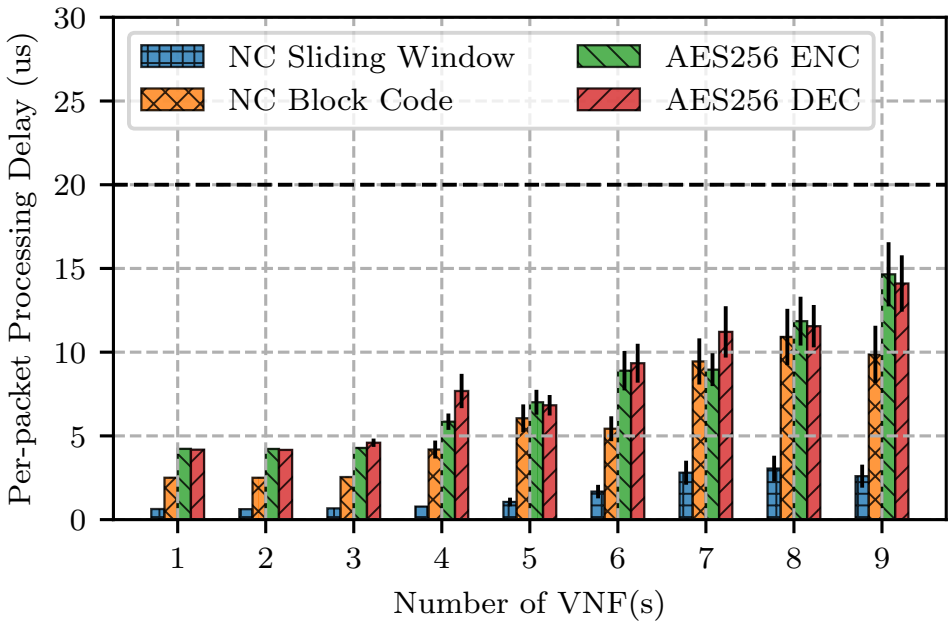
To evaluate the impact of VNF workload requirements on processing latency, computational operations of deployed VNFs should be performed in parallel. This requirement can be very difficult to meet if the probing traffic is generated by a client running on a remote VM. Precise synchronization mechanisms have to be deployed on the virtualized network infrastructure to ensure that all probe packets arrive at each VNF at the same time. Therefore, to evaluate advanced VNFs, instead of using additional probing clients to generate UDP traffic, probing UDP packets are generated locally by each VM running the VNF. Locally generated traffic ensures that VMs are continuously backlogged so that the worst-case processing latency can be measured. Each VM is always busy processing the probing traffic and the OpenStack scheduler needs to handle resource scheduling between all running VMs. Latency values for warm-up and tail probing packets are not included in the measurement results. For each number of deployed VNFs, 50000 UDP probing packets are generated.

3.5.4 Measurement Results and Evaluation for Advanced Network Functions

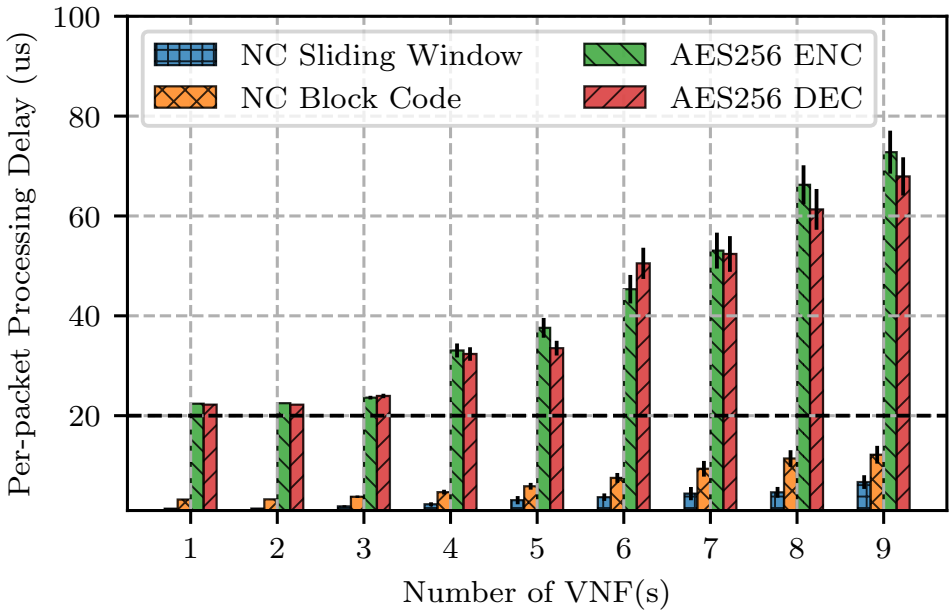
The evaluation of elementary functions in this work presents a mean RTT latency of 0.32 ms for the elementary FWD VNF for large packets with CALVIN framework. Based on the latency budget of 0.35 ms assumed in this work, a latency budget of 20 μs is considered with a safety margin of 10 μs.

The measurement results of processing time is illustrated in Figure 3.12. For small 256 bytes size packets, the processing time for all evaluated advanced network functions are within the 20 μs constraint. As the number of deployed VNFs increases, so does the CPU load, and as soon as the number of VNFs exceeds the number of CPU cores dedicated to VM processing, the processing time increases linearly. In my measurement setup, one of the four available CPU cores is heavily used by the OVS-DPDK software bridge, which runs in polling mode with the default DPDK behavior. The increase in latency is mainly caused by contention for shared and limited physical CPU resources. For a specified maximum allowed latency budget, e.g. 5 μs, the maximal number of VNFs allowed to run in parallel can be observed, e.g. 3 VNFs. As the load balancer redirects a given workload traffic to multiple VNFs, the higher the number of parallel VNFs supported, the higher the supported overall available bandwidth.

For large packets of 1400 bytes size, a significant increase in processing time can be observed in Figure 3.12 compared to small packets of 256 bytes. Although the processing time for RLNC is still relatively low and well within the budget of 20 μs, AES encryption is not feasible anymore even when the VNF is exclusive to one dedicate CPU core, i.e. for three or fewer parallel deployed VNFs. For RLNC network functions, the processing time for sliding window encoding scheme is significantly shorter than for block codes. Even with large packets and high contention for limited CPU resources, the latency of sliding window RLNC remains below 7 μs for 9



(a) UDP payload size: 256 bytes.



(b) UDP payload size: 1400 bytes.

Figure 3.12: Means and 95% confidence intervals for processing times in microseconds for computationally intensive advanced VNFs. Reprinted from **my journal paper** [4].

parallel running VNFs.

3.6 Summary

In this chapter, the design, implementation and evaluation of the proposed Chain bAsed Low latency VNF ImplemeNtation (CALVIN) framework is described, which is an approach for the orchestration of distributed SFC for ultra-reliable low-latency tactile Internet applications. In CALVIN, high-performance VNFs are implemented and deployed either purely in the Linux kernel space (for basic network functions) or in the user space (for advanced network functions) to avoid all latency overheads required for context switching and data exchange between these two spaces. Furthermore, in CALVIN, all VNFs are implemented in a fully distributed manner with one dedicated VM for one VNF mapping. All VNFs in CALVIN are implemented with the best STOA fast packet IO and processing technologies to avoid complex and heavy metadata processing and large batch processing of conventional Linux network stack and APIs.

In the rigorous evaluation of this work, the performance of elementary forwarding network function implemented with various STOA high-performance packet processing technologies is firstly measured. According to the measurement results, the promising XDP technology can achieve a RTT performance of 120 μs for 256 bytes packets and 180 μs for large packets with 1400 bytes. The conventional Linux kernel forwarding function can incur about 10% higher latency. The user space DPDK technology further increase the latency performance provided by XDP by up to 50%. Based on these observations, in CALVIN, while the XDP is used to implement all in-kernel VNFs for basic network functions, DPDK is adopted to implement user space computationally complex VNFs.

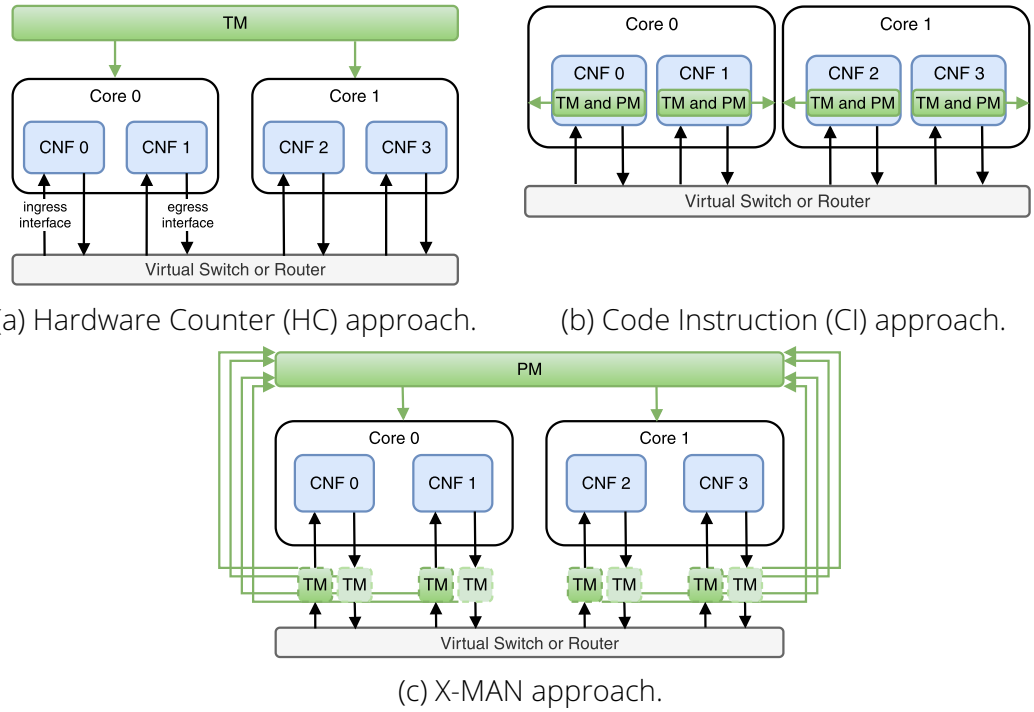
In this work, the proposed CALVIN approach is rigorously benchmarked against the STOA centralized approach proposed in [33]. According to my measurements on practical OpenStack cloud platform, CALVIN is able to achieve significantly better latency performance (0.32 ms for 1400 bytes UDP packets) compared to the centralized approach (2.39 ms for 1400 byte UDP packets). In terms of disadvantages, CALVIN can only supports much lower packet bandwidth, about 1.5 Mbit/s, rather than centralized methods (depending on the UDP packet size, between 6 to near 30 Mbit/s). As a result, CALVIN is able to achieve much shorter per-packet latency at the cost of reduced packet throughput, which is a strict requirement for typical tactile Internet applications with a 1 ms end-to-end RTT latency budget.

4 X-MAN: A Non-intrusive Power Manager for Energy-adaptive Cloud-native Network Functions

All contents in this Chapter has been published in my journal paper [7]: "X-MAN: A Non-intrusive Power Manager for Energy-adaptive Cloud-native Network Functions." IEEE Transactions on Network and Service Management (2021).

4.1 Introduction

The need for flexible and ultra-reliable low-latency network service provisioning in emerging network paradigms such as the fifth-generation communication systems (5G) [74] and the tactile Internet [4] has given rise to the emerging paradigm of microservices that are connected together to form network services. In response to this promising trend toward microservices, conventional VNFs deployed in VMs (described and used in Chapter 3) are shifting to the new Cloud-native Network Functions (CNFs). More specifically, the CNF operates in the application container, that is, in the so-called cloud-native manner, is therefore referred to as cloud-native network function or containerized network function [75]. The strict QoS requirements of advanced network paradigms, such as 5G and Tactile Internet, require CNFs to provide ultra-reliable low-latency packet processing performance [76]. Meanwhile, growing concerns about the energy consumption of the network and IT infrastructure require smart power management of the CPU cores that are used by CNFs for packet processing. For example, all advanced network functions in the CALVIN framework introduced in Chapter 3 are implemented with DPDK technology, which works in the polling mode by default and consumes 100% of the available CPU time all the time even without any workload traffic. This design can significantly improve the packet processing performance, but also introduce hard challenges of the energy efficiency of CALVIN. When VNFs are updated to the latest CNFs, this issue become more challenging since more services need to be deployed on the same physical node.



(a) Hardware Counter (HC) approach.

(b) Code Instruction (CI) approach.

(c) X-MAN approach.

Figure 4.1: Conceptual comparison of existing approaches and the proposed X-MAN approach. Reprinted from **my journal paper** [7].

To address the above introduced issue, two main power management mechanisms have been published at the time of this research: (i) A CPU Hardware Counter (HC) based strategy [8]. (ii) A Code Instruction (CI) strategy [9]. So far, the HC strategy has focused on estimating the workload of CPU cores from the counter activity of CPU cores using a pre-trained regression model. According to the evaluation performed in [8], performing a relatively high accurate workload estimation of CPU cores purely from counter actives is significantly challenging for a complex CNF deployment. Improving the estimation accuracy with relatively complex Machine Learning (ML) models could increase the estimation latency and make HC-based approaches unsuitable for highly responsive power management required for e.g. tactile Internet. In addition, as illustrated in Figure 4.1a, the HC approach can only estimate the total aggregated workload of all CNFs running on a CPU core. Therefore, HC based power management cannot accurately optimize the current operating frequency of CPU cores based on the workload intensity of each individual CNF. However, the workload intensity of each individual CNF can effectively reduce the required frequency of a CPU core, for example, when several low-intensity CNFs can run in parallel at low latency on a CPU core with a relatively low frequency.

Compared to HC approach, the CI approach, which is illustrated in Figure 4.1b, requires modifying or patching the source code of the CNF program. Therefore, this approach is intrusive because all CNFs that involved in the power management have to be modified or patched. Furthermore, until the time of this work, the CI approach focuses on the power management of only a single CNF. For global power management, an additional orchestration layer is required to manage multiple independent

CNFs running simultaneously, which is very common for the scenario of microservices.

This Chapter presents a system integration research study that performs the design, implementation and evaluation of the novel XDP-Monitoring energy-Adaptive Network functions (X-MAN) framework to enable a non-intrusive traffic workload monitoring of each individual CNF and frequency scaling of each individual CPU core through a power management module with a global view of all running CNFs deployed on a CPU core.

The XDP-Monitoring energy-Adaptive Network functions (X-MAN) framework, which is graphically presented in Figure 4.1c, performs an integration of the following two main system components:

- Linux Kernel Traffic Monitors: As conceptually presented in Figure 4.1c, X-MAN is able to monitor workload traffic through the in-kernel traffic monitoring modules, thus avoid the conventional intrusive traffic monitoring of CNF code instruction. These in-kernel monitoring modules analyze the characteristics of the workload traffic at the virtual interfaces of each CNF with minimal overhead using the functionalities provided by XDP technology.

- User Space Power Management: Based on the monitored characteristics of workload traffic, a global PM in the user space adjusts the operating state (P-state) of the available CPU cores to guarantee ultra-reliable low-latency packet processing while striving to be energy efficient as much as possible. The power management algorithm used by X-MAN employs low-complexity workload prediction and step-wise P-state adjustments.

In conclusion, the X-MAN architecture novelly integrates lightweight in-band kernel space traffic monitoring and out-of-band user space power management. Kernel-space traffic monitoring and user-space power management of X-MAN build a practical and complete system for reducing energy consumption while supporting ultra-reliable low-latency cloud-native networking functions. Based on an extensive literature review, to the best of my knowledge, the proposed X-MAN framework introduced in this Chapter is the first research work to pursue this system integration approach across Linux kernel space and user space to address effective energy savings for CNFs.

In order to perform the performance evaluation, comprehensive and rigorous measurements of the X-MAN system are performed on a practical physical testbed supporting CNF deployment and 10 Gbps Ethernet. These measurements include comparisons to STOA CI approach and HC approach. According to my measurements and evaluations:

- Traffic Monitoring Latency: The measurement results present that X-MAN can consistently monitor workload traffic of 4 virtual interfaces with a monitoring latency of only 10 μs, while the HC approach proposed in [8] produces monitoring latency in excess of 20 μs and even ranges up to 80 μs.

- Energy Consumption Reduction: The X-MAN framework can reduce the CPU energy consumption of the random workload traffic profile described in Subsection 4.4.2 to less than half of the energy consumption of the STOA CI approach [9].
- Energy Consumption of the X-MAN system itself: Only 2.5% energy overhead is required by the mechanisms used by X-MAN framework. As a result, the energy overhead introduced by X-MAN is negligible and X-MAN is able to achieve a significant net reduction in overall energy consumption.

Comprehensive and rigorous measurements of practical physical testbed configurations demonstrate that the performance of the unique and novel X-MAN system integration approach greatly exceeds existing STOA solutions.

4.2 Background and Related Work

A large body of cloud computing research literatures [77, 78] examines load-adaptive energy-aware management of large-scale systems consisting of multiple CPUs, each with multiple dedicated cores. Several recent research works have focused on the use of ML technologies [79, 80]. For example, the so called Elastic resource flexing for Network function VIrtualization (ENVI) [81]. In [82], non-machine learning based adaptive management methods for large-scale systems are investigated. Abovementioned approaches are mainly targeted at scaling the total aggregated workload of many simultaneously running CNFs for CPU operations on long time scales or on the scale of large computing infrastructures that contain numerous CPUs. In comparison, the goal of the proposed X-MAN is to rapidly monitor the traffic workload of each CNF and adaptively adjust the core frequency of each CPU to take advantage of power saving opportunities that arise due to fluctuations in the workload of individual CNFs.

4.2.1 Power Management in Linux Kernel

In Linux kernel, both working-state power management and system-wide power management mechanisms are available [7]. Because the target of this work is to reduce the overall energy consumption under workload traffic, the working-state power management is focused by this work, like two recent most related works [8, 83]. This type of power management dynamically adjusts the power state of CPUs based on the current workload. Two different groups of performance states are generally available for the power management of x86-type CPUs, which are currently widely deployed on cloud-native computing platforms and are mainly offered by Intel Corp. and Advanced Micro Device (AMD) Inc: (i) Sleep state (C-states): They can significantly reduce the CPU energy consumption by entering sleep mode. However, relative long transition latency is required to sleep or wake-up the CPU. According to the measurements performed in [83], tens or even hundreds of microseconds are required. C-states are fully managed by the `CPUIdle` subsystem of Linux kernel

and can not be managed in user space (at least until the completion of this work). (ii) Operating Performance Points (P-states): P-states provide different Dynamic Voltage and Frequency Scaling (DVFS) configurations [84]. Adjusting P-states is currently the de facto standard approach to perform power management when CPU is busy running processing tasks. In comparison to C-states, which can only be managed by the Linux kernel, P-states can be managed in user space via the APIs provide by the `CPUFreq` subsystem in Linux kernel. Typically, by default, the Linux kernel adjust the P-states based on the current CPU utilization, which is sufficient for most common applications. However, for high-performance CNFs using PMD, the CPU utilization is always 100%, even if there are no any workload packets to process. Therefore, instead of relying on the current default mechanism provided by Linux kernel out-of-box, new mechanisms are needed to manage P-states based on the actual workload traffic to reduce overall energy consumption.

It should be noted here that Advanced RISC Machines (ARM) processors, which are also used in lightweight edge cloud systems, provide a similar DVFS power management mechanism available in user space. Therefore, X-MAN can be used on ARM processors. Due to limited time and hardware resources, the work in this Chapter focuses only on x86 CPUs.

4.2.2 CPU Core Load Estimation with Hardware Counters (HCs)

An estimation approach for CPU workload without any source code modification is proposed in [8]. This approach is so called out-of-band approach since it does not directly measure the workload traffic and use an indirect metric, namely the hardware counters of a CPU. The approach is also referred to as a black-box approach because it does not specifically systematically analyse the details of each individual running CNF and treat all CNFs as black boxes. In [8], Gupta *et.al.* argues that the non-empty polls of packets can change the CPU events. For example, these events can be cache misses and branch prediction errors due to the processing received packets. The relationship between CPU events, which can be measured with hardware performance counters (HCs) with small overhead [85], and the actual CNF utilization of the physical CPU core is researched. Based on measurements of hardware counters, an estimation method based on training with regression models is designed, which takes the frequencies of a chosen small set of CPU events (1 to 3 from over about 700 available CPU events) as algorithm input and estimate the corresponding actual CPU utilization. According to the evaluation performed in [8], this HC approach can achieve an estimation error below 5%. But this error can increase for complex CNFs and workload traffic profiles. Smarter estimation methods based on ML may reduce the estimation error at the cost of non-negligible increase of estimation delay. In [8], the authors only focus on the CPU utilization estimation and do not propose any energy saving mechanisms based on this estimation.

In comparison with HC approach, the X-MAN approach proposed in this Chapter can monitor the actual workload traffic on a much finer-grained basis of each individual CNF.

4.2.3 In-band Power Management with Code Instruction (CI)

Compared to out-of-band HC approach, CI approaches integrate software modules directly into the source code of CNFs programs to monitor the actual workload, which is used to further enable efficient power management. In contrast to the HC approach, the intrusive monitoring modules only require elementary counter increments when workload packets arrive. At contemporary Gigahertz processing frequencies, these counter are updated with some nanosecond latency. Therefore, the monitoring latencies introduced by CI approaches are relatively negligible.

The de facto high-performance user space packet processing library DPDK provides well-defined APIs for power management through adjusting P-states of the CPU core [9]. However, leveraging these APIs requires source code modification or patching of CNFs programs, so to the best of my knowledge, this approach has not been widely adopted by most popular NFV data plane frameworks. Partly because of this strict intrusiveness of the code instrumentation, several popular NFV data plane frameworks from both academia, e.g. Berkeley Extensible Software Switch (BESS) [86], FastClick [87], Libmoon [88], and Netbricks [89], as well as industry, e.g. VPP [90] and SampleVNF (from OPNFV) [91], do currently not employ these APIs for energy saving. Furthermore, the vanilla CI approach provided by DPDK performs only local power management by each CNF individually, which can easily lead to conflicts and instabilities when multiple CNFs are running on one CPU core simultaneously. This problem become much more challenging for cloud-native systems, because with the design of microservices, many CNFs are usually deployed to run simultaneously on a single physical CPU core. Running the power management module tightly inside each CNF requires an additional orchestration layer for the global optimization across all CNFs.

For the CI approach, an Adaptive Polling Mechanism (APM) is proposed in [92] to adjust the polling frequency based on the workload of incoming traffic. In this approach, the special `pause` instruction provided by Intel Streaming SIMD Extension 2 (SSE2) is used by APM modules to pause the polling of the default PMD of DPDK when a gap time between packets is detected. This mechanism can reduce the overall energy consumption.

At the time of this work, to the best of my knowledge, the most recent published research work on reducing energy consumption of high-performance packet frameworks such as DPDK is described in [83]. Li *et.al.* investigate the relationship between the average waiting time of an incoming packet in the buffer and the actual utilization of CPU. According to measurements perform in [83], the average waiting time of a packet reaches a cliff point and increases dramatically when the utilization of CPU exceeds 80%. Furthermore, the average idle periods for typical workload traffic are even shorter than the required transition time of C-states. Therefore, using C-states for power management with high responsiveness is impractical.

In contrast to the above introduced CI approach, the proposed X-MAN approach treat CNFs as black boxes and does not require any modification of the source code of the CNF programs. In comparison, X-MAN monitors the workload traffic directly with a lightweight in-band traffic monitor implemented with XDP which can be dynamically attached to the virtual interfaces of each CNF. Furthermore, in CALVIN, the

power management mechanism is performed through a separate user space module, which has a global view of all attached XDP traffic monitors for running CNFs deployed on a given CPU core. This provides a centralized power management and optimization for a CPU core without the need of an additional orchestration layer.

4.3 Proposed Approach: XDP-Monitoring energy-Adaptive Network functions (X-MAN)

4.3.1 X-MAN Design Imperative: Per-core Power Management Based on Per-CNF Traffic Monitoring

The design requirements of X-MAN are to adjust the frequency of each individual CPU core based on the packet traffic workload imposed on the CPU core by the individual CNFs deployed on the CPU core. The design of CALVIN considers the common COTS multi-CPU servers with multiple CPU cores in each CPU. For STOA CPU hardware, a CPU with multiple CPU cores is normally referred as one “CPU package”. For simplicity, in this dissertation, the terminology “CPU” is used to refer to “CPU package” and “CPU core” is used to refer a single core on a CPU package. In general, for servers with many CPUs (Multi-core processors), CALVIN runs independently for each individual CPU core. This work focuses on the common scenario with several CNFs deployed on a single CPU core, which is illustrated in Figure 4.2.

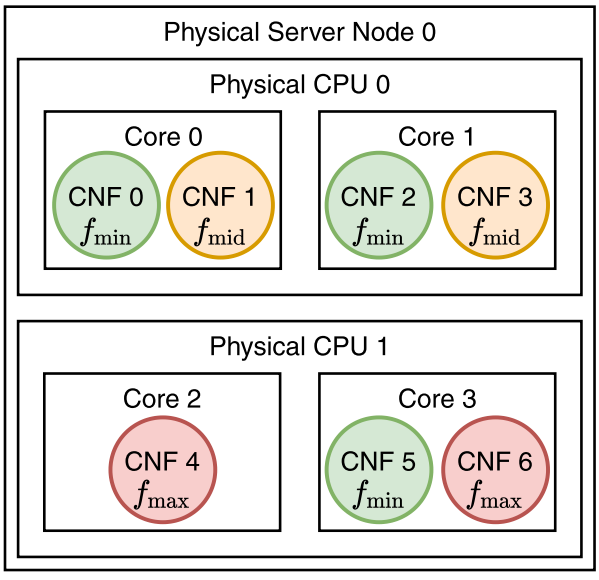


Figure 4.2: Example of the design of X-MAN for a physical server with two CPU packages. Reprinted from **my journal paper** [7].

To illustrate the important need for the X-MAN design, three different frequency levels for each CPU core are assumed as plotted in Figure 4.2: f_{min}, f_{mid} and f_{max}. Suppose that the f_{min} is the optimal frequency for low workload CNFs, f_{mid} is the optimum frequency for middle workload CNFs, and f_{max} is the optimal frequency for

high workload CNFs. For the scenario illustrated in Figure 4.2, in order to support running high-performance CNFs, the optimal frequency of both core 0 and core 1 should be scaled to f_{mid}. At the same time, the optimal frequency of core 2 and 3 should be scaled to f_{max}. It needs to be noted here all CNFs running on the same core share the current frequency of the underlying physical CPU. Therefore, the optimal required frequency of a given CPU core depends on the maximum value of required frequencies of all deployed CNFs on this CPU core.

Heterogeneous CNF workload intensity levels and CPU core frequencies exist in real-world cloud platforms. To align to this fact, A key design of the proposed X-MAN is to monitor the traffic workload of each individual CNF separately on a given shared CPU core. Based on this finer-grained traffic workload monitoring, which is achieved by the utilization of XDP technology in X-MAN, The frequency of each individual CPU core can be optimally managed by X-MAN. So in X-MAN the optimal working frequency of each individual CPU core is designed to be scaled separately and individually. For the scenario when numerous CNFs are assigned to a single given CPU core, as is very common with cloud-native networking deployments based on the microservice paradigm [93, 94], X-MAN is able to scale the CPU core frequency to the optimal frequency required to support all high-performance CNFs running on this core.

Compared to the proposed X-MAN, the HC approach described in [8] can only monitor the aggregated total workload of a single CPU core, which is generated by all deployed CNFs running on this given core. Therefore, compared to the finer-grained monitoring of X-MAN, a much coarser traffic monitoring granularity is provided by HC approach. For the scenario illustrated in Figure 4.2, the HC approach is not able to distinguish the actual workload of each CNF on core 0 and on core 2, when the low workload CNF (f_{min}) and the middle workload CNF (f_{mid}) add up directly to the high workload CNF workload (f_{max}). So when HC approach is used, the power manager would scale the frequency of core 0 to f_{max} even the f_{mid} is already sufficient here for more efficient energy consumption. All in all, X-MAN is able to avoid the unnecessary energy waste because of the over-scaling problem (exists in HC approach) by taking the actual workload of each individually CNF into consideration. Furthermore, compared to HC approach, which estimate the traffic workload indirectly through hardware counters [8], X-MAN is able to directly monitor the actual packet workload at all data plane network interfaces of each individual CNF.

At the time of this work, it is acknowledged here that some STOA CPU designs are limited in that the frequencies of the individual CPU cores on a given CPU package are automatically synchronized by default in the hardware or very low software level, thus allowing only packet frequency adaptation of a whole CPU package. However, new generation of server-oriented CPU designs increasingly allow independent per-CPU-core frequency management [95], and therefore can fully take advantage of the per-CPU frequency adaptation capabilities provided by X-MAN.

It should also be noted here that X-MAN approach considers a practical cloud-native cloud system operation where the system orchestrator deploy CNFs to available CPU cores (and migrates CNFs between CPU cores when needed) based on configured orchestration mechanisms, e.g. usually based on minimum CPU core workload or based on the basic round-robin CNF assignment to CPU cores in STOA

systems [96–98]. X-MAN treats the orchestration of CNFs to CPU cores as a given and strives to reduce the energy consumption of CPU cores by judiciously and dynamically adjusting the working frequency (P-state) of CPU cores. Future research direction can further explore the effective operating frequency and energy consumption characteristics of CNFs considered in the coordination mechanism to facilitate better energy consumption reduction.

4.3.2 X-MAN System Architecture: User Space Power Management Based on Kernel Space Traffic Monitors

Two main components of the X-MAN system architecture is illustrated in Figure 4.3: (i) Flexible and lightweight TMs implemented with XDP technology, which run in the Linux kernel space. (ii) An adaptive PM implemented as a separate program running in user space for power management with a global view. As shown in Figure 4.3, for a single given CPU core, the lightweight XDP traffic monitoring programs (namely, TM A and TM B) are attached to the physical Receive (RX) interface of compute node server. These TM programs perform lightweight traffic monitoring, collecting only the most important data required by the power management algorithm used by X-MAN. At the time of this work, implemented TMs count only the number of packets received and store this information on a per-CNF basis along with the accurate timestamp of the last packet received in the generic eBPF map data structure, which is shared by default with the PM running in user space. Thus, each interface in Figure 4.3 can be monitored by a dedicated XDP TM module with an associated eBPF map; Therefore, a typical CNF with ingress and egress interfaces has two eBPF maps to implement X-MAN with FeedBack (X-MAN-FB), which is a variant/extension of the vanilla X-MAN. TM performs no additional operations and (after counting) redirects/forwards all received packets (with the `XDP_REDIRECT` action provided by XDP) to the virtual interface of the corresponding CNF. The packets are then fetched into user space by the high-performance CNF via the efficient `AF_XDP`-based software PMD provided by the DPDK library.

In contrast to this lightweight X-MAN traffic monitoring, existing STOA traffic monitoring tools are mainly targeted at orthogonal use cases of that focused in this work, e.g. traffic monitoring for specific end-host or server applications, such as Packetbeat [99], or comprehensive packet and traffic monitoring, such as [100, 101]. Typically, to the best of my knowledge, these existing packet traffic monitoring tools use non-XDP packet sniffing mechanisms, such as memory-mapped packet sniffing in Packetbeat [102], which tend to struggle with practical real-world high packet traffic bit rates. Packet sniffing based on XDP can usually significantly improve the support for high packet traffic bit rates [103]. In addition, XDP-based packet traffic monitoring can be offloaded from the CPU to a physical NIC with the native XDP support, freeing up limited CPU resources. Therefore, XDP-based packet traffic monitoring is adopted in X-MAN.

For ultra-reliable low-latency advanced network functions, packets are processed in user space with DPDK. Accordingly, both the main and working DPDK logical cores (lcores) of a given CNF execute on a single CNF core (since it is common for containerized

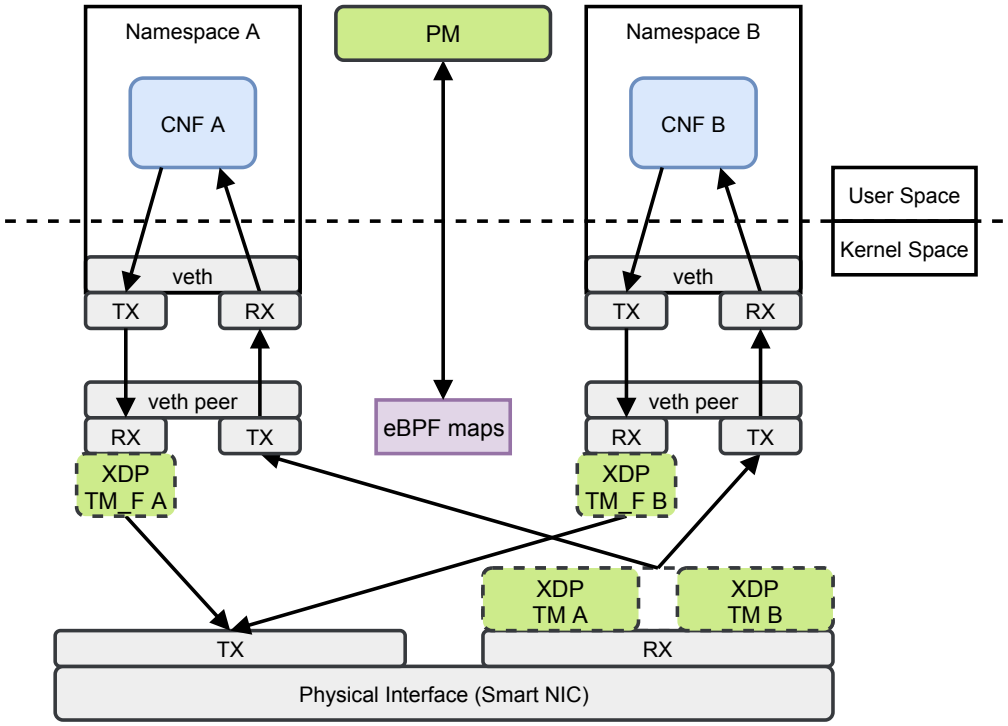


Figure 4.3: System architecture of the X-MAN power management for a given CPU core. Reprinted from **my journal paper** [7].

virtualization platforms to have a given container utilize only one CPU core, which avoids non-negligible communication delays between CPU cores). The processed packets are then forwarded to the virtual transmission (TX) interface of the CNF. The XDP program attached to the egress path forwards all packets to the TX ring of the physical NIC. In addition to the basic forwarding, an additional feedback mechanism can be implemented on the egress path to measure the actual processing overhead introduced by the CNF. This mechanism is used for the X-MAN-FB extension. The PM program running in user space has access to all the monitoring data stored in the eBPF map by default, as shown in Figure 4.2, for one CPU core. In general, X-MAN uses a user-space PM module to collect traffic information from multiple CNFs via eBPF maps so that the PM has a global view of all CNF running simultaneously on the CPU core. In X-MAN, only the user space PM needs the required privilege to adjust P- and C-states of the physical CPU cores. Based on the traffic monitoring data, and possibly additional feedback data, the PM is able to perform globally optimized power management.

The X-MAN architecture is designed to enable flexible management of TM modules within the Linux kernel while seamlessly supporting a high-performance user-space packet processing framework. In contrast to common architectures in industrial NFV data plane systems that use dedicated virtual switch components running in user space, such as VPP [90] and OVS-DPDK [104], the XDP-based in-kernel data plane is preferentially used to provide connectivity between CNFs. In the experimental Mizar project [105], a similar XDP-based in-kernel data plane architecture is used. The Mizar project concluded that the XDP-based fast in-kernel data plane architecture

Table 4.1: A summary of main notations used by X-MAN related modeling. Reprinted from my journal paper [7].

Symbol	Description
j	Number of received packets
S_j	Service time of a batch of j packets
c_{IO}	Number of CPU core cycles for I/O of a single packet
c_{task}	Number of CPU core cycles for CNF proc. of a single packet.
c_{call}	Number of CPU core cycles to invoke one batch handling function
c_{batch}	Total number of CPU core cycles to handle a batch of j packets
c_v	CPU core cycles for the empty polls in a batch
f_{CPU}	Frequency of the CPU core
V	Vocation time of the CPU core (for empty polls)
N_t	Number of received packets during time duration t
ρ	CPU core utilization of a CNF

significantly simplifies the programming of the data plane and reduces the management overhead of the switching and routing mechanism in a cloud-native environment.

X-MAN's unique and novel hybrid kernel-space and user-space architecture leverages the OS level virtualization provided by the Linux application container technology, where all containers running on the same OS share the same OS kernel by default. Since X-MAN TMs are running in kernel space, the packet traffic counters are kernel space data structures (implemented with the standard eBPF map data structure provided by the Linux kernel) and the workload traffic information of the counters can be read directly with ultra low overhead by the global X-MAN PM running in user space without any additional communication of control messages between the PM and the CNF. Since both the X-MAN TM and PM are completely transparent to the running CNF, the X-MAN does not need a dedicated user space orchestration layer with specified APIs to communicate with the CNF, for example, to poll monitoring statistics or perform power management (whereas such an orchestration is required for full user space packet processing approaches). Therefore, X-MAN is able to perform per-CNF traffic monitoring without interfering or interacting with the CNF in any way. It is acknowledged that the global in-kernel approach in X-MAN has several drawbacks, for example, it may cause significant security issues (which can be mitigated with introduction of a new secure eBPF map manipulation mechanism). All in all, the joint design of in-kernel and user space mechanisms is advocated to achieve a simple and very effective CNF management plane design.

4.3.3 Native X-MAN Adaptive Power Management

Service Time and Workload Model

$$S_j = \frac{j(c_{IO} + c_{task}) + c_{call}}{f_{CPU}} \tag{4.1}$$

$$V = \frac{c_v}{f_{CPU}}. \tag{4.2}$$

For effective power management without compromising CNF performance, especially the latency performance, X-MAN has to monitor a specific metric to accurately evaluate the actual CPU core workload imposed by the CNF. Based on the modelling work performed by Li *et.al.* [83], the service time S_j of a batch consisting of j packets and the vocation time V (The CPU time or cycles wasted on empty polls, namely a polling of zero packets) can be calculated with Equations 4.1 and 4.2. To simply the modelling, the numerator of the Equation 4.1 can be further concluded as $c_{batch} = j(c_{IO} + c_{task}) + c_{call}$. For a specialized given CNF implementation, the c_{IO} and c_{call} can be assumed as constants. The component c_{IO} generally depends only on the employed IO framework. This c_{IO} is as a first-order approximation independent of the specific CNF and the actual packet size when modern packet processing techniques such as zero-copying and data buffer pre-allocation are employed [9, 106, 107]. This component c_{IO} can be determined from the CNF data sheet or through practical benchmarking. The component c_{call} is relative small. The last item c_{task} depends on the configuration and function of the specific CNF. For instance, the advanced RLNC network function may have different c_{task} depending on the configured generation size and field size. In this work, the c_{task} is assumed to be determined from the data sheet provided by the CNF vendor or via direct benchmarking. The X-MAN-FB extension can estimate the c_{task} using its feedback mechanism. Therefore, for a given specific CNF, the service time of the CNF depends mainly only on the current CPU core f_{CPU}, namely the current P-state. For the proposed X-MAN power management mechanism, this means that the P-state of the CPU must be adjusted according to the incoming workload traffic.

In the work [83] performed by Li *et.al.*, the workload IP traffic is modelled as a Batch Markovian Arrival Process (BMAP). With this, each CNF in the system can be modelled as an $M^B/D/1/C$ queue. Based on this theoretical modelling, the inter-packet arrival time of workload traffic for each individual CNF can be modeled as an exponential distribution. Packets are served by a single queue (currently, most virtual interfaces do not support multiple queues by default) with a constant service time of S_j, see Equation 4.1. Therefore, the utilization of each CNF can be modeled as in Equation 4.3.

$$\rho = \frac{N_t \cdot c_{batch}}{t \cdot f_{CPU}}. \tag{4.3}$$

The utilization metric ρ indicates the relative workload intensity (namely, the busyness) imposed by the CNF on the CPU cores (running simultaneously at the current

CPU core frequency) and serves as the basis for developing the X-MAN employed algorithms.

According to the conclusions drawn from Li *et.al.* [83], when the ρ exceeds the 80% threshold, the performance of the CNF begins to degrade dramatically. Therefore, the target of the power management algorithm designed for CALVIN should keep the utilization ρ close to, but below, the threshold 80%. As described in Equation 4.3, ρ depends on the N_t, which is the total number of packets during a given time period t. In CALVIN, the power management algorithms prioritize the ultra-reliable low-latency performance over the net reduction of the energy consumption. Therefore, X-MAN is designed for real-time traffic, which does not tolerate any significant increase in latency due to energy saving mechanisms. The impact of X-MAN on packet latency should be negligible, which is verified in the following section.

Real-time Workload Prediction: Simple Moving Average (SMA) and Weighted Moving Average (WMA) Principle

To avoid accidentally exceeding the 80% threshold and to prioritize latency performance, the X-MAN power management algorithm needs to actively scale the CPU core frequency. This requires real-time prediction of workload traffic preferably without prior training and simple fast computation. Iqbal *et.al.* [108] performed a survey of statistical traffic prediction mechanisms for real-time applications, ranking these methods based on the evaluated accuracy and complexity. Unfortunately, the top-ranked method in [108] is a double exponential smoothing algorithm that requires parameter training for very similar traffic before practical deployment, as the smoothing algorithm will face similar traffic in real deployments. However, in the edge cloud, which is the focus of X-MAN, the workload traffic is highly dynamic. Therefore, it is significantly difficult to guarantee the quality and accuracy of the training data. The approach in the second position in [108] involves very complex computations. Therefore, we chose the third ranked method in [108], a moving average predictor. The moving average predictor has a negligible low complexity and achieves similar accuracy to its more sophisticated competitors in most test scenarios. Low complexity is highly prioritized in CALVIN algorithm design because low algorithm complexity allows for much more frequent updates.

The mean value over the last n_s sample values is calculated as a SMA for a one-step-ahead point forecast. For SMA, the accuracy of the prediction mainly depends on the number of considered samples n_s in each iteration. In order to make this prediction more accurate and robust, the SMA predictor is enhanced with an additional WMA predictor. Compared to simple SMA, the WMA assigns the more recent samples with a larger weight in the calculation. The combination of both SMA and WMA is able to predict the trend of the workload traffic [109]. Therefore, one-step prediction from the WMA, combined with a traffic trend (The trend can be an increase or decrease in the upcoming workload) learned from the comparison of the WMA and SMA, is able to provide reliable and punctual frequency adjustment decisions to be made.

In general, the number of samples used to calculate SMA and WMA in each iteration determines both the accuracy and responsiveness of the predictor. In the

algorithm used by CALVIN, n_s number of samples are used to calculate the SMA, while only half of the most recent samples, namely the $n_s/2$, are used to compute the WMA to improve the responsiveness of the predictor. The n_s is configurable parameter and is set to 50 in this work based on preliminary empirical measurements on my testbed.

Real-time Workload Prediction: Detailed Implementation

Algorithm 1: Main management loop of the native X-MAN. Reprinted from my journal paper [7].

```
1  c_task = Number of CPU core cycles to process a packet;
2  T_m = Time interval for one iteration of the power management loop;
3  i = Index of the power management interval;
4  p_i = Total number of received packets through the beginning of interval i (from kernel eBPF);
5  t_{s,i} = Timestamp for received packets;
6  n_z = Counter of polling rounds with zero received packets in current management interval;
7  n_u = Counter of scaling up hints;
8  n_d = Counter of scaling down hints;
9  f_{cpu,min}, f_{cpu,max} = Minimum and maximum supported CPU core frequency;
10 n_s = Number of samples to calculate ρ;
11 ρ̄_SMA, ρ̄_WMA = SMA and WMA of n_s samples of ρ;
12
13 p_0, t_{s,0}, n_z, n_u, n_d = 0;
14 i = 1;
15 n_s = 50;
16 f_{cpu,0}, f_{cpu,1} = f_{cpu,max};
17 need_scale = false;
   /* The power manager makes a decision every T_m seconds.                    */
18 while true do
19     power_management_timer.start(T_m);
20     p_i, t_{s,i} ← read_ebpf_map();
21     ρ_i ← ((p_i − p_{i−1}) * c_task)/((t_{s,i} − t_{s,i−1}) * f_{cpu,i});
22     if p_i == 0 then
23         n_z ← n_z + 1;
24         if n_z ⩾ n_{z,max} then
25             f_{cpu,i+1} ← f_{cpu,min};
26             scale_to_freq(f_{cpu,i+1});
27             freq_is_min = true
28         end
29     else
30         n_z ← 0;
31         if freq_is_min = true then
               /* Fast scaling to max. freq. to avoid pkt. loss                */
32             f_{cpu,i+1} ← f_{cpu,max};
33             scale_to_freq(f_{cpu,i+1});
34         end
35         n_u, n_d ← update_traffic_trend(ρ̄_SMA, ρ̄_WMA, ρ_i, n_s);
           /* Check if frequency scaling is necessary                          */
36         need_scale ← get_scale_decision(n_u, n_d);
37         if need_scale == true then
38             f_{cpu,i+1} ← get_target_freq();
39             scale_to_freq(f_{cpu,i+1});
40         end
41     end
42     i ← i + 1;
43     power_management_timer.reset(T_m);
44 end
```

As explained above, the combined SMA-WMA predictor is implemented in X-MAN system as a user space program using my home-grown packet high-performance packet processing library:Fast Forward Packet Processing (FFPP). The pseudocode of the core power management loop, which runs inside the user space power manager program for all CNFs on one given CPU core, is described in the Algorithm 1. As presented in the Algorithm 1, the X-MAN periodically (with a configurable duration) reads workload traffic information, in particular the number p_i (read from the cumulative counter provided by the Linux kernel eBPF mechanism) of all received packets up to the beginning of the current management interval i and the timestamp $t_{s,i}$ of the latest received packet in the previous management interval, from the eBPF map. (This traffic information (p_i, $t_{s,i}$) is updated by the attached XDP program for each incoming individual packet). Each management interval consists of several steps, including: (i) Read the monitoring data from the eBPF map. (ii) Calculate the CPU utilization and check if a frequency adjustment (namely, an adjustment of current P-state) is required. (iii) If an adjustment is needed, perform the corresponded frequency adjustment.

The traffic information (p_i, $t_{s,i}$) is read only at the beginning of each power management iteration. In particular, at the beginning of the i-th management interval, the traffic information (p_i, $t_{s,i}$) is read from the eBPF map. This eBPF map information is then used to make a power management decision along with the information for the previous interval $i-1$, namely (p_{i-1}, $t_{s,i-1}$). The power manager process then sleeps until the start of the subsequent power management interval $i+1$ and this periodic management process is repeated.

The timestamp $t_{s,i}$ (together with the timestamp $t_{s,i-1}$ of the previous interval) is used to calculate the exact length of time between two read moments in two consecutive management intervals (time period t in Equation 4.3). In theory, for a strictly real-time system and negligible overhead of the management function, the read interval should be T_m. However, the implementation on the COTS server requires this timestamp mechanism to accurately calculate the exact time period t. Based on the count p_i of the total number of packets received during time period t and up to the beginning of the current management interval i (used to calculate the number of packets received during time period t: N_t), the CNF utilization ρ of the given CPU core is calculated according to Equation 4.3.

Based on empirical tests with the practical testbed used in this study, the power management interval to T_m = 1 ms. In general, the power management interval T_m should be configured according to the characteristics of the underlying cloud infrastructure.

X-MAN Power Management: CPU Core Frequency Scaling

In the next step, the time series values of CPU utilization $\bar{\rho}_{SMA}$ and $\bar{\rho}_{WMA}$ are computed by the `update_traffic_trend` function on line 35 of Algorithm 1 and compared to each other to see if the traffic is currently following a trend. This function then checks whether $\bar{\rho}_{WMA}$ is above ρ_{up} or below ρ_{down}. If so, an up counter n_u or a down counter n_d is increased, respectively, while the other counter is set to zero.

Inspired by the process that led to the initial parameter setting of the TCP [110], following algorithm parameters are configured based on an intuitive understanding of frequency scaling and pilot experiments.

Two thresholds ρ_{up} and ρ_{down} are configured to 75% and 65%. The upper threshold ρ_{up} to a value below 80% to ensure that the CPU utilization of the CNF stay below the 80% threshold. The ρ_{down} is needed to determine the situation when scaling down is required. There is a 10% gap between these the two thresholds, leaving some room for fluctuating values and preventing too rapid down-scaling. In an up-trend, the up counter is increased by 3 and in a down-trend, the down counter is increased by 2. Finally, both up and down counters are checked against the counter threshold and if the counter threshold is exceeded, a scaling of the CPU core frequency is induced (in lines 37-39), which is determined by the `get_scale_decision()` function in line 36. The counter threshold depends on the P-state transition time and the eBPF map read interval T_m. It should not be possible for the counter to exceed the threshold earlier than the transition time has elapsed. If the calculated frequency is between two available P-states, the P-state with the higher frequency is selected. The management of the P-states is performed using the APIs provided in the `rte_power` library provided by DPDK. The previous P-state determination is performed for each CNF running on a given CPU core during each power management interval of duration T_m. Then, the P-state of the CPU core is set to the highest frequency of an available P-state determined for the CNF on the CPU core. Thus, the computational complexity of the X-MAN power management is linearly related to the number of CNFs running simultaneously on the CPU core.

In addition to the utilization prediction algorithm presented, the native X-MAN power management algorithm has some additional features: (i) Immediately after some consecutive empty readings, it scales down to the minimum available frequency (lines 24-27 in the Algorithm 1). (ii) After this scaling down, the map read interval is automatically shortened to T_m = 100 μs (from the default 1 ms). This allows the PM to quickly detect a new packet burst for timely packet processing. As soon as a new packet is detected, the CPU frequency is immediately increased to the maximum $f_{cpu,max}$ (lines 31-33) to avoid any loss of packets or increase in latency.

4.3.4 X-MAN Extensions

Beside the native X-MAN power management algorithm introduced above, two extensions are also implemented and evaluated in this work:

X-MAN with C1 State Management (X-MAN-C1) In addition to P-state management, the X-MAN-C1 extension also includes the use of C-state to further significantly save energy during idle time. While the native X-MAN scales down to $f_{cpu,min}$ when a long idle time is detected, X-MAN-C1 is able to suspend CNF execution by manually guiding the CPU into the C_1 sleep state. Since it is currently not possible to directly manage the C state of the CPU in user space, X-MAN-C1 is implemented with a simple client-server model. The native X-MAN is implemented as a client, while the suspend request is sent to the server via a `ZeroMQ` socket. The server-side implementation

functions by changing the `cpu_quota` parameter of the container running the CNF. This change in the *cpu_quota* parameter effectively signals the kernel to enter C_1 state.

X-MAN with FeedBack (X-MAN-FB) The native X-MAN requires detailed specification and benchmark data of the deployed CNF, such as the number of CPU core cycles, namely c_{task}, used for CNF packet processing. X-MAN-FB aims to address this limitation and is designed to enable fully self-regulating power management. The native X-MAN is extended by X-MAN-FB to have feedback about its scaling decisions. A suitable feedback is the packet count from the CNF egress interface, as illustrated in Figure 4.3. If the CNF is not overloaded, the packet counts of the ingress and egress interfaces should be the same, and if the CNF is overloaded, they will be off. These two counters are constantly compared. An algorithm based on Additive-Increase Multiplicative-Decrease (AIMD) is performed on the comparison results. X-MAN-FB iteratively detects the minimum possible frequency until there is a loss in the CNF, namely, until the ingress and egress interface counters are different. The scaling mechanism of X-MAN-FB follows the basic principles of TCP congestion avoidance algorithms [111]. If no packet loss is detected, the P-state index is increased by 1 at each iteration to scale down the CPU core frequency. As soon as any loss occurs, the current P-state index is immediately halved, which leads to an immediate increase in frequency.

4.4 Performance Evaluation Setup for X-MAN

4.4.1 Testbed for X-MAN Evaluation

Similar to the testbed used to evaluate CALVIN introduce in Chapter 3, the evaluation testbed for X-MAN consisted of two physical COTS servers connected back-to-back. One server is used as the Packet Generator (PacketGen) and the other server is used as the Device under Test (DuT) on which the CNF is deployed. PacketGen runs on an Intel Core i7-6700 CPU@3.4 GHz, while DuT is equipped with two Intel XEON CPUs E5-2643@3.30 GHz, i.e. DuT has two CPUs with four cores each. Both PacketGen and DuT are equipped with a Mellanox ConnectX-5 EN 25 GbE dual-port SmartNIC and 32 GB of memory. PacketGen and DuT are connected with 25 GbE Small Form-factor Pluggable (SFP) cables.

Cisco `TRex` (v2.81) [112] is used in this work, which is an industrial-grade high-performance packet traffic generator and monitor, to generate packet traffic in stateless mode for different workload traffic profiles and measure latency performance results. Ubuntu Server 20.04 LTS (With Linux kernel version 5.4) is used as the host OS for all servers. Docker 19.03-CE is used for application container management. For the data plane, DPDK (v20.02) and XDP-Tools (v0.0.3) [55] are used to develop CNF and Power Manager (PM) of X-MAN. The software tool `turbostat` [113] is used to measure CPU power and energy consumption. Based on the measurements performed in [114], `Turbostat` provided results comparable to those obtained using

additional physical measurement equipment, which is less portable and much more expensive. Hyperthreading and Basic Input/Output System (BIOS) power management features on DuT are fully disabled in order to obtain both consistent and repeatable measurements.

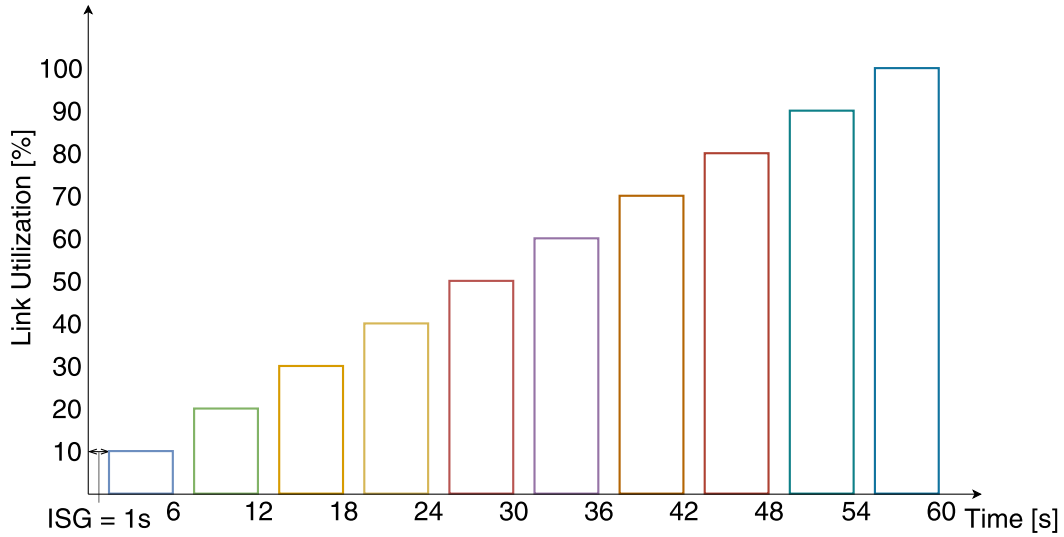
To prevent background processes from interfering with the actual energy measurements, the physical CPU cores that are used to run CNFs and the X-MAN PM are isolated from the Linux OS by using the `isolcpus` kernel parameter to remove these cores from the Linux kernel scheduler.

4.4.2 Workload Traffic Profiles

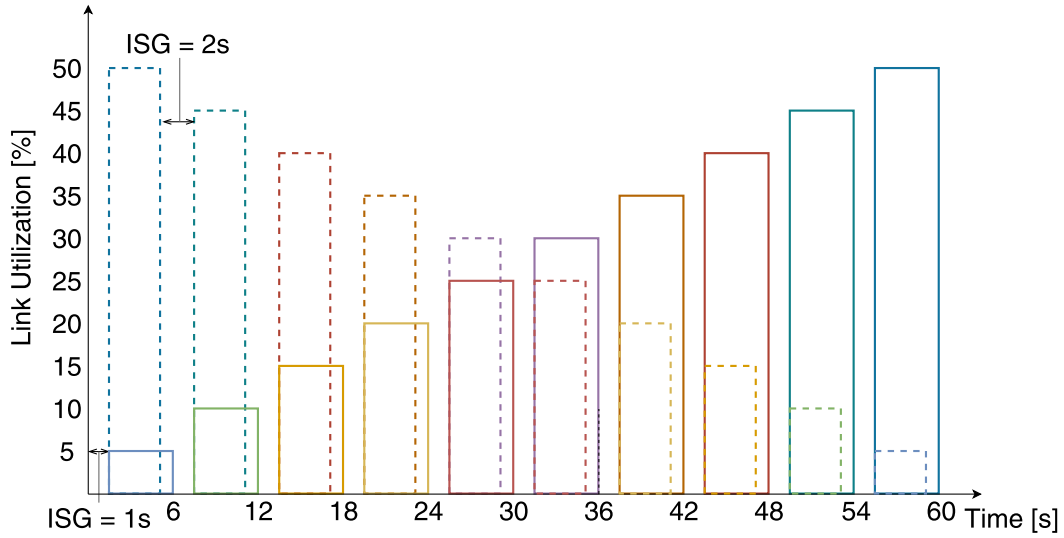
Two types of probing workload traffic is generated by TRex to evaluate the performance of X-MAN and compare it with the STOA existing approaches.

Deterministic Traffic A deterministic ON/OFF traffic profile is used mainly for deterministic and reproducible measurements. As shown in Figure 4.4, the deterministic traffic profile produces packet traffic that alternates between packet flows (or called packet trains or packet bursts) with a fixed packet arrival time (5 seconds ON-period) and an Inter-Stream Gap (ISG) (1 second OFF-period). During the ON-period (i.e., during a given burst of packet traffic), the Packet-per Second (PPS) rate corresponds to a specified link utilization from 10% to 100% of the underlying 10 Gbps physical network for a single CNF evaluation. This stepped traffic pattern (as plotted as solid lines in Figure 4.4a) is suitable for examining how the applied power management mechanism adapts to different PPS and uses ISG to reduce the overall energy consumption. In order to evaluate the designed global power management provided by X-MAN, the step-up pattern above is supplemented with a corresponding step-down traffic pattern (as plotted as dashed lines in Figure 4.4b) to evaluate the scenario with two running CNFs. For the step-down traffic pattern, the ON cycle is reduced to 4 seconds and the ISG is extended to 2 seconds. The step-up and step-down traffic profiles simulate a scenario where two separate flows with different traffic trends are sent to two separate CNFs deployed on DuT. Since the maximum bit rate is configured to 10 Gbps, each traffic pattern in the two CNFs evaluations is limited to a maximum bit rate of 5 Gbps. The multi-burst flow profile of the stateless mode provided by TRex is used to generate these deterministic flow profile packets. Each burst or packet train contains several UDP packets with a fixed IP datagram length of 1400 B (namely, 1400 bytes).

Random Traffic This profile is mainly used to test the robustness and resilience of X-MAN to relatively more realistic workload traffic. In the random traffic profile, packet streams have a random exponential distribution of arrival times (i.e. random ISGs). Instead of a fixed packet size used for deterministic traffic, the packet size distribution in each packet traffic burst follows the Internet Mix (IMIX) genome [115] (53.84% 64 B, 38.46% 570 B and 7.49% 1400 B). The PPS rate is fixed to maintain a 30% link utilization. Compared to the deterministic traffic profile, more than 90% of



(a) For a single CNF.



(b) For two CNFs.

Figure 4.4: Deterministic probing traffic profiles for X-MAN benchmark. Reprinted from **my journal paper** [7].

ISGs are less than 0.5 second. So this traffic profile requires relatively more granular traffic monitoring.

Both model-based deterministic and random active probing traffic profiles are used in this work because they could cover a wide range of traffic dynamics and are reproducible. Therefore, these profiles enable a rigorous evaluation.

4.4.3 CNF Deployment

Two scenarios are deployed for the measurements:

One single CNF A single CNF is deployed on one dedicated isolated CPU core. The CNF program runs inside a Docker container with two data plane virtual network interfaces. The `AF_XDP` PMD acts as the packet IO mechanism for the CNF and the in-kernel XDP forwarder to redirect workload traffic from the physical NIC to the corresponding virtual Ethernet pair (`veth-pair`). This basic deployment is used to evaluate the baseline performance of the X-MAN.

Two independent CNFs In this scenario, two independent CNFs are deployed simultaneously on two CPU cores. The in-kernel XDP forwarder forwards the workload traffic to corresponding CNF based on the pre-configured metadata of the traffic flow.

In both cases, the built-in L2 forwarding (L2FWD) example application provided by DPDK library is extended and then used as the basic function of each CNF. Other more complex CNFs can be built on top of this elementary forwarding application. Since the focus of this work is to evaluate the performance of different power management mechanisms, the elementary forwarding CNF is considered without loss of generality.

4.4.4 Monitoring Latency for CPU Utilization Estimation

CPU utilization estimation is required for all common power management mechanisms. Due to some limitations, the power measurement and corresponding algorithms are not researched in the paper [8]. However, it provides a black-box utilization estimation method based on hardware counters.

The monitoring latency is defined in this work as the time period required for the power management system to obtain a new sample of the current workload status. Specifically, in X-MAN, the monitoring latency is equal to the time to execute line 20 of Algorithm 1, which is the read time of the eBPF map. The comparison for this monitoring latency is performed between the HC approach described in [8] and the X-MAN.

Hardware Counter (HC) utilization estimation HC utilization estimation uses CPU hardware counters to estimate the workload of deployed NFs [8]. This HC-based approach has a strict limitation that it only works when each CNF is assigned to an isolated physical CPU core. If not, other NF processes or background processes can affect the hardware counters, resulting in inaccurate or even incorrect estimations. This assignment of a NF to an isolated physical CPU core is very expensive and generally unacceptable for cloud-native network function systems. For the HC mechanism, up to four CPU cores can be monitored on the testbed to deploy up to four CNFs.

X-MAN Utilization estimation Compared to HC approach, X-MAN does not have above described limitations. Traffic monitors of X-MAN are very lightweight and can be directly attached to the network interfaces of deployed CNFs, regardless of whether they are assigned to the same or different CPU cores. Assuming two data

plane network interfaces per NF, X-MAN has to monitor eight network interfaces to meet the same scenario for four CNFs of the HC approach. Therefore, in fact, in the evaluation of the monitoring latency, the monitoring of four physical CPU cores (for the HC approach) and four network interface pairs (namely, eight data plane network interfaces for the X-MAN approach), is compared. In addition, because the number of network interfaces monitored by X-MAN is theoretically not limited by the number of available physical CPU cores, the scalability of X-MAN monitoring is evaluated for different numbers of virtual network interfaces. Specifically, the monitoring latencies for 10, 100 and 1000 Linux veth pairs [116] are measured.

4.4.5 Power Management Mechanisms

Following power management mechanisms are used to compare the proposed X-MAN with the most recent and advanced CI mechanisms available from the DPDK community, as well as between X-MAN extensions.

- NPM: This scenario does not perform any power management mechanisms. Thus, this NPM scenario is used as the baseline for the best latency performance and the worst energy consumption results on the deployed testbed.
- X-MAN: The native X-MAN power management without using C_1 state or any feedback mechanisms.
- X-MAN-C1: Native X-MAN approach plus the additional management of C_1 state.
- X-MAN-FB: Native X-MAN approach with the additional feedback mechanism.
- Code Instruction with Heuristic power management (CIH) [9]: This approach is used in the official sample application for power management provided by the DPDK library. This CIH approach is used as one representative design and implementation of the CI based approaches.

It is clearly noted that HC approach is not in this list of power management mechanisms, because HC approach described in [8] is only a utilization estimation method. In contrast to HC approach, both X-MAN and CIH are complete solutions for both CPU utilization estimation and power management.

4.4.6 X-MAN Performance Metrics

- Monitoring latency required for CPU utilization estimation: For this metric, only X-MAN and HC approaches are compared. For all other metrics, the X-MAN is compared against NPM, CIH and also two variants of X-MAN.
- Optimal CPU frequency and power for deterministic traffic profiles: Since the timing information of the ON/OFF periods used is fully determined, according to [83], the optimal value of both CPU frequency and the corresponding power value can be calculated based on the preliminary empirical measurements. The

optimal power value is used as a lower bound (the best accessible lowest energy consumption) for all power management mechanisms.

- Average CPU frequency and power (namely, energy consumption): The average CPU frequency and power values of all physical CPUs on the DuT are measured with the widely used `Turbostat` tool with its highest available measurement resolutions. The measurement values include per-CPU package power value, per-CPU core frequency, and also per-CPU core C_1 state residency time.
- Bandwidth accessed indirectly with number of lost packets: Because `TRex` [112] (v2.81) does not provide throughput values directly, the throughput values are measured indirectly. Bandwidth or throughput is equal to the maximum sending rate without any dropped and out-of-order packets. Therefore, the number of packets dropped in each packet stream is measured to indirectly evaluate the throughput performance.
- End-to-end latency: For this work, the average, maximum and jitter of the end-to-end RTT of each packet stream are used as key latency metrics. The maximum latency value and jitter allow evaluating both the worst-case delay performance and the delay inconsistency between packets.

For each measurement setup and scenario, 100 independent replications are performed for statistical results. At least 10 packet streams (namely, packet bursts or trains. The documentation of `Trex` uses the term "streams") are used in each measurement replication. Each metric introduced above is sampled for each `TRex` stream in each replication.

4.5 X-MAN Measurement Results and Evaluation

4.5.1 X-MAN CPU Measurements

To get the detailed characteristics of the used physical CPU of the DuT server, some preliminary measurements are needed.

The actual power required by the CPU package in different P-states is illustrated in Figure 4.5. The matrix product workload provided by the tool `stress-ng` [117] is executed on the CPU for 1 second before switching to the next P-state. This switching can be performed in both increasing and decreasing directions with two common widely used scaling drivers: `intel_pstate` for Intel x86 CPUs and `acpi-cpufreq` for both AMD and Intel x86 CPUs. The `intel_pstate` driver is highly optimized for the latest Intel CPUs. It can be observed from Figure 4.5 that the power values in each P-state is independent of the scaling driver and the previous P-state. The `intel_pstate` driver is used on my testbed in the following evaluation because it is the default driver used by most Linux distributions today.

The temperature in °C of the CPU package in different CPU states are listed in Table 4.2. The results show that increasing the CPU frequency leads to a significant increase in CPU temperature, almost 30 °C between the C_1 and P_0 states. High CPU

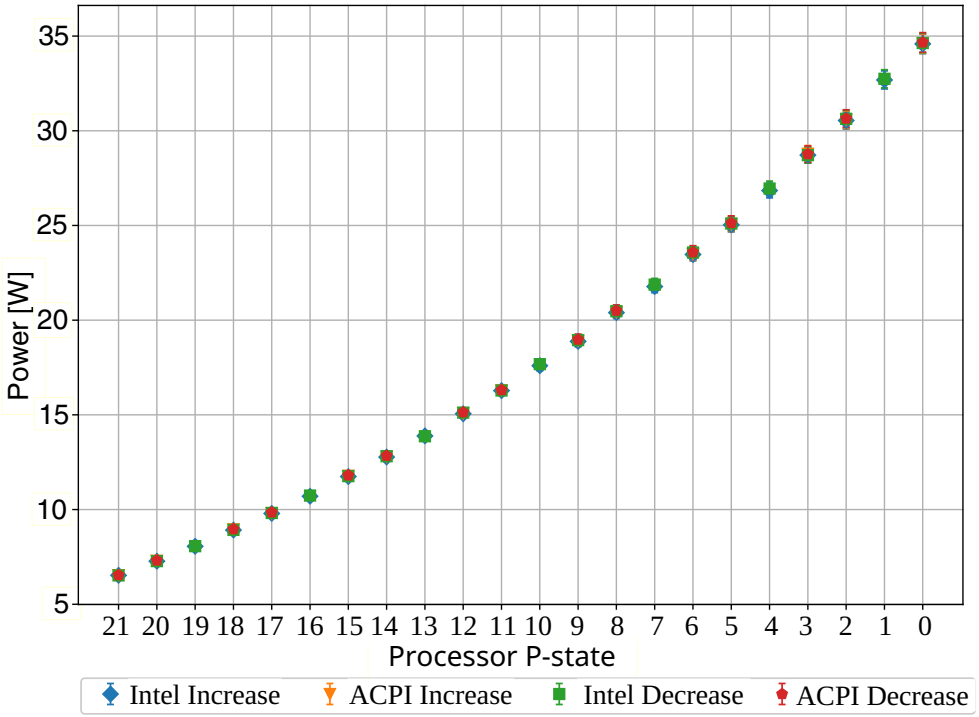


Figure 4.5: CPU frequency increase and decrease test. Reprinted from **my journal paper** [7].

Table 4.2: CPU package temperature for different CPU states. Reprinted from **my journal paper** [7].

State	C_1	P_{21}	P_1	P_0
Temperature [°C]	42.2	44.6	68.5	71.1

temperatures have a significantly negative impact on the energy efficiency of the system (since the required cooling process incurs non-negligible energy costs) and can significantly shorten the CPU lifetime.

Another important CPU measurement performed is to evaluate the minimum time duration (namely, the sojourn time) that the CPU should stay in a P-state before changing to another P-state. A so-called Tic-Toc-test is performed in this work to periodically switch between the highest and lowest available CPU frequency with different periods, where this period contains two P-state sojourn times. The average CPU frequency and power values as a function of the period duration are illustrated in Figure 4.6. For this measurement, each scenario is replicated for 50 times. As present in Figure 4.6, when the switching period of P-states is shorter than about 30 ms, the effective CPU frequency starts to drop dramatically while the CPU starts to consume significantly more energy. So the period for frequency scaling is configured to be higher than 30 ms on my testbed to avoid issue of over-scaling illustrated in the Figure 4.6.

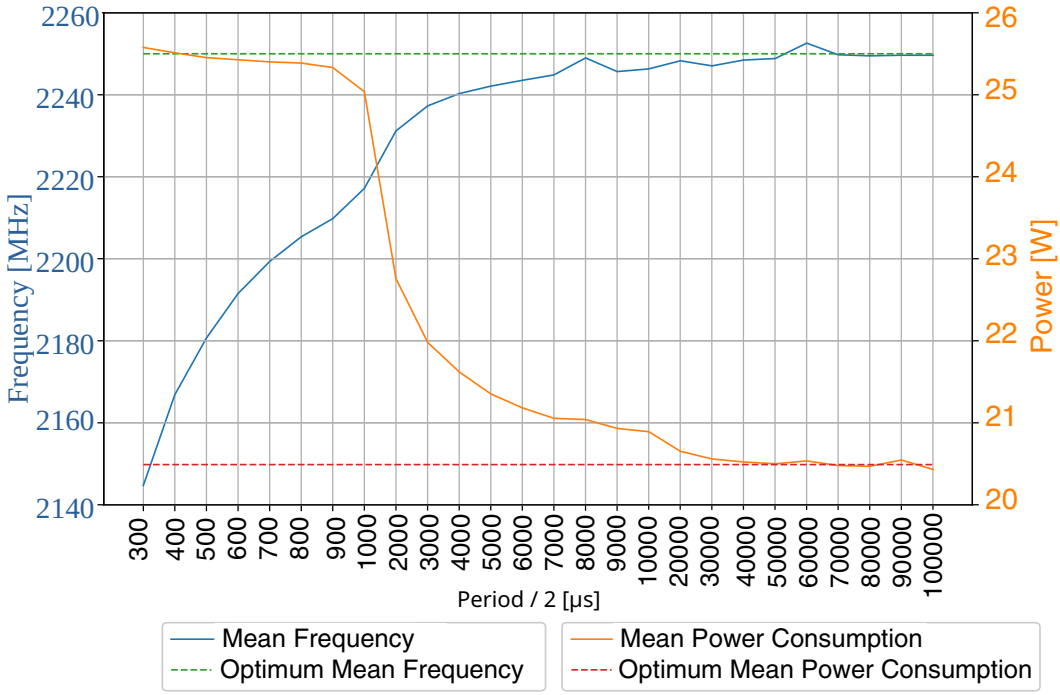


Figure 4.6: Tic-Toc test for CPU frequency and power. Reprinted from **my journal paper** [7].

4.5.2 Monitoring Latency for CPU Utilization Estimation

The comparison of the Cumulative Distribution Function (CDF) of monitoring latency between HC approach and the proposed X-MAN is illustrated in Figure 4.7. Monitoring latencies are measured when DuT is in IDLE state or stressed with 100% workload via the `stress-ng` tool. For each scenario, the measurements are replicated independently for 100 times. These two scenarios are evaluated here for the best and worst cases regarding the monitoring latency. According to the measurement results, X-MAN can achieve a significantly shorter and much more stable monitoring latency compared with the HC approach. It can be observed that the monitoring latency of the HC approach is affected by the workload of the DuT and shows a large variance when CPU is idle. The HC approach takes nearly 5 times more time than X-MAN for latency monitoring for the likelihood of 80%. For the scenario of 100% CPU workload, the HC approach normally needs about 5 times longer monitoring latency than the X-MAN with nearly 100% likelihood. Due to the lightweight and efficient XDP-based traffic monitoring mechanism, the monitoring latency of X-MAN is always less than about 10 μs for arbitrary workload.

The scalability and stability of the X-MAN monitoring latency are evaluated and illustrated in Figure 4.8. As presented in the Figure, the monitoring latency of X-MAN shows a linear relationship with the deployed number of parallel veth pairs, namely data plane virtual network interfaces. And this behavior is not affected by the CPU

workload. All in all, X-MAN is able to provide low-latency, efficient, scalable and robust CPU utilization estimation.

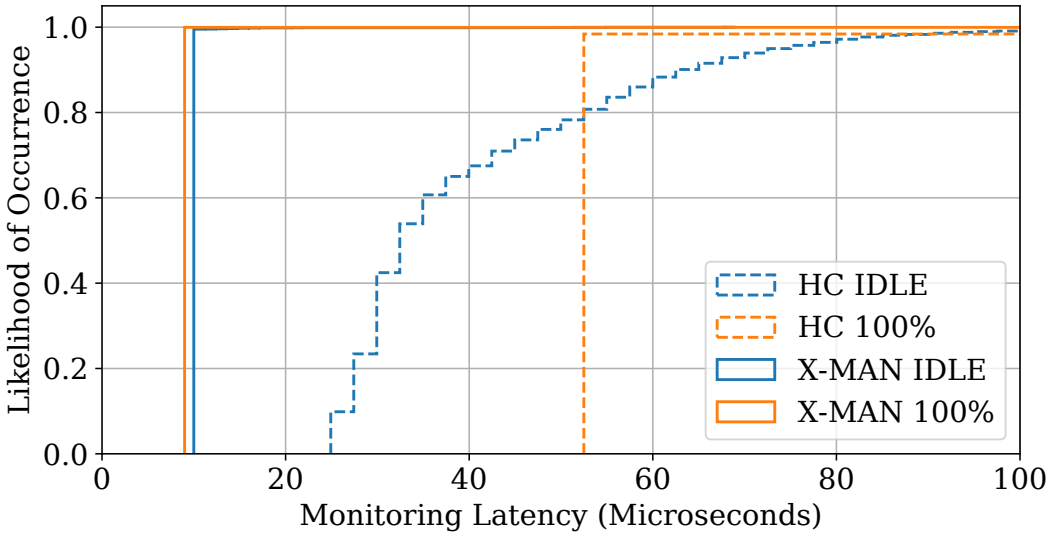


Figure 4.7: Comparison of monitoring latency between HC approach and X-MAN. Reprinted from my journal paper [7].

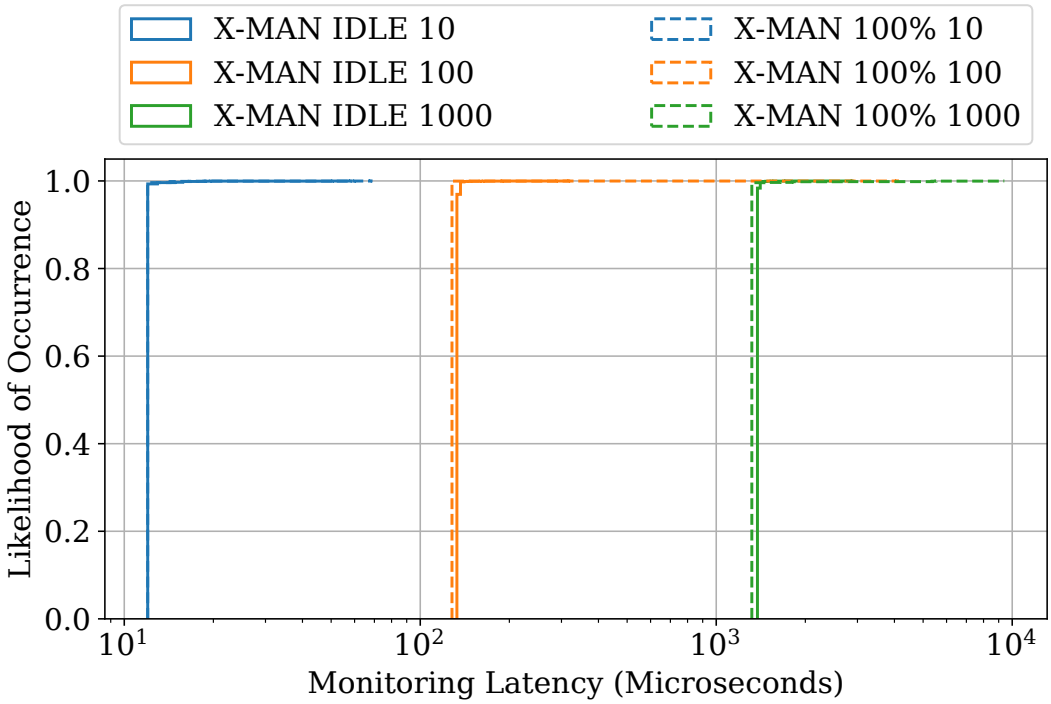


Figure 4.8: Monitoring latency of X-MAN for different number of veth pairs. Reprinted from my journal paper [7].

4.5.3 Single CNF with Deterministic Traffic

Figure 4.9 shows the average frequency and power values of the CPU on DuT for a single CNF deterministic traffic scenario. It can be observed from Figure 4.9 that the CNF achieves full utilization of the CPU cores at link utilization levels of 40% and 50% (as shown for optimal frequencies up to 3.3 GHz). Therefore, there is an opportunity for an adaptive intra-stream (namely, inside each packet stream) energy saving when the link utilization is 40% or lower. By examining the Figure 4.9 more closely, it can be concluded that for the 20% link utilization, the X-MAN power overlaps with the X-MAN-C1 and optimal power values; In addition, the X-MAN-FB exhibits relatively high energy consumption at 20%, which is consistent across 100 independent measurement replications. It can be observed from the Figure 4.9 that the CPU core frequencies of X-MAN and X-MAN-C1 are always close to the theoretical optimum (in fact, they usually overlap with each other and mask the optimum symbols in the Figure 4.9) during ON periods when the link utilization is 40% or lower. At the same time, the CIH performs very similarly to the NPM, namely mostly being at the maximum frequency during ON time. In comparison, for ON periods where the link utilization is 30%, native X-MAN almost is able to halve the energy consumption. While for ON times when the link utilization is 40%, X-MAN-FB can reduce the power consumption by about a quarter compared to CIH and NPM approaches. In addition, it can be observed from the Figure that during the OFF periods, the CIH cannot reduce the frequency in time, leaving the average CPU frequency only slightly below 3 GHz. At the same time, both X-MAN and X-MAN-FB reduce the average CPU frequency to well below 2 GHz (usually about 1.3 GHz). Furthermore, X-MAN and X-MAN-FB reduce the CPU frequency to a minimum of 1.2 GHz, and the deviation of the average CPU frequency from 1.2 GHz originates from the first sample when the ISG is about to be detected but the CPU is still at the ON period frequency. X-MAN-C1 reduces the average CPU frequency below 1 GHz during most OFF periods and is able to approach the optimal CPU frequency of zero. In general, the native X-MAN approach and its extensions reduce the energy consumption to one-quarter ($1/4$) of the NPM energy consumption during the shutdown period, while CIH only reduces the power consumption to three-quarters ($3/4$) of the NPM energy consumption.

The end-to-end RTT latency results are listed in the Table 4.3. The results listed in the Table presents the increases in the RTT latency due to additional monitoring and power management mechanisms introduced by X-MAN based and CIH approaches compared to the baseline NPM. It can be observed from the Table that X-MAN, X-MAN-FB and CIH approaches cause only negligible and slight increases in the latency performance. In comparison, the X-MAN-C1 can even double the latencies in some scenarios. The low latency overhead provided by X-MAN is mainly due to the judicious use of the native Linux kernel NIC driver (Linux kernel version 5.4 provides native support for veth network interfaces), which is the fastest mode offered by XDP. Therefore, it can be concluded that the traffic monitoring and adaptive power management (CPU core frequency scaling) algorithms of X-MAN and X-MAN-FB introduce only negligible additional latency overhead. At the same time, the large increase in latency of X-MAN-C1 in Table 4.3 and the minimal additional energy savings of X-MAN-C1 compared to X-MAN and X-MAN-FB lead to the conclusion that exploiting

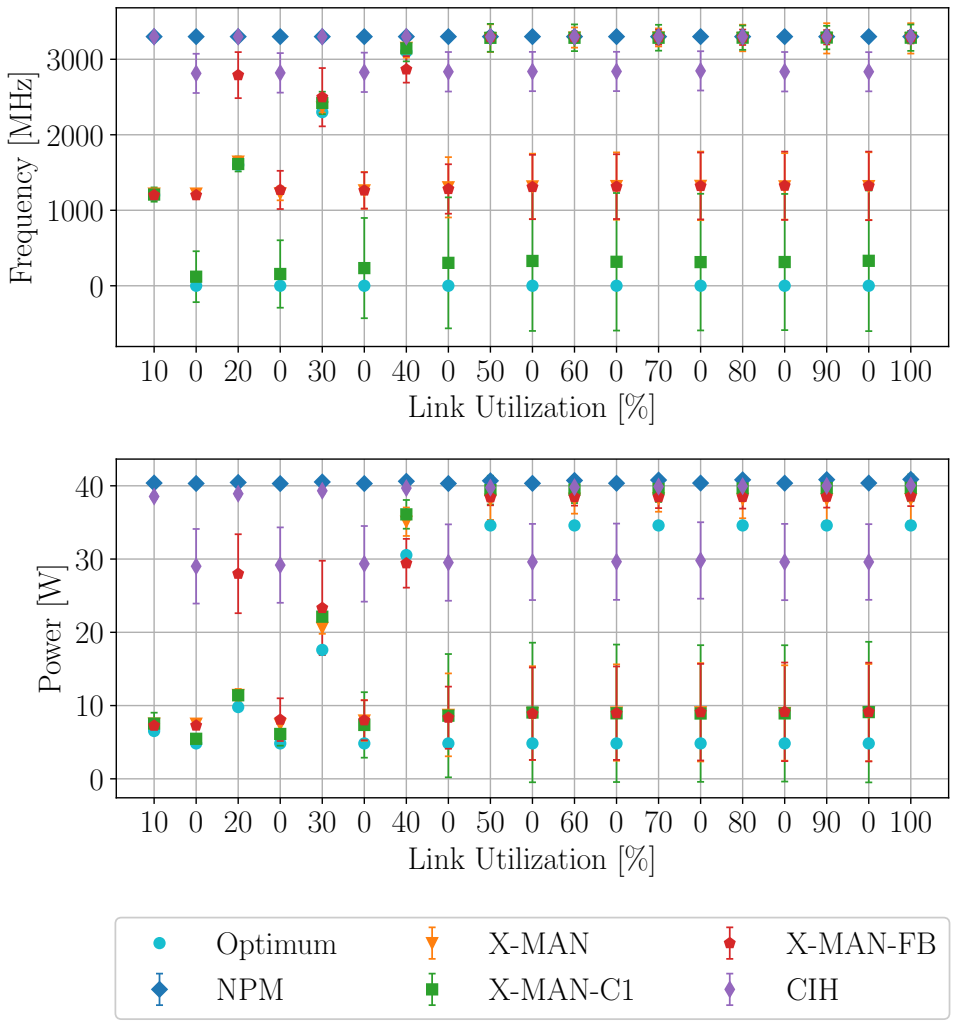


Figure 4.9: Average CPU frequency and power values of a single CNF as a function of the link utilization for deterministic traffic. Reprinted from **my journal paper** [7].

the C-state does not usually pay off, which confirms the conclusions summarized already by Li *et.al.* [83].

With above listed results and evaluations, it can be concluded that native X-MAN and X-MAN-FB are able to significantly reduce the energy consumption at both ON and OFF times with negligible latency performance overhead compared to the NPM and CIH approaches.

4.5.4 Two CNFs with Deterministic Traffic

The evaluation of the two simultaneously deployed CNFs in this Subsection is intended to examine the performance of X-MAN in the scenario with multiple inde-

Table 4.3: Single CNF latency results for the deterministic traffic. Latency increases are listed as percentage with respect to the performance of the baseline NPM approach. Reprinted from **my journal paper** [7].

Link Utilization (%)	10	20	30	40	50	60	70	80	90	100
Average Latency (%)										
X-MAN	0.00	0.01	0.00	0.13	0.01	0.02	0.09	0.10	0.09	2.33
X-MAN-C1	0.02	0.68	1.01	1.17	1.02	1.01	1.02	0.12	1.01	137.27
X-MAN-FB	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.01	0.01	1.13
CIH	0.00	0.00	0.01	0.02	0.03	0.03	0.03	0.03	0.03	4.05
Maximal Latency (%)										
X-MAN	0.00	0.00	0.00	0.24	0.00	0.00	0.21	0.21	0.20	13.51
X-MAN-C1	0.01	0.02	0.20	0.29	0.33	0.33	0.34	0.34	0.34	142.79
X-MAN-FB	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	11.43
CIH	0.03	0.26	0.26	0.02	0.01	0.01	0.01	0.01	0.01	5.99
Jitter (%)										
X-MAN	0.76	5.98	0.02	0.01	0.02	0.01	5.91	6.92	6.00	6.45
X-MAN-C1	0.00	6.81	129.47	102.30	6.47	40.33	12.34	12.41	22.30	9.68
X-MAN-FB	0.76	0.03	0.00	0.01	0.02	6.60	6.20	6.63	6.22	3.23
CIH	0.76	0.11	0.01	0.01	6.26	0.01	0.03	0.01	6.24	6.45

pendently running CNFs. This scenario with two CNFs require the monitoring of two independent CNFs with different workload levels and power management appropriate to the needs of the higher load (higher intensity) CNF without conflict and without over-scaling the frequency of the CPU core.

Due to the limitations of the testbed used for this work, two CNFs are deployed on two different CPU cores on the same CPU package, whereby the frequencies of them are automatically forced to be synchronized, i.e. all cores on a given CPU run at the same frequency (and the X-MAN power manager controls the operating frequency of the entire CPU package). For the testbed used in this work, the frequency synchronization between CPU cores on a particular CPU ensures that the evaluation is equivalent to the evaluation scenario of running two CNFs on the same CPU core, in terms of frequency scaling.

The average CPU frequencies and the power values of NPM, X-MAN and CIH are compared against the theoretical optimum as illustrated in Figure 4.10. Because of the frequency synchronization issue introduced above, the theoretical optimal power values are the ones corresponds to the optimal power values running in all the cores of a CPU package. In this evaluation, X-MAN extensions are not used to avoid clutter.

It can be observed from Figure 4.10 that the average CPU working frequency of X-MAN is usually closer to the optimal average CPU frequency than that of CIH approach. Particularly, for ON time slots, X-MAN is able to scale to a much lower optimal CPU frequency when the workload traffic allows for a lower CPU frequency. It can be observed that for streams (packet bursts) 2 through 7, the X-MAN CPU frequencies are fairly close to the optimal CPU frequency when the optimal CPU frequency is below 3 GHz. In comparison, for these ON periods, average CPU frequencies of CIH approach reach only slightly below the NPM CPU frequency of 3.3 GHz. According to my analysis, this is mainly due to the difficulty of CIH approach in managing the CPU

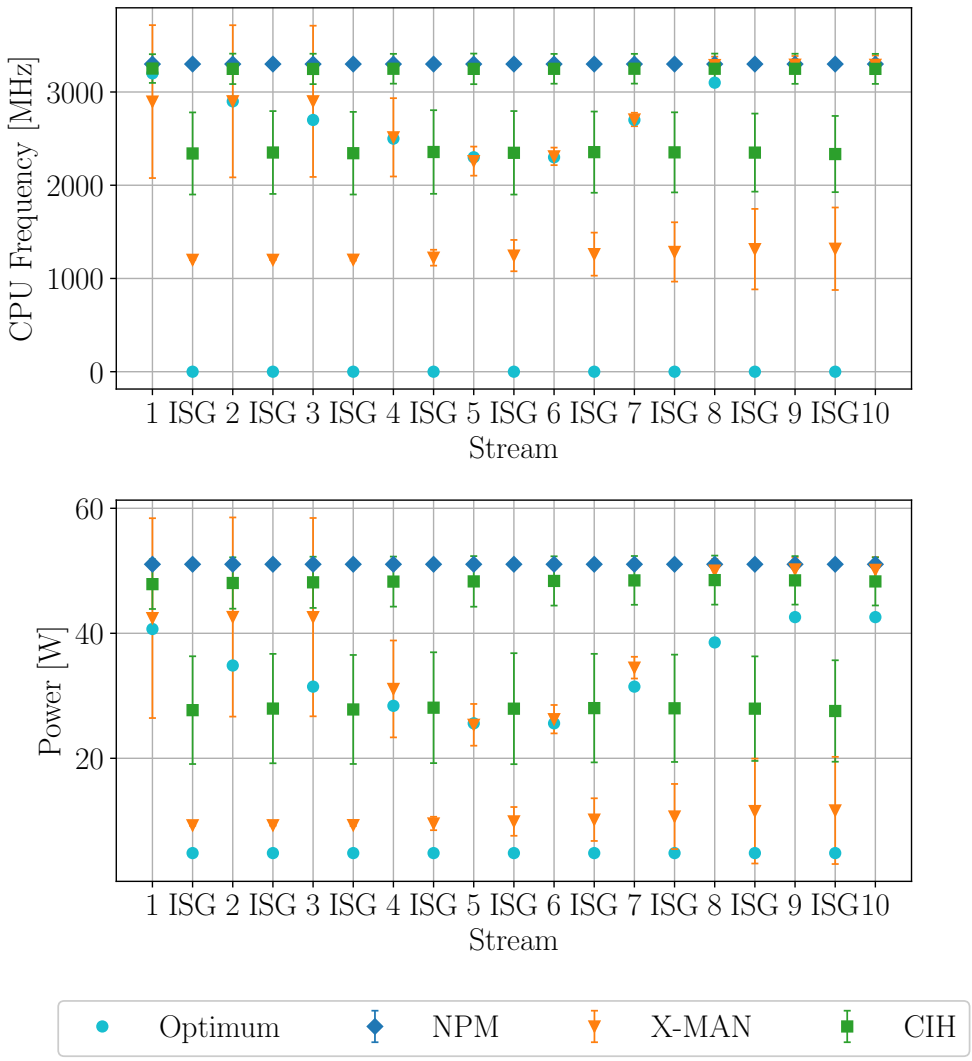


Figure 4.10: Average CPU frequency and power values of two CNFs for deterministic traffic as a function of packet traffic (illustrated in Figure 4.4b) train index ranging from 1 to 10. Reprinted from **my journal paper** [7].

frequencies of CNFs with heterogeneous CPU frequency requirements. CIH is mainly designed for CPU frequency control of each individual CNF separately, namely, CIH approach optimizes the CPU frequencies of two CNFs individually to find an optimal CPU frequency for each CNF. In the case of multiple heterogeneous CNFs running simultaneously on the same CPU, CIH approach thus gives different (i.e. could be very conflicting) optimal CPU frequencies, which are resolved in CIH by scaling the frequencies to a high level. In comparison, due to the separate and independent traffic monitoring of each CNF, the PM of X-MAN can wisely set the CPU frequency for each ON period to the higher required CPU frequency of the two CNFs.

Under heterogeneous workload traffic loads, a conflict arises where the end of a low link utilization traffic burst prompts the CNF of the stepping-down traffic pattern

to request a lower CPU frequency, while the other CNF handling the persistent high link utilization traffic bursts request a much higher CPU frequency. The CIH design then reduces the frequency only a bit, resulting in only a slightly lower average CIH CPU frequency with an average value of 3.25 GHz. In comparison, the native X-MAN performs a global CPU frequency scaling and keeps the CPU frequency around the NPM value to guarantee low-latency processing of persistent high link utilization traffic bursts.

For the OFF periods, it can be observed from Figure 4.10 that X-MAN is able to reduce the average CPU frequency to close to 1.2 GHz, while CIH only keeps the average CPU frequency above 2 GHz. This is mainly due to the limited responsiveness of CIH approach, and the default parameters provided by the official DPDK sample are used. At the same time, when the PM detects the ISGs based on the data provided by the responsive XDP-based TM, the X-MAN is able to immediately scales the CPU core frequency to minimum available frequency.

Echoing the CPU frequency results, it can be observed from the power results in Figure 4.10 that X-MAN takes advantage of the energy savings due to the lower required CPU frequency to handle the relatively low link utilization traffic bursts (streams ranging from 3 to 7), as indicated by the significant energy reduction achieved by X-MAN for streams 4-7. In addition, X-MAN is able to utilize existing ISGs to reduce power values to about 10 W, while CIH approach keeps power values during OFF periods at about 27.9 W.

Table 4.4: Two CNFs latency results for the deterministic traffic. Latency increases are listed as percentage with respect to the performance of the baseline NPM approach. Reprinted from my journal paper [7].

Link Utilization (%)		5	10	15	20	25	30	35	40	45	50
Average Latency (%)											
Step-up	X-MAN	0.00	0.00	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.40
	CIH	0.01	0.00	0.00	0.00	0.00	0.00	0.01	0.11	0.30	15.28
Step-down	X-MAN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.02
	CIH	0.00	0.00	0.00	0.00	0.00	0.01	0.03	0.04	0.19	0.21
Maximal Latency (%)											
Step-up	X-MAN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.36
	CIH	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.02	0.08	35.60
Step-down	X-MAN	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	CIH	0.32	0.00	0.00	0.00	0.00	0.00	0.02	0.04	0.06	0.07
Jitter (%)											
Step-up	X-MAN	1.46	1.06	0.57	3.12	0.00	0.01	0.01	0.01	0.00	0.45
	CIH	4.39	0.35	0.82	6.02	0.05	0.05	0.11	0.01	6.35	5.55
Step-down	X-MAN	5.17	0.21	0.29	0.00	0.15	0.00	0.01	0.00	0.14	0.03
	CIH	6.45	0.65	6.32	6.54	0.02	0.03	6.61	0.01	0.08	6.20

It can be observed from Table 4.4 that X-MAN presents very similar latency performance as the baseline NPM approach. In contrast, it can be observed that CIH can cause significant increases of both average and maximal packet latencies for the packet traffic burst with the highest (namely, 50%) link utilization of the traffic pattern for step-up. This increase in latency is mainly due to the fact that the corresponding

low link utilization traffic burst in degraded traffic patterns ends exactly one second before the end of the high link utilization traffic burst. The slight down scaling of the CPU frequency directly after the ending of the low link utilization traffic burst can lead to substantial increase of packet latency. All in all, it can be concluded from the results in Figure 4.10 and Table 4.4 that X-MAN can perform globally optimized power management for numerous independently deployed CNFs with negligible latency overhead for each packet.

4.5.5 Single CNF with Random Traffic

For the Poisson random traffic profile, while Figure 4.11 illustrates the box plots of CPU frequency and power values, Figure 4.12 presents the box plots of percentage deviation of packet RTT latency and packet loss results with respect to NPM. It can be observed from Figure 4.11 that NPM and CIH are always at the highest CPU frequency. At the same time, X-MAN can repeatedly reduce the CPU frequency in ISGs. Furthermore, X-MAN is also able to boost the frequency to the highest frequency directly in case a new packet arrives. Therefore, X-MAN is able to significantly reduce the energy consumption to only half of the NPM and CIH energy consumption for about three quarters (namely, $3/4$) of the workload traffic streams.

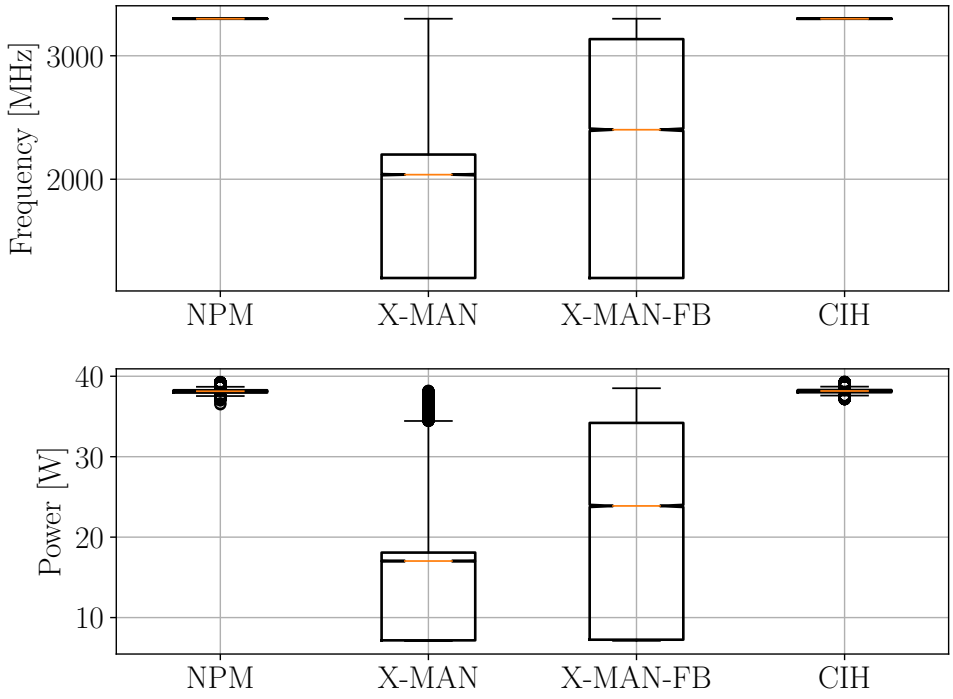


Figure 4.11: Box plots of CPU frequency and power values for a single CNF with random traffic. Reprinted from **my journal paper** [7].

It can be also observed from Figure 4.11 that X-MAN-FB tends to have a higher CPU frequency and energy consumption than native X-MAN. This is mainly because

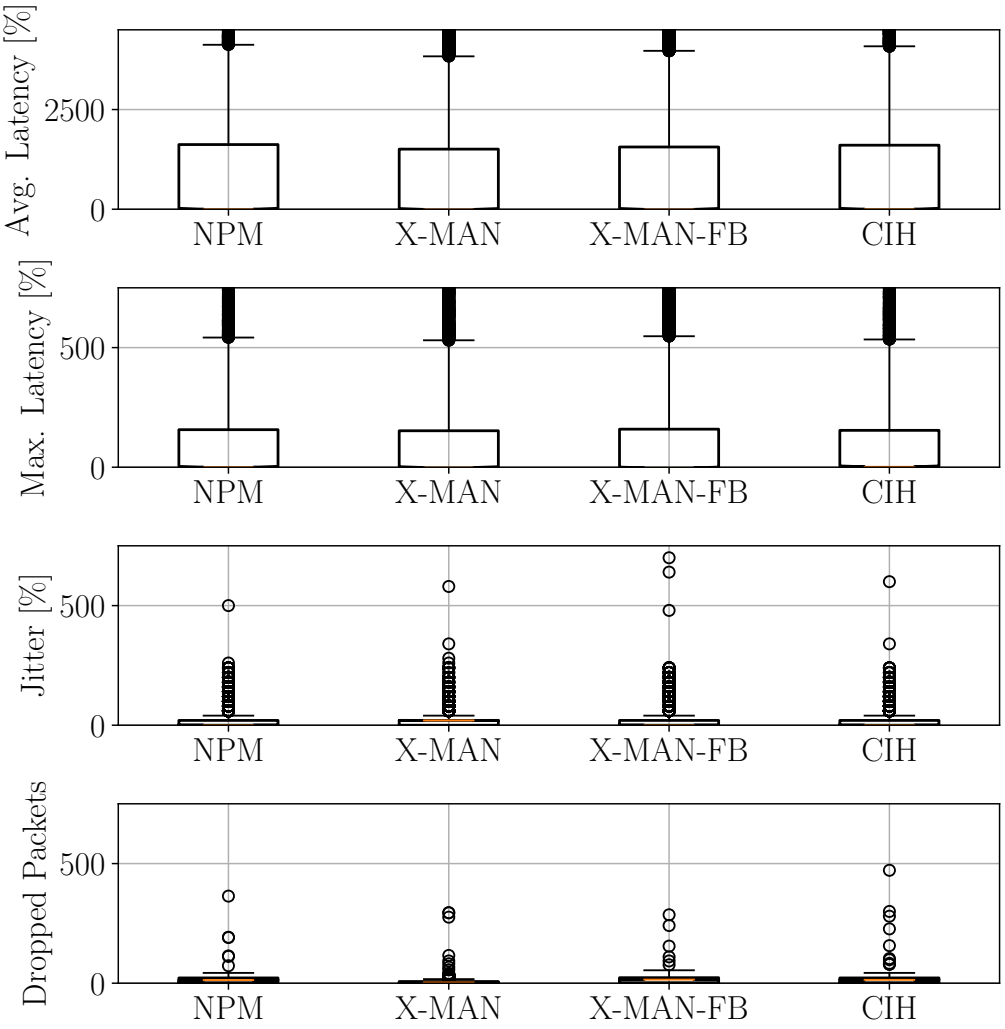


Figure 4.12: Box plots of percentage deviation of RTT latency characteristics and number of dropped packets with respect to the baseline NPM. Reprinted from **my journal paper** [7].

X-MAN-FB strives to iteratively reach the optimal CPU frequency, and in the process of approaching the optimal frequency, if any packet losses occur, for example due to interference from the OS background processes, there will be an immediate rise in frequency. While the native X-MAN scales frequency only with respect to the workload traffic, X-MAN-FB also takes the number of packet losses into consideration. The amount of packet loss depends not only on the PPS of the input traffic, but also on the processing delay of the CNF. In theory, the processing latency of the deployed CNF can be a fixed number. However, deployments of CNFs on actual physical machines exhibit some variation in actual CNF processing latency, which may result in some non-negligible packet losses within the management interval. X-MAN-FB can monitor the number of dropped packets and tries to scale up when this number of dropped packets increases. Therefore, X-MAN-FB tends to use a higher CPU frequency and power value.

As shown in Figure 4.12, the overall RTT latency performance of X-MAN is not degraded. Particularly, all latency related metrics are in the same range of NPM and CIH.

4.5.6 Energy Consumption of X-MAN

In this Subsection, the energy consumption required by X-MAN TM (runs in Linux kernel space) and PM (runs in user space) modules are evaluated for probing traffic with a constant bit rate. As introduced above, the DuT server used in this work has two CPU packages, and each package has four CPU cores (0-3) where the working frequencies are synchronized automatically. While the in-kernel TM modules run on the CPU package 0, the CNF run on core 0 of the CPU package 1. Then the PM process of X-MAN is deployed on the core 1 of the CPU package 1.

The additional energy overhead required by the TM module is indirectly measured in this Subsection by comparing the energy consumption of the CPU package 0 without and with TM modules are enabled and running. This indirect measurement is performed because, to the best of my knowledge, there is no available software tool to accurately measure the power of each individual process running in kernel and user space on the Linux OS. So these tools normally only provide a per-CPU-package granularity, which is able to measure the energy consumption of the TM modules directly. It can be observed from the results in Table 4.5 (see the second column labeled as X-MAN TM) that the additional energy required by the TM modules is relatively negligible (less than 0.21 W). Furthermore, it should be noted that XDP programs can be offloaded to dedicate hardware that has native support for XDP hardware offloading, for example, SmartNICs [95]. Therefore, this energy overhead on CPU can be fully avoided by offloading TM modules to dedicated hardware.

Table 4.5: Power measurements of NPM and X-MAN: Additional energy consumption (Power Δ) with the TM in the Linux kernel space relative to the operation without TM. Energy consumption for the CPU without power management (NPM) and with X-MAN PM enabled, and percentage of C_1 residency time for CPU core running PM for different CPU operational states. Reprinted from my journal paper [7].

	X-MAN TM	NPM		X-MAN PM	
CPU state	Power Δ [W]	Power [W]	C res. [%]	Power [W]	C res. [%]
1.2 GHz	0.03	7.13	100	7.21	97.83
idle @ 1.2 GHz	0.00	7.01	100	7.19	89.0
3.3 GHz	0.21	38.36	100	38.41	99.17
idle @ 3.3 GHz	N/A	37.66	100	N/A	N/A

In order to evaluate the PM module of X-MAN, it needs to be noted that the used CPU of the DuT synchronizes the working frequencies of all CPU cores on a given package, namely all 4 cores on CPU package 1 increase their frequencies when the CNF requires a higher frequency, even if core package 1 has only minimal processing tasks from the PM itself and the other two CPU cores are nearly idle (these two cores

should sleep in the C_1 C-state for the most time. Deeper sleep states are disabled in this work). Furthermore, the used tool `turbostat` cannot directly measure the power values of each individual CPU core, namely it only has per-package granularity. However, `turbostat` can provide C_1 residency time of each individual CPU core of a CPU package. Therefore, the C_1 residency times are measured in this work to indirectly measure the energy consumption of the PM module of the X-MAN. The power values of the CPU package 1 is also listed in the Table 4.5. The difference of energy consumption between the baseline NPM and X-MAN indicates the additional energy consumption required by the PM module of X-MAN.

The power values for two representative frequency values, namely 1.2 GHz for lower bound and 3.3 GHz for upper bound, for a single deployed CNF without any workload and with high workload (100% link utilization) are listed in the Table 4.5.

As showed in the Table 4.5, The additional energy overhead introduced by PM of CALVIN is only about 0.05 W, namely negligible, when the frequency of the CPU is 3.3 GHz. Furthermore, for the same 3.3 GHz frequency, about 99.2% of the time, the CPU works in the C_1 state. Thus, the PM of the X-MAN only introduces a minuscule overhead for the CPU. For low workload, namely a CPU frequency of 1.2 GHz, it can be observed that the X-MAN PM consumes only slightly more energy, i.e. only 0.18 W when no workload traffic arrives (about only 2.5% more than the NPM approach).

4.6 Summary

In this work, a novel and unique approach of power management named as XDP-Monitoring energy-Adaptive Network functions (X-MAN) for Cloud-native Network Functions (CNFs) is designed, implemented and rigorously evaluated on the State of the Art (STOA) practical high speed 10 Gbps testbed. Compared to the STOA Code Instruction with Heuristic power management (CIH) and Hardware Counter (HC) approaches, X-MAN is able to monitor the workload of each individual CNF independently through lightweight and accurate in-kernel Traffic Monitors (TMs) that are dynamically attached to the data plane virtual NICs. Thus, X-MAN is able to provide fully non-intrusive in-band traffic monitoring. Furthermore, a user space Power Manager (PM) with a global view of all deployed CNFs is used by X-MAN to perform globally optimized power management to significantly reduce energy consumption. All source code of X-MAN and corresponding evaluation is publicly available to facilitate the deployment of X-MAN and future research and development of highly responsive and energy efficient power management systems for CNFs.

Rigorous measurements and evaluations are performed in this work on a high-performance hardware testbed to fully evaluate the performance of X-MAN. According to the measurement results, X-MAN is able to provide much more responsive traffic monitoring than the STOA HC approach [8]. Furthermore, significantly much more energy can be saved with the adaptive power management algorithm used by X-MAN. At the same time, the X-MAN is able to achieve this energy saving with relatively negligible impact on the latency and bandwidth performance.

5 ComNetsEmu: An Open Source Testbed for Virtualized Communication Networks

All contents in this Chapter has been published in **my journal paper** [118]: "An open source testbed for virtualized communication networks." IEEE Communications Magazine 59, no. 2 (2021): 77-83.

5.1 Introduction of ComNetsEmu

In previous chapters, the measurements and evaluations are performed on practical physical testbeds consisting of multiple physical COTS servers. Although the physical testbed is mostly close to the real-world cloud platform and is able to provide realistic measurement results, the cost, maintenance overhead and portability of a such complex physical testbed limit the adoption, deployment and reproducibility of the testbed [119].

In order to address this challenge with my best efforts, a novel emulator for communication networks named as Communication Networks Emulator (ComNetsEmu) is designed, implemented in this work to simply the prototyping and evaluation of research ideas for STOA and future softwarized communication networks.

The complexity of the future communication networks can significantly increase because the COmputing In Network (COIN) or In-Network Computing (INC) paradigm needs to be enabled by the network for a broad variety of the latest trending use cases. It needs to be noted here that COIN and INC are different/similar terms for this emerging paradigm. Until the time of this work, to the best of my knowledge, this paradigm is not yet formally named and standardized. Common scenarios range from MEC to large cloud platforms. A typical and timely example is the current reliance of the fifth generation (5G) mobile communication networks and beyond on this paradigm shift from the current network forwarding of data to network processing of information. In fact, several of these scenarios are being addressed in the

upcoming 5G standards, where the architecture of 5G-based services will play an essential role [120]. The virtualization and softwarization of network resources and functionalities have the dominant impact in these scenarios, because the function of the communication network starts to migrate from traditional dumb bit pipes to information-oriented provisioning of flexible services. This revolutionary paradigm migration can include resource-intensive Machine Learning (ML) based applications that need to be dynamically and flexibly deployed and configured across the network. Two trending emerging paradigms, namely SDN and NFV, can enable and accelerate this paradigm migration. In short, while the SDN gives programmability of the networking, NFV enables dynamic and flexible deployment and orchestration of NFs in the softwarized network.

However, for normal network researchers, educators and especially students, learning, prototyping and teaching this novel field of study can be difficult due to the requirement for an appropriate softwarization virtualization environment. In addition, a number of different approaches to implementing custom virtualization configurations exist for some network testbeds based on pure simulation, such as the `GENI`, which is described in [121]. These are often too difficult to set up and combine in the general target of hands-on, active learning in the general education environment that is focused by this work. These solutions typically add unnecessary barriers and complexity to achieve a realistic environment in which it is convenient to study how such a network will operate in practice, how to deploy real services on its infrastructure, and how to perform performance evaluation of the novel approach.

In response to these challenges, this work describes the design, implementation, and operation of a new software framework or testbed called Communication Networks Emulator (ComNetsEmu). ComNetsEmu enables any student, researcher, or networking expert to build a fully virtualized network on a single COTS device (e.g. a normal laptop or COTS desktop servers). Due to the complexity of a complete (standard-aligned) NFV system implementation and the focus of ComNetsEmu as a research-oriented prototyping and teaching tool, at the time of this work, ComNetsEmu focuses solely on providing NFV Infrastructure (NFVI). This allows the study and application of basic and advanced principles of SDN and NFV to be performed at any time, on any COTS device, while allowing the implementation of other essential NFV components in the future. The main contribution of this work is therefore to enable research prototyping as well as teaching and training through problem-based learning of modern networks driven by SDN and NFV.

This softwarization testbed framework is developed to run inside a single standalone VM that combines the widely used SDN network emulator Mininet [122] and the de facto standard application container framework Docker [123] for NFVI into an integrated framework. The freely accessible collection that makes up ComNetsEmu also provides numerous representative off-the-shelf examples of SDN and NFVI basics, as well as more advanced practical applications and APIs to further extend them as needed. All source code and comprehensive documentation are publicly available in the Gitlab repository [119] of the The Deutsche Telekom Chair of Communication Networks (ComNets) at Technische Universität Dresden (TUD). Furthermore, an in-depth introduction of the ComNetsEmu and review of representative examples and their corresponding applications can be found in the book written by Fitzek *et.al.* [1].

5.2 The Architecture of ComNetsEmu

As introduced in Section 5.1, the SDN functionalities of ComNetsEmu are provided by the widely used famous Mininet framework. Therefore, a brief introduction of Mininet is performed before describing the novel enhancements provided by ComNetsEmu.

5.2.1 SDN Environment with Mininet

Mininet is a lightweight and most widely used STOA network emulation environment that enables rapid prototyping and evaluation of a complete and practical network system based on the functionality of the Linux OS [122]. In special, Mininet utilizes the built-in virtualization capabilities in the modern Linux kernel to create a virtualized network of network applications, hosts, switches, routers and other common types of network nodes on a single underlying physical machine. Because Mininet supports OpenFlow [124] protocol and other SDN components, it is able to support rapid prototyping for SDN development in a very straightforward, rapid and reproducible manner. While like many Mininet-based projects, Ryu SDN controller [125] is chosen by ComNetsEmu to demonstrate most SDN related examples, any OpenFlow-enabled SDN controller can be adopted, including ONOS [126] or OpenDayLight [127]. However, industrial-grade and production-oriented SDN controllers are not the focus of this research-oriented platform, namely ComNetsEmu.

Since Mininet makes use of the application container related mechanisms of the Linux kernel, Mininet hosts all share the same Linux OS kernel as well as process IDs, usernames and file systems, since they execute as regular processes with configured namespace and control groups. Each host in Mininet also has a separate network stack (by using the network namespace), including common resources such as Address Resolution Protocol (ARP) caches or routing tables. In addition, each host can be allocated with multiple virtual network interfaces. By default, the veth devices provided by Linux kernel are used. veth device can be connected to a virtual software switch. Similarly, the connected virtual links can be individually configured with different representative characteristic parameters, such as (propagation) latency, bandwidth and loss rate. Therefore, the Mininet emulator provides an ideal lightweight and high-fidelity environment for reproducible SDN-powered network research.

5.2.2 ComNetsEmu Enhancements and Architecture

ComNetsEmu extends the vanilla Mininet to support better support emulation of versatile COIN related applications. It extends and puts forward the concepts and work in the Mininet-fork Containernet project [128]. It uses a slightly different approach to extend the Mininet compared to Containernet [128]. One main focus of ComNetsEmu is to use “sibling containers” to emulate softwarized network systems with computing in the loop. Compared to vanilla Mininet and Containernet, ComNetsEmu is able to manage Docker containers inside Docker hosts. Docker-IN-Docker (DIND) (or, more precisely, the “sibling container”) mechanism is used as a

lightweight emulation of nested virtualization scenarios. A Docker host with multiple internal Docker containers deployed is used to mimic an actual physical host running Docker containers (application containers). While the default Mininet module for hosts is extended to provide functionality for managing DockerHost instances, ComNetsEmu provides an additional `APPContainerManager` component to coordinate internal application Docker containers ("sibling containers"). Emulating physical hosts with Docker containers also enables the Mininet manager to execute long-running processes in these internal containers.

One main motivation for extending the Containernet project [128] to a nested virtualization strategy comes from the important requirement to control the emulated common hardware on which the VNFs or CNFs are expected to run in the emulation. As introduced above, Docker hosts deployed with multiple internal Docker containers are used to emulate the actual physical hosts running the application Docker containers. In an environment purely using Containernet, CNFs are deployed as Docker containers, replacing vanilla Mininet hosts. However, this approach is limiting if the user wants to emulate and investigate the effect of multiple CNFs running simultaneously in a single physical host. Moreover, this approach is very inflexible because it does not allow to easily limit and dynamically adjust the available compute resources allocated to each CNF by the physical hosts running multiple CNFs. In my approach with ComNetsEmu, these limitations are addressed by emulating physical hosts as Docker hosts (just like in Containernet). This allows to firstly emulate heterogeneous physical hosts by limiting the number of CPUs and the available CPU time per host. CNFs are then deployed on top of these simulated physical hosts. This represents a scenario where multiple CNFs have to share the limited computational resources of a single physical host (with the described CPU limits). Therefore, this allows for the evaluation of more complex and practical scenarios (and algorithms) where CNFs must be migrated to other hosts in the network because they cannot meet latency requirements (not due to propagation latency), but due to the length of time required to access host CPU cycles. This enables users of ComNetsEmu to deploy and test algorithms that optimize the placement of CNFs to minimize total latency while experimenting with propagation latency and computation latency. Thus, the ComNetsEmu framework can be used to test algorithms and technologies to find the optimal placement of CNFs with not only the practical networking constrains of bandwidth, latency, loss rates but also the computational constraints of available CPU time and number of CPUs. It should also be noted here that another important benefit of this mechanism ("sibling containers") is the significantly reduced overhead compared to approaches that employ full VMs for host or network function emulation, while maintaining ease of use and flexibility.

ComNetsEmu provides a collection of numerous built-in examples for its main features and the use of `Python` APIs. For instance, `dockerindocker.py` and `dockerhostmanageappcontainer.py` demonstrate how to deploy and manage a Docker container within each virtual Docker host. The `dpdk` directory in the example shows the basic setup for running a DPDK-accelerated layer 2 forwarding application on ComNetsEmu, without specific hardware support. The overall ComNetsEmu architecture and its main components are illustrated in Figure 5.1. Virtual hosts are connected to a configurable data plane managed by ComNetsEmu, which provides an

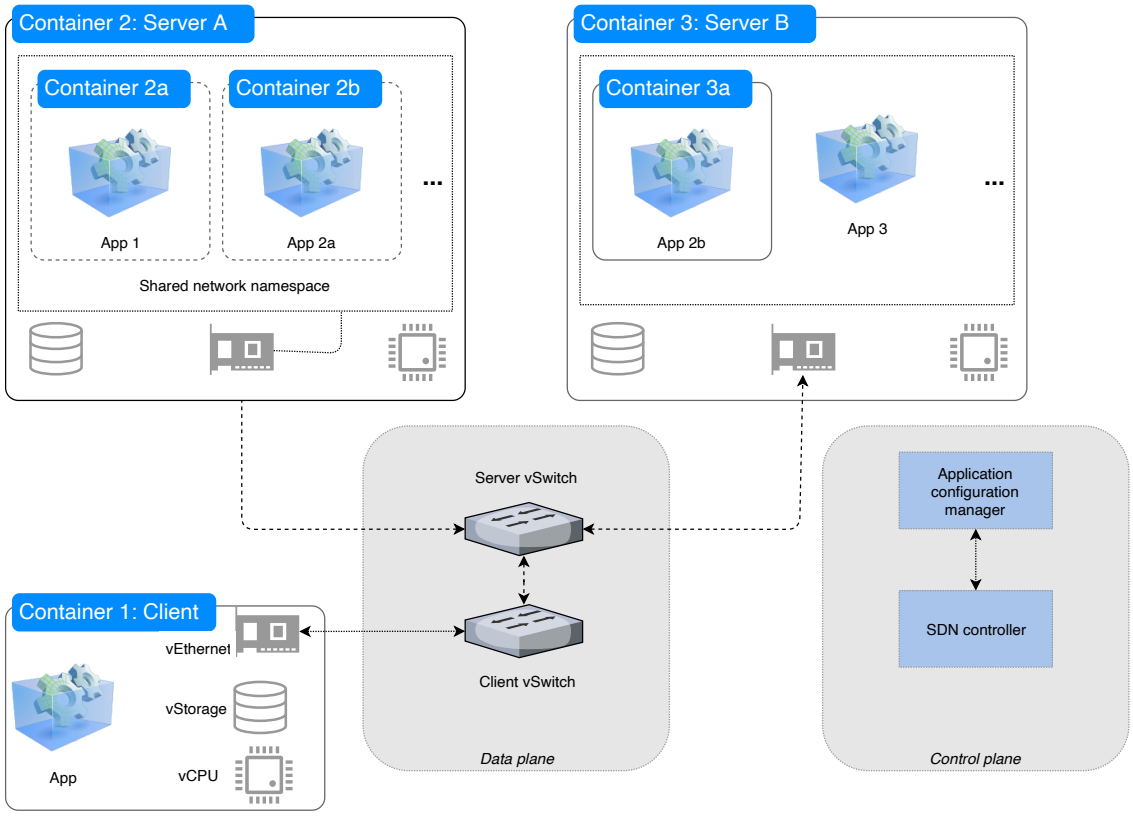


Figure 5.1: The architecture view of the ComNetsEmu. Reprinted from my journal paper [118].

interactive shell to execute commands. Each networked host is implemented as a Docker container within Mininet networking, allowing for finer-grained and flexible resource isolation.

5.3 ComNetsEmu Hands-on Examples

The practical usage of ComNetsEmu is highlighted in this Section with two representative hands-on examples. Information and information about additional examples and practical applications can be found in the book [1].

5.3.1 ComNetsEmu Echo Server Example

The purpose of this first basic and elementary example is to show the general interaction with APIs provided by ComNetsEmu by creating a generic example that demonstrates the basic usage of the emulator. Users can consider this example as a general template or canvas on which to further design and emulate their own desired system. In this basic example, a system consisting of two interconnected emulated computing hosts will be created. These hosts represent the computing infrastructure of user devices and service providers where different network applications

can be deployed. All files for this example are located in the `examples/echo_server` folder.

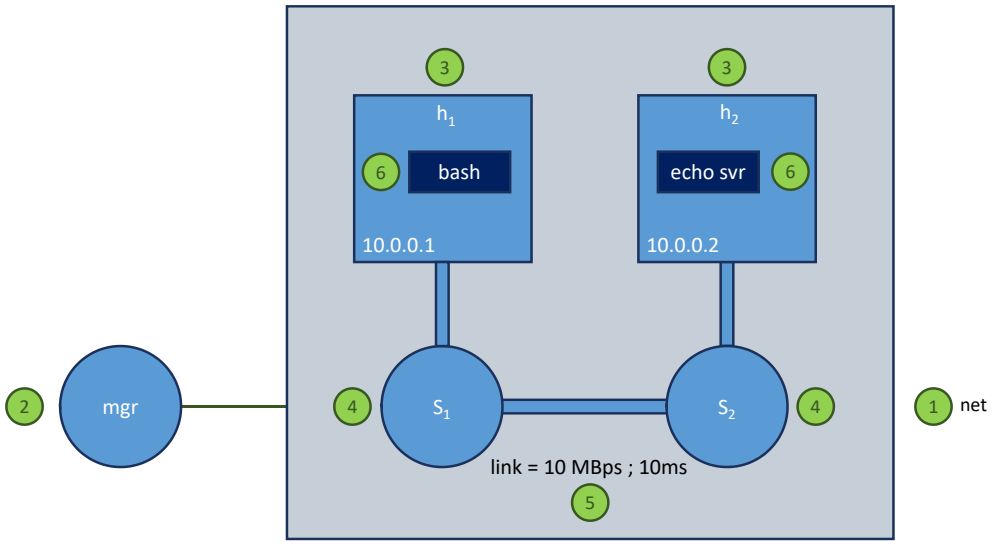


Figure 5.2: The topology of the echo server as a NF. Reprinted from **my journal paper** [118].

Topology for Emulation In this example, a TCP echo server is deployed on one host, while a TCP client is deployed on the other host sending data to the echo server. The to be emulated topology is illustrated in Figure 5.2. Each host is connected to a software switch, and the switches are then interconnected to each other via an analog communication link. Each link is configured with customized bandwidth, latency and random loss rate. This topology is created using the Python script `topology.py`. Corresponding `Containernet` and `Manager` objects are firstly created, called (1) and (2) in Figure 5.2. Subsequently, h_1 and h_2 are created and assigned IP addresses. These hosts were initialized with a base Docker image called `dev_test` provided by ComNetsEmu. After the hosts were created, switches S_1 and S_2 are created, as well as links for bandwidth and latency limits. Once the topology was set up, the echo server is created by running the Docker container. For the client, it is executed with the BASH shell provided by h_1.

The Client and Server The TCP server can be implemented using any programming language or any software the user prefers. As long as it can be integrated into a Docker container, ComNetsEmu can deploy and run it as a NF. In this basic example, the TCP server is implemented in Python, the file is called `server.py`. To containerize it, a `Dockerfile` is provided that simply uses a Python base image and copies the Python script to the container's image. The deployed server simply waits for TCP connections, accepts them, listens for a TCP segment, and echos it back to the sender's IP address. The `netcat` tool is used as the TCP client.

5.3.2 ComNetsEmu Mobile Edge Cloud Example

This Subsection targets to demonstrate the holistic approach provided by ComNetsEmu to emulate a practical 5G network by using its SDN and NFVI capabilities to emulate a MEC system. Source code of this example can be found in the directory of the abovementioned Gitlab repository: `app/realizing_mobile_edge_clouds`. The white paper [129] from European Telecommunications Institute (ETSI) defines MEC as a system that provides IT service environments and cloud computing capabilities at the edge of the mobile network, within the RAN, in closely proximity to mobile users to meet latency constraints for time-critical applications. Implementing MEC for trending latency-sensitive applications such as self-driving cars also requires migrating cloud services from one edge computing node to another in time, all seamlessly and transparently to the end application user. In this example, an MEC system is prototyped that hosts the previously discussed echo servers in an emulated geographically distributed network.

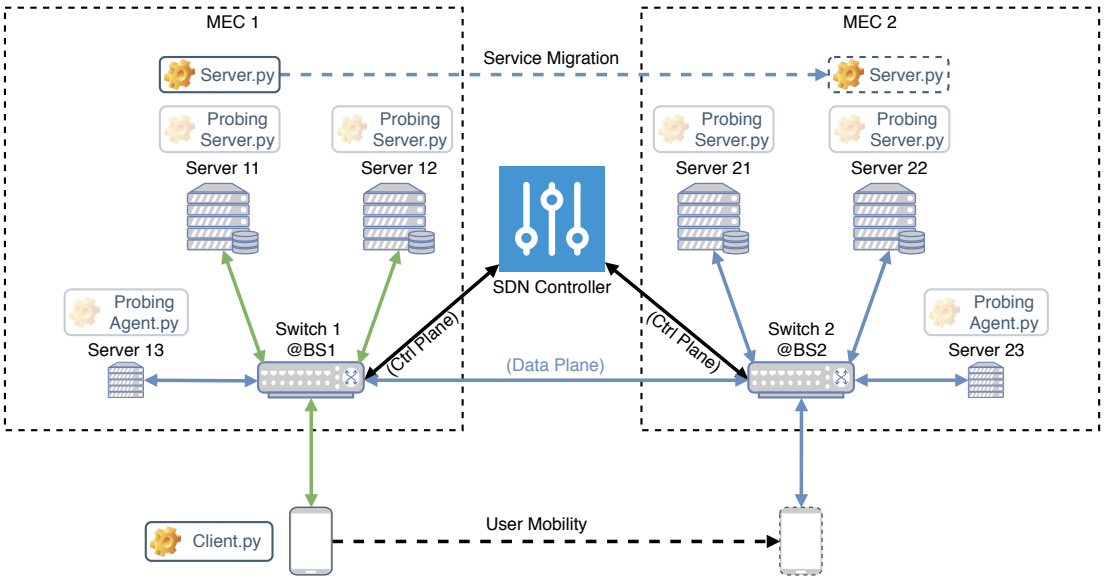


Figure 5.3: The topology of the MEC migration example. Reprinted from **my journal paper** [118].

Topology for Emulation The topology designed for this example is illustrated graphically in Figure 5.3. It is assumed that there are two MECs, each with one Base Station (BS) (BS1 and BS2, respectively), that are geographically separated. Each BS is represented by its respective SDN switch, namely Switch 1 and Switch 2. Both switches are connected to the same Ryu SDN controller. Each MEC also includes a cluster of heterogeneous physical servers connected to their representative switches. Clients, which can be assumed to be standard User Equipments (UEs), can be connected to up to one BS at a time. Therefore, UE mobility can be easily emulated by disconnecting one switch and connecting to another. In this example, the basic echo service is

hosted on one of the servers as a targeted service provided by MEC system. When a UE, namely a client, moves from one BS to another, the echo service is migrated to a different emulated physical server, and the migration is completely transparent to the client.

Selection of the Server to Host the MEC Service Choosing the optimal server to host the MEC service is critical to ensure low latency services. The server selection decision depends not only on network traffic and congestion-related latency, but also on latency due to performance capacities per host, which can vary dynamically depending on the current workload of servers running other parallel services. A straightforward and intuitive way to measure the end-to-end latency from the BS to each host is to simply obtain the latency from the probe request microservice running closest to the switch to the probe response microservice running on each potential MEC host. A dedicated probing service is used in this work to measure latency instead of using standard Internet Control Message Protocol (ICMP) based pings, because the probing service process running in user space can also take into account any processing latency incurred on the host due to dynamic server load. The probing server behaves similarly to a standard echo response server in that the probe agent periodically sends a probe packet to the probe server with a timestamp in the payload. This timestamp is used by the probe agent to estimate the latency of each host from the BS to the MEC system.

Implementation on ComNetsEmu Similar to the Echo Server example described above, a `Containernet` object with a corresponding `manager` and a Ryu SDN controller are firstly created. After that, two virtual switches are created, representing the endpoints of two geographically distributed BSs. Two Docker hosts are spawned to emulate MEC servers. By using `manager` object, a Docker container implementing the probe server is added to each of the 4 hosts. In addition, another host implementing a probing agent is generated and connected to each of the 2 switches. The probing agents periodically report their respective latency to each MEC host to the SDN controller. Based on probing results, the controller determines the ideal host for NF placement and spawns or migrates the application service to the corresponding host. For the sake of simplicity, the challenges involving stateful service migration are not considered in this example. Therefore, the migration process is as simple as spawning the application server container in the new MEC host, updating the traffic in the switch to reroute packets to the new host, and remove the application server from the old host (in that order). The client implementation is similar to the example described earlier, where the migration of clients is achieved by simply disconnecting from one switch to another.

5.4 Summary

This Chapter describes a lightweight and practical software network emulator, namely the Communication Networks Emulator (ComNetsEmu), capable of running on any

laptop or COTS server, which is tailored to teach the basic and advanced concepts related to the introduction of SDN and NFV in modern networks, namely network softwarization. While currently focused on NFVI, future implementations of other important NFV components are able to greatly expand the scope and use cases of ComNetsEmu. The ComNetsEmu framework, which is freely available to the community [119], significantly simplifies the learning of such concepts by providing comprehensive built-in examples and offers the possibility to prototype new networking systems and experiments with softwarization solutions.

6 You Only Look Once, but Compute Twice (YOLO-CT): COIN for Low-latency Object Detection in Softwarized Networks

Part of contents in this Chapter has been published in my journal paper [10]: "You only look once, but compute twice: Service function chaining for low-latency object detection in softwarized networks." Applied Sciences 11, no. 5 (2021): 2177. However, some transport protocol issues are not addressed in [10]. This open topic is further researched and described in this Chapter. The proposed Real-time Transport Protocol (RTP) based transport solution is designed, implemented and also evaluated (with ComNetsEmu) in this work.

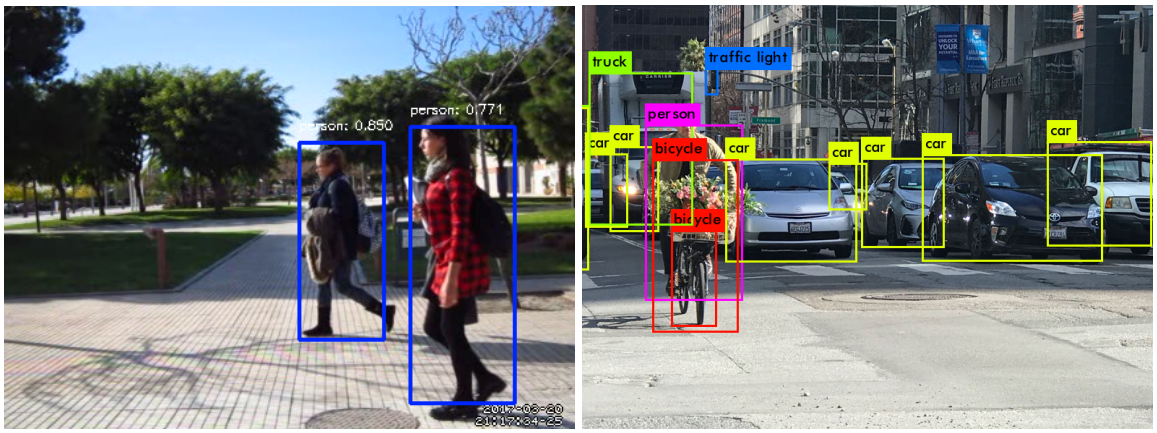
6.1 Introduction

6.1.1 Overview and Motivation

As summarized in Section 3.6, the CALVIN approach is able to achieve ultra-reliable low-latency RTT performance at the cost of significantly reduced bandwidth performance. For STOA softwarization data plane technologies, to the best of my knowledge, there is a fundamental trade-off between maximal available bandwidth and the per-packet latency performance. However, a lot of real-world applications have strict requirement for both relative high bandwidth and ultra-reliable low-latency performance. For example, an emerging use case for 5G and beyond is low-latency and reliable real-time video streaming analysis.

According to the prediction performed by Cisco [130], by 2022, 82% of the IP traffic will consist of video traffic. Within the domain of video data traffic analysis, specifically, the object detection subcategory presents an additional significant latency requirement, especially when applied in some certain scenarios [131]. The object de-

tection and analysis in live video streams is mainly based on real-time video analysis in the field of Computer Vision (CV). Typical examples for real-time object detect and analysis contain Google Lens, smart city applications based on video surveillance [132] or autonomous driving cars. Two sample results of object detection applications are illustrated in Figure 6.1.



(a) Pedestrians (image from [133]). (b) Vehicles (image from [134]).

Figure 6.1: Object detection use cases including pedestrian and vehicles detection.

Due to the fundamental computational complexity involved, there are significant challenges to responsively and reliably perform real-time video analytics on relatively resource-constraint devices, such as mobile phones (namely normal UEs) or ad-hoc IP cameras for video surveillance. These requirements become even more challenging when taking STOA relative high frame rates of live video streaming into consideration. The computational and energy overheads are often very high when data is processed locally, as the captured image analysis and CV tasks constitute the visual understanding, often incorporating the ML-based algorithms involved. In recent years, ML based approaches have undergone steady improvements with significantly increasing responsiveness, precision and recall, especially for the approaches based on the promising Deep Learning (DL) technologies [135].

Because these DL-based approaches can significantly outperform conventional approaches, DL-based approaches are becoming increasingly popular and widely deployed. These approaches are often based on the CNN model. Although the training process of these CNN models are highly resource intensive, pre-trained models can be used to perform the inference on real-time video streams with a reasonable accuracy. Therefore, the focus of this work is only on the inference step. Representative STOA approaches of this type include Regions with CNN (R-CNN) [136], Faster R-CNN [137] and especially YOLO [138], which novelly combines the high precision with significantly improved inference speed.

In the environment with high mobility, the focus on latency optimization must combine several requirements, such as resource usage and low latency object detection. Common computational resources considered include memory, CPU and other acceleration hardware including e.g. Graphics processing unit (GPU). However, overall system cost often needs to be factored into the full solution. For example, future

smart transportation systems and object detection applications connected to self-driving cars are highly latency sensitive and mission critical at the same time. Current conventional approaches are usually limited in realizing the full potential offered by the upcoming and trending softwarization-enhanced networking system. Several questions and challenges exist for the conventional *store and forward* network architecture:

- Object detection based applications introduced above are very resource intensive. Therefore, they are not suitable for locally and prolonged execution on resource-constraint and battery-limited devices.

- In comparison, high computational power and flexible resource management provided by the STOA trending cloud platforms can be used to accelerate these computationally intensive tasks through networking enabled computational offloading [139].

- One interesting promising approach to overcome challenges of fully local processing while ensure low latency performance is to combine local in-network processing (or COIN) and remote cloud computing service. Although conventional networking infrastructure and related protocols can not provide native support for COIN functionalities, new softwarization based networking system provides support to deploy and orchestrate COIN applications.

- The cloud-based offloading requires all video data to be transmitted through the network. The task of real-time streaming of high resolution video, which has normally very high bandwidth requirement, is challenging for the QoS of the underlying networking system, especially the latency performance.

- Bufferbloat [140] or buffer overflow of network nodes can be one of the main cause of delayed or even losses of packets in a network even within a limited geographical area, e.g. a campus network. Bufferbloat describes the behavior of network nodes when their buffers to store and queue packets almost reach the maximal capacity and have to queue packets for a significantly long time or even drop the incoming packets until buffers are freed by processing and transmission processes. According to the initial measurements perform by Rischke *et.al.* [141] on a practical 5G campus network testbed, there are nearly zero network link losses in the campus network, especially the core network, and packets can be dropped mainly due to the bufferbloat issue. Compared to full hardware solutions, the software based packet processing proposed by NFV paradigm makes this problem more challenging due to the limited raw performance provided by software.

- Convolutional STOA transport layer protocols including TCP and UDP are designed and implemented based on the end-to-end principle without considering the COIN paradigm. Therefore, these conventional protocols in transport layer and beyond are not able to be directly used by COIN applications. Several challenges, open questions and related works are listed by one draft of IETF

COIN Research Group (COINRG). Thus, transport protocol issues for networking system with COIN enabled is an interesting research area when conducting this work.

One novel and promising direction to address above described challenges is to reduce the total amount of data required to be transmitted by the network from end devices to MEC cloud platform. One can argue that moving everything to hardware, e.g. with P4 [142] programmable switches and other special hardware network forwarding chips can solve the performance problem. But it's argued in this work that even if it's faster, there's still a limited queue size that can not be ignored. The amount of data generated by end devices would keep increasing with the increased number of connected devices and their requirement of high-quality multimedia services. Moving everything to hardware may not be the silver bullet to solve all latency problems, especially the price must be paid to lose the easy programmability and flexibility of software solutions. Therefore, the programmability and support for intelligent data processing of the software based solutions are researched in this Chapter to ensure and even improve the latency performance of the network.

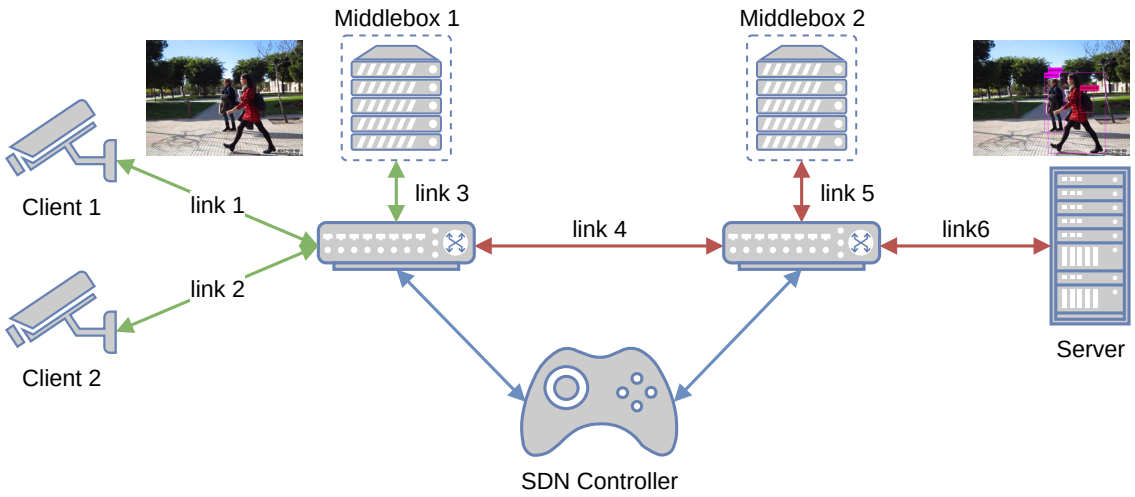


Figure 6.2: A basic dumbbell topology for remote cloud based objection detection application.

A minimal dumbbell topology of a typical minimal MEC cloud system enhanced object detection service is graphically illustrated in Figure 6.2. Because not always both clients send traffic at the highest available bandwidth, the network node and links (namely, the links ranging from 3 to 6 in Figure 6.2) are shared between clients in order to save the deployment and maintenance cost for Internet Service Provider (ISP). In real-world deployment, the maximum available bandwidth for link 4 can be as little as a quarter of the maximum bandwidth for link 1, or even less. However, the client 1 can request the object detection service provided by the server running in the cloud while client 2 may also send high speed traffic to the cloud server for other services. Then the shared link 4 becomes the bottleneck and the middleboxes 1 and 2, on which the CNFs are deployed, are overwhelmed and start queueing or

even dropping received packets. This is so called the dumbbell problem. The common STOA solution for this issue is performed only on the end hosts, namely the client is able to detect this bottleneck and use a pre-configured congestion control algorithm to reduce the sending rate. However, these congestion based approaches usually require multiple RTT to reach the stability and have negative impact on bandwidth and latency performance. Furthermore, approaches only rely on end hosts can waste the computing power and programmability provided by the softwarized modern network devices. If the middlebox 1 in the Figure 6.2 is able to perform some smart computation on the incoming video stream to significantly remove redundant or unnecessary data and thus reduce the raw amount of data needed to be transmitted through the networks, the bandwidth pressure on other network nodes and links can be significantly reduced. Therefore, in this Chapter, a novel approach named as You Only Look Once, but Compute Twice (YOLO-CT) is designed, implemented and evaluated with the ComNetsEmu testbed introduced in previous Chapter 5 which utilizes the computational power provided by emerging COIN paradigm to deploy part of the CNN model into the network for image pre-processing and data reduction.

6.1.2 Related Work

With the development emerging network softwarization technologies such as SDN and NFV, the programmability and flexibility of the communication network are significantly improved. This trend leads to a new and novel paradigm for the next generation communication network, namely COmputing In Network (COIN). COIN is accompanied by the prospect of deploying data processing functions on network devices such as switches, routers, middleboxes and NICs [143]. The Internet was designed as a best-effort packet network which offers very limited guarantees for timely and successful deliver of packets and corresponding data. Complex data manipulation, content-aware data computation, and advanced protocol features of transport layer and beyond are generally only deployed by the end hosts which makes the network nodes and links as a “dumb pipe” focusing only on a simple data transmission in a *store and forward* manner [143]. This design has been shown to be suitable for a wide variety of applications and has contributed to the rapid and wide adoption and growth of the Internet.

However, as already introduced in the previous Subsection, emerging fields and use cases require more than best-effort forwarding for higher and stabler performance, especially the latency performance. The vision of COIN or INC and corresponding paradigm of joint optimization of both computation and networking resource of the underlying communication network draws a lot of research interests from both academia and industry.

Sapio *et.al.* explores the programmability provided by the modern network data plane to offload data aggregation tasks into the network. This work sheds the light on the direction to utilize COIN capabilities to reduce the bandwidth pressure on the transmission network. However, the initial prototype in this work uses UDP and has the strict limitation that the size of the used Application Data Unit (ADU) must be smaller than the MTU of the underlying network. The ADU is the Protocol Data

Unit (PDU) for the application layer. For instance with video streaming over network, the ADU can be a single decodable video frame. Normally, the size of a video ADU is much larger than the MTU provided by the underlying network. Therefore, each ADU has to be fragmented and reassembled for network transmission. Conventionally, these operations are only performed on the end hosts. However, when COIN paradigm is considered, these two essential operations related to ADU processing must be considered for all COIN-enabled network nodes. The idea to offload the execution of part Artificial Neural Network (ANN) layers to in-network devices to reduce the workload of CPUs is explored in [144]. However, this work focuses mainly on how to efficiently split the ANN. The proposed distributed system is not evaluated neither on a network emulator nor on a distributed physical testbed. The transport issues introduced above are also not covered in this work.

In [145], an edge detection filter is prototyped for programmable network devices. It explores the challenges involved in offloading CV applications to the in-network devices. However, the transport issues of network communication are also not researched in this work.

Wu *et.al.* [146] designed, implemented and evaluated the novel Network Joint Independent Component Analysis (NJICA) approach based on COIN paradigm. Compared to the conventional centralized Independent Component Analysis (ICA) approaches, NJICA can significantly reduce the computation latency on remote servers. However, this work proposed a clean-slate (namely, not mature and robust) message transport protocol, which has the critical challenge to be deployed in real-world networking and coexist flawlessly with other standardized network protocols.

COINRG Besides the academia works, COIN also draws many interests from industry. Firstly, IETF now has a COINRG [147] exploring this topic. Its goal is to investigate how to benefit from this emerging disruption to the Internet architecture to improve network as well as application performance [147]. Several drafts are published already by COINRG to discuss promising use cases and open questions of COIN.

COINRG published a draft about targeted use cases for COIN [143]. In this draft, accelerating large volume applications in modern industrial networks can be promising use case of COIN. As described in the draft [143], end devices (e.g. industry sensors or cameras) in modern industry networks can generate a large volume of data that can not be efficiently processed by end devices themselves. Off-premise cloud platforms offer promising and cost-effective solutions with better flexibility and scalability. However, there's no free lunch, transmitting large volume data to remote cloud platform poses new challenges, especially for the latency performance. Preprocessing or filtering data with in-network computing can be a very promising solution to address the latency challenge. So this work exactly looks into this scenario and proposed a prototypical system covers the core questions: (i) How to pre-processing large volume video data with reasonable complexity and reduce the volume as much as possible ? (ii) What network protocols should be implemented and deployed to enable this application?

COINRG also published a special draft [148] discussing the open challenges related to the transport layer for COIN applications. COIN breaks one fundamental consideration for conventional Internet, namely the end-to-end principle. The network should only perform transparent and reasonable operations without modify-

ing data packets. Therefore, typical transport protocols like TCP are not designed taking COIN into consideration. Several open challenges including addressing, flow granularity, collective communication and transport features are listed in the draft without concrete solutions. So this work explores the opportunity to address these open challenges and propose solutions as much as possible. The proposed design is implemented and evaluated on network emulator to show its practicality.

6.2 Proposed Approach: You Only Look Once, but Compute Twice (YOLO-CT)

The design of the proposed system, called You Only Look Once, but Compute Twice (YOLO-CT), is based on the targeting use case: Interactive and real-time Computer Vision (CV) application based on cloud computing and Deep Learning (DL). Examples of this application type include real-time and low-latency video streaming for surveillance or robot control. It should be highly noted here that this type of applications has the following characteristics that have significant impact on the system design:

1. These applications favor timeliness over reliability. Delivering video frames is very sensitive to latency. If the frames are not delivered to the cloud in time, the object detection results may be meaningless. Therefore, retransmitting packets or reducing the sending rate has a non-negligible impact on the functionality of the application. In order to guarantee the QoS and responsiveness, the sender needs to generate Constant Bit Rate (CBR) traffic. These two requirements make it challenging to trade-off latency and bandwidth using some conventional approaches based on queue management or batch processing.

2. Unlike the transmission of control messages, video streaming applications have relative high bandwidth requirements. Even though the video frames can be compressed at the source, the compressed frame still needs to be fragmented into multiple network packets. If the network functions want to perform smart computing on the ADU, it can not just assume all needed data in an ADU can be packed into a single network packet. However, most papers related to COIN use this assumption and mention that this is a strict limitation that should be addressed in their future work or other research works. As described in [148], conventional devices are built to process incoming traffic on a per-packet basis with very limited support for stateful traffic processing. For payload-aware processing, flow granularity must be determined to enable network nodes to reassemble the ADU and re-fragment processed payload into new packets. For payload-aware smart computations, the granularity of the flow must be determined so that the network nodes can reassemble the ADUs and re-fragment the processed ADU into new packets.

3. High performance Deep Learning (DL) based applications have high requirement for computational resource. For example, some gigabits (GBs) of RAM are needed to fully load the neural network model. It is challenging to do low

latency and energy efficient DL model inference on end devices or network devices with limited computational resources and features.

6.2.1 YOLO-CT Design and Architecture

These essential requirements listed above really limit the choices with STOA accessible technologies. The proposed the YOLO-CT system, which is illustrated in Figure 6.3, has the following characteristics to address the aforementioned challenges:

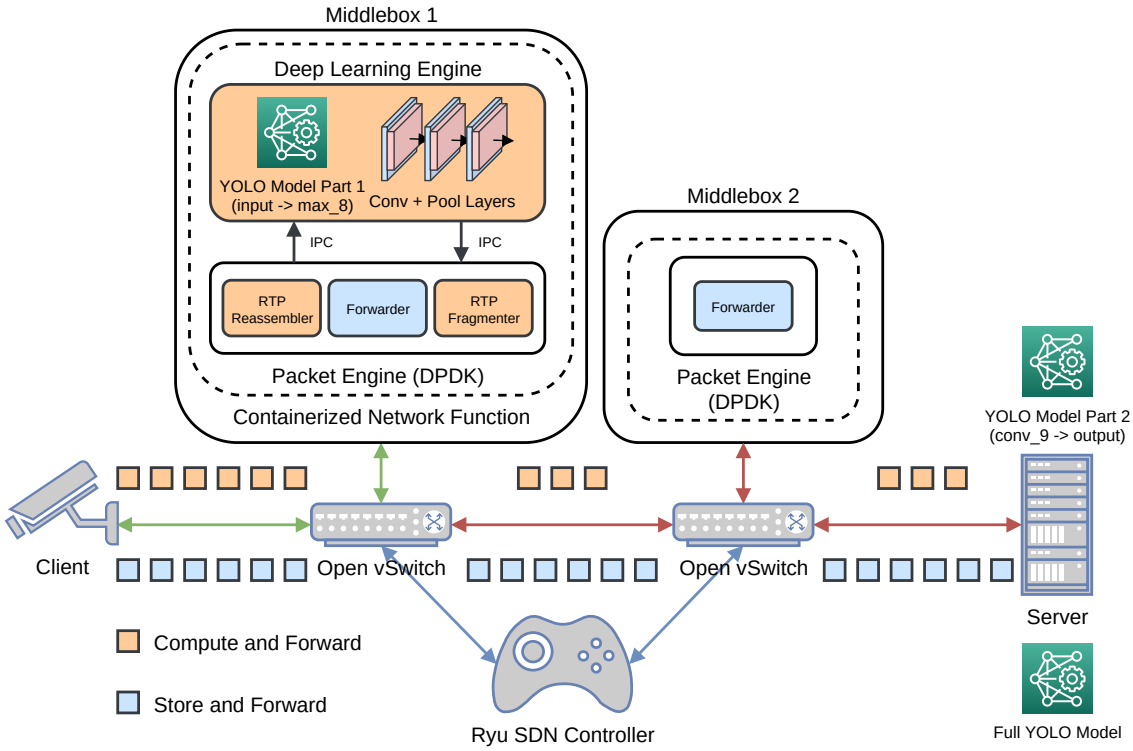


Figure 6.3: The proposed approach You Only Look Once, but Compute Twice (YOLO-CT). A detailed illustration of the system components and traffic flows.

1. The proposed system prioritizes latency performance over throughput and reliability. It utilizes the computing resources in the network to reduce end-to-end latency. Based on this consideration, mechanisms based on retransmission or aggressive reduction of sender speed are not considered in this work. The idea of COIN is explored here and perform *compute and forward* (in detail, feature extraction using ANN) to reduce the amount of data that must be transmitted through the network. Instead of tuning batching or scheduling mechanisms, the proposed containerized network function explore the counter-intuitive idea if reducing end-to-end response latency by deliberately buffering and processing packets before forwarding them, when the network nodes are under heavy load. The response latency is the time between the client finish sending the frame until it receives the response from the server, so namely it is the end-to-end latency including all network and compute latencies. The

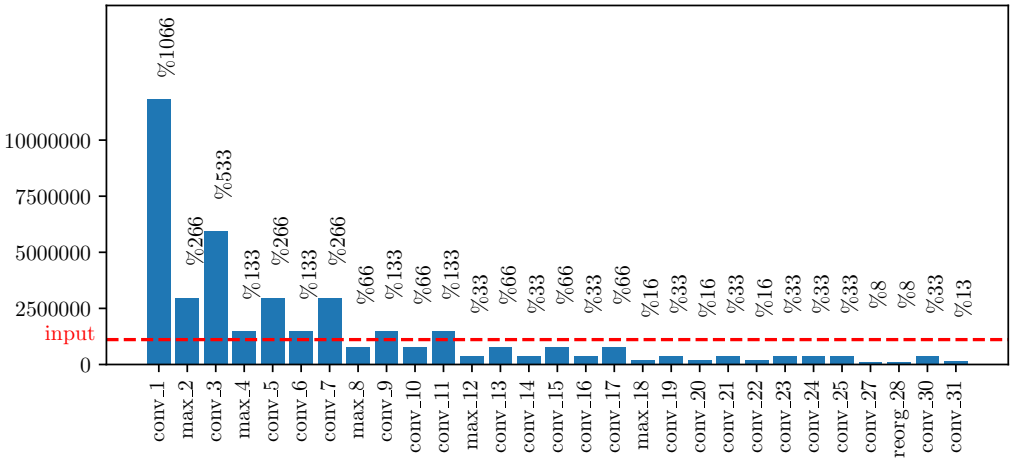


Figure 6.4: Output size of each layer of the YOLO-v2 model. Reprinted from **my journal paper** [10].

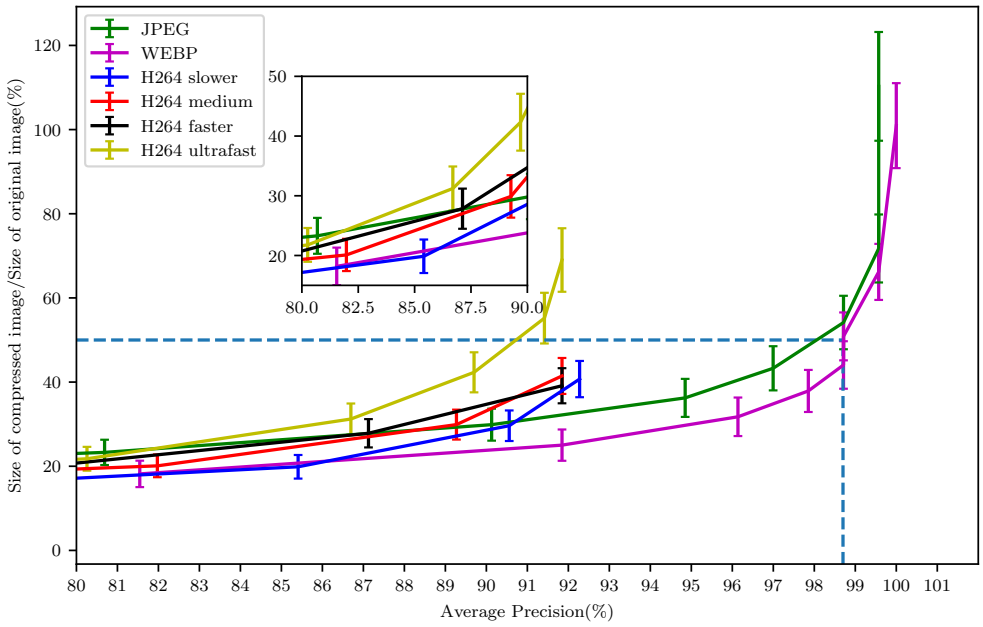


Figure 6.5: Basic image-based compression methods for feature maps. Reprinted from **my journal paper** [10].

innovative method used to reduce the data is to split the convolutional neural network model of YOLO and offload relative lightweight layers (namely convolu-

tional and pooling layers) to extract feature maps from the source coded image frame. Then instead of forwarding all packets of the original image frame, the network function only send the compressed feature maps extracted from the image to the next hop. Compared to the source coded image frame, these feature maps can be further compressed. According to Figure 6.4 and corresponding detailed description in my published journal paper [10], at least 50% of the data can be compressed with negligible impact on the performance of the object detection application. The reason for negligible impact on the object detection application is because the DL model running on the server node actually just wants feature maps instead of the original image to further process and perform object detection and classification. Therefore, the performance (precision and recall) of this application is not impacted when offloading this feature extraction process to the network nodes. The performance of the application suffers only when the feature maps are further compressed with lossy methods. However, according to Figure 6.5 and corresponding detailed evaluation in my journal paper [10], the impact of the performance is very small (less than 2%) when feature maps are simply flatted and compressed with Joint Photographic Experts Group (JPEG). The proposed network function is fully implemented with high performance software technologies and containerized with Docker. Therefore, it can be deployed on any NFV network nodes. The design is to deploy this feature extraction NF to the nearest NFV node to client node. Then, all subsequent nodes in the network have much less traffic to queue and handle. The end-to-end latency performance can be improved.

2. In order to enable the proposed in-network computing NF for video traffic, existing and newly proposed network protocols are surveyed and analyzed. As fully described in a recent draft [148] published by the COINRG in the IETF, conventional end-to-end design (dumb network) principle and transport layer protocols brings big challenges to COIN applications. Several open questions are raised in the draft without solutions, including addressing, flow granularity and transport features. This draft draws some research interests and there are publications [146, 149] that work on completely new clean-slate transport layer protocols to address those open questions. It is argued in this work that instead of introducing yet another new transport layer protocol, essential parts of the already standardized application protocols for real-time streaming protocols can be implemented in the network function to provide minimal and sufficient features for COIN applications. The reason for avoiding new transport protocols is clear: All nodes in the network including clients and serves need to support this new transport protocol. This means that a solid implementation of this transport stack must be available for all types of network nodes. According to the adoption of the Stream Control Transmission Protocol (SCTP) protocol, introducing a new transport protocol just to provide some limited support for COIN applications may be worth the significant effort. Then the same direction of the relative successful protocol Quick UDP Internet Connections (QUIC) is followed to select a protocol built on top of UDP for this real-time communication task. After some comprehensive survey and analyse, RTP is chosen as a

promising protocol to explore for COIN applications that smartly process packets and drop unnecessary data directly and early in the network. According to my point of view, RTP is the most suitable standardized protocol (or one of the most suitable standard protocols) for the aforementioned applications and it can be utilized and extended for COIN applications:

a) RTP protocol uses application-layer framing [150]. This allows applications and also network nodes to handle ADUs. Its standardized mechanisms for ADU fragmentation and reassembling solves the flow granularity problem described in [148].

b) RTP has a IETF working group for congestion control mechanisms for real-time media. So the proposed network function can adopt these mechanisms instead of re-inventing congestion control mechanisms that do not work well with e.g. TCP.

c) Like QUIC, RTP is built on top of UDP and normally implemented fully in user space. This enables quick and simple iteration of the protocol itself and its implementation. This makes it easier to utilize the mature user space acceleration technologies and be deployed and maintained on the network edge.

3. Instead of using programmable hardware switches or routers. Middlebox with containerized network functions.

Before the introduction of the implementation of the proposed YOLO-CT system, a theoretical modelling of the response latency, which is the key latency metric for this work, of the topology illustrated in Figure 6.3 is described in the next Subsection 6.2.2. This modelling targets at the demonstration of why this counter-intuitive design of YOLO-CT approach is able to reduce the service latency, when underlying network is overwhelmed with high traffic workload.

6.2.2 Modelling of Service Latency

A typical remote cloud based application has different types of latencies. These latencies contain two parts: latencies in the packet-switched network and latencies on end hosts. Network latencies typically contain four parts: propagation delay, transmission delay, queuing delay and processing delay. Latencies on end hosts contain packet IO latency to get all packets and then processing latency for ADU. For the end-to-end response latency modelling, following simplifications are applied in this work: (i) The packet IO latency on the server side can be ignored because it's relative small. (ii) The response message by the server, e.g. the object detection result, is relative small compared to the video frame message uploaded by the client. So it can be packed into a single packet. (iii) For simplification, it's assumed the network latency performance is symmetric. Namely, all types of latencies for a single packet are the same for both client to server and server to client directions.

Table 6.1: YOLO-CT: Summary of main notations.

Symbol	Description
M	Number of hops in the multi-hop topology
i	Index of the current hop
N	Number of packets contained in an ADU
P	Number of packets contained in a processed ADU
j	Index of the current packet
T_{stor}^{resp}	Response latency for store and forward
T_{comp}^{resp}	Response latency for compute and forward
$T_{i,j}^{tran}$	Transmission latency of j-th packet on i-th hop
$T_{i,j}^{queu}$	Queuing latency of j-th packet on i-th hop
$T_{i,j}^{proc}$	Processing latency of j-th packet on i-th hop
T_{i}^{prop}	Propagation latency of i-th hop
T_{serv}^{comp}	Computation latency of ADU on the server
T_{N}^{reas}	Latency to reassemble N received packets
T_{nf}^{comp}	Computation latency of ADU on the network function
T^{frag}	Latency to re-fragment processed ADU
α	Ratio between computing power between NF and server

$$
\begin{aligned}
T_{stor}^{resp} = & \sum_{j=1}^{N} \left(\sum_{i=1}^{M} \left(T_{i,j}^{prop} + T_{i,j}^{queu} + T_{i,j}^{proc} \right) \right) \\
& + T_{serv}^{comp} \\
& + \sum_{i=1}^{M} \left(T_{i,j}^{prop} + T_{i,j}^{queu} + T_{i,j}^{proc} \right)
\end{aligned}
\tag{6.1}
$$

For a multi-hop topology with M network nodes, the end-to-end response latency of a single flow T_{stor}^{resp} is formulated in Equation 6.1, when all network nodes only perform *store and forward* for all N packets. The N is the number of individual packets for one video frame (ADU). Due to limitation of the MTU size of the network, one large video frame typically has to be fragmented into multiple RTP packets. Each packet needs to be transmitted, queued and processed by all M network nodes in the path, so these latencies for j-th packet on the i-th hop are denoted as $T_{i,j}^{tran}$, $T_{i,j}^{queu}$ and $T_{i,j}^{proc}$. The first line in Equation 6.1 presents the network latency from client to server. The second line expresses the computation latency on the server for ADU processing. The third line expresses the network latency for the response from server to client.

$$
\begin{aligned}
T_{comp}^{resp} = & \\
& \sum_{j=1}^{N} \left(T_{1,j}^{prop} + T_{1,j}^{queu} + T_{1,j}^{proc} \right) \\
& + T_{N}^{reas} + T_{nf}^{comp} + T^{frag} \\
& + \sum_{j=1}^{P} \left(\sum_{i=2}^{M} \left(T_{i,j}^{prop} + T_{i,j}^{queu} + T_{i,j}^{proc} \right) \right) \\
& + (T_{serv}^{comp} - \alpha \cdot T_{nf}^{comp}) \\
& + \sum_{i=1}^{M} \left(T_{i,j}^{prop} + T_{i,j}^{queu} + T_{i,j}^{proc} \right)
\end{aligned}
\tag{6.2}
$$

Assuming that the proposed feature extraction network function is deployed on the first and nearest network node, the corresponded response latency T_{comp}^{resp} is formulated with Equation 6.2. By the way, it is also my recommendation to deploy this network function to the nearest network node as much as possible. Mainly for two reasons: (i) All following hops have to process less packets, so it's beneficial for latency. Also, the latency tax paid to collect N packets on the first hop is relative smaller. (ii) In reality, N packets can be distributed over multi-path in the transmission. Normally there's no multi-path routing between the client and the first network node, namely the gateway, so it's doable for the network to collect N packets for ADU processing. Compared to the *store and forward*, there are multiple differences: (i) Additional latencies are introduced on the first network node to enable the computing on the ADU which is split into N packets, as expressed by the first two lines. Additional latencies include four parts: (1) Additional time to collect N packets in an ADU. Because the network function needs to collect N packets before it can start the processing. So it can not just store and forward each individual packet separately. So we need to sum up the latencies on the first hop for all N packets (the first line in the Equation). (2) Additional time to reassemble the collected fragments T_{N}^{reas}. (3) Additional time to process the reassembled ADU T_{nf}^{comp}. (4) Additional time to re-fragment the processed ADU into new RTP fragments T^{frag}. (ii) The number of packets is reduced from N to P for all following network nodes. If the feature extraction function has a very good compression performance, then $P \ll N$. The network latencies for P packets along the path can be significantly reduced compare to the original N packets. (iii) Because part of the application function, namely the feature extraction, is already performed on the network node, the computing latency on server can be reduced to $(T_{serv}^{comp} - \alpha \cdot T_{nf}^{comp})$. The reason to have a factor α (with $0 < \alpha \leqslant 1$) is because the computational power of cloud server and edge network node is normally not equal. Cloud server is normally more powerful than network node. So $alpha = \frac{1}{2}$ means that the server has twice the computing power of the network nodes for the same task. So when the network node needs T_{nf}^{comp} to perform the feature extraction function, same operation needs normally less computational time when it's deployed on the server. So the computational time it saves

for the server is actually $\alpha \cdot T_{nf}^{comp}$.

$$\begin{aligned} T_{diff} &= T_{stor}^{resp} - T_{comp}^{resp} \\ &> -(1-\alpha) \cdot T_{nf}^{comp} - T_{N}^{reas} - T^{frag} \\ &- \sum_{j=1}^{N} (T_{1,j}^{prop} + T_{1,j}^{queu} + T_{1,j}^{proc}) \\ &+ \left(\sum_{j=P}^{N} \left(\sum_{i=2}^{M} \left(T_{i,j}^{prop} + T_{i,j}^{queu} + T_{i,j}^{proc} \right) \right) \right) \end{aligned} \tag{6.3}$$

The latency difference, namely the gain of the proposed *compute and forward* approach, can be expressed with the Equation 6.3. By analyse the Equation 6.3, it can be seen that for *compute and forward*, the availability and size of the latency performance gain mainly depends on two parts: (i) Negative earnings: The additional latencies introduced on the first node to enable ADU processing and payload compression. These latencies are like latency tax we need to pay and are expressed in first two lines in the Equation. (ii) Positive earnings: Reduced network latencies at subsequent network nodes in the multi-hop path. This part is expressed as the last line in the Equation. So there is a trade-off here to determine whether there is really a latency gain, namely $T_{diff} > 0$. Given a network topology with fixed parameters and a practical implementation of the feature extraction network function, the first part of latency is almost fixed. The second part, namely the networking latency, depends heavily on the status of the network. For example, this part can be increased when one of the subsequent network nodes do not have enough resource to handle the flow (noisy neighbor or scheduling issue or overwhelmed).

So the proposed approach YOLO-CT suggests switching between *store and forward* and *compute and forward* dynamically. The network status can be monitored with e.g. SDN controller or other SDN mechanisms. The expression 6.3 shows that there's a trade-off between positive and negative earnings in the end-to-end response latency. This is also reflected in the evaluation result, the *store and forward* is not always worse than the proposed *compute and forward*. When network status is good, there is no latency gain.

6.2.3 YOLO-CT Implementation

A demonstration of the proposed end-to-end system is illustrated in Figure 6.3. According to the design, network nodes and server are aware of the offloaded computing in the network while this is totally transparent to the clients. There are discussions [148] about whether and how clients can address and control in-network computing nodes. The proposed approach suggests that this type of *compression and forward* offloading should be transparent to clients. This avoids adding additional complexity of the networking system on edge clients, which are typically not very powerful. Clients send captured video frames in RTP packets and wait for the object detection results in the response. All traffic flows are managed by the SDN

controller. The controller can add additional OpenFlow rules to redirect RTP flows from the clients to the middlebox for packet processing. So SDN controller is responsible to monitor the network and trigger computing and forward on demand. The components to enable feature extraction network function are implemented in the containerized network function.

These components are plotted in the middlebox 1 in Figure 6.3. The network function contains two main components: packet engine and deep learning engine. (i) The packet engine based on DPDK acceleration is used for packet IO, RTP ADU reassembling and re-fragmenting. This component needs to perform high performance packet processing. It uses DPDK to capture each packet with low latency and buffer packets (RTP fragments) belonging to the same frame. When the network function receives the last fragment of the frame, it uses the RTP reassemble module to reassemble the ADU and extract the frame payload for further feature extraction. Since DPDK does not provide functionalities for this type of stateful processing and ADU related operations, RTP reassembler and fragmenter are implemented by us on top of DPDK. Since this is just a prototype, but the implementation is adapted to the principles of DPDK and tries to achieve low latency performance as much as possible. This prototype also shows that it's doable to implement part of RTP stack on top of high performance software framework designed for NFV forwarding plane. This prototype also shows that it is feasible to implement a partial RTP stack on top of a high-performance software framework designed for the NFV forwarding plane. It is much less complex than implementing a generic TCP or SCTP protocol stack. The reassembled frame is then pushed to the deep learning process via IPC and the packet engine waits and pulls the extracted and compressed feature maps from the deep learning component. Then the packet engine re-fragments the compressed feature maps into new RTP fragments and update the content of corresponded received RTP fragments with new fragments. Because this feature extraction processing can remove about 50% of the redundant data, the packet engine only needs to send about half the number of RTP fragments. (ii) The deep learning component is implemented using `Tensorflow`. The pre-trained YOLO-v2 model is split into two parts. The feature extraction layers deployed on the network function contain the first eight layers of the full model, namely from input to `max_8` pooling layer. When the deep learning component starts, it loads the pre-trained and split model file and waits for the packet engine pushing frames for processing. When it receives one frame, it performs the partial model inference and compress the output feature maps with JPEG. Then it pushes compressed feature maps as one message back to the packet engine.

All source code of the You Only Look Once, but Compute Twice (YOLO-CT) prototype is publicly available in the GitHub repository [151].

6.3 Comparison with Clean-slate Message Transport Protocol (MTP)

Firstly, it is analysed here whether the MTP protocol proposed in [149] is revolutionary or is really necessary for most COIN applications, especially applications that require low latency. In [149], several important requirements are identified to support in-network computing. As the comparison performed in [149] shows, UDP protocol already satisfies the requirements for data mutation, low buffering requirements and inter-message independence.

The open main challenges are: (i) How to apply application layer framing when ADU has to be fragmented into multiple packets due to MTU limitation. (ii) How to enable multi-entity isolation. (iii) How to perform congestion control with COIN into consideration.

For the issue (i), both RTP and MTP apply almost the same mechanism. In RTP, each packet contains a fragment offset field which stores the offset in bytes of current packet in one ADU. One ADU can be packed into a group of RTP packets, namely one RTP message, with the same timestamp. The RTP common header also has a marker bit to indicate the last packet in a ADU. These fields enable efficient fragmentation and reassembly of ADUs in the presence of misordered or lost packets. Fragmentation and reassembly processes are simple and described in RTP related RFCs. Compared to RTP, MTP protocol header also has a unique message ID for each message. The header also has fields that describe the offset of the current packet and the current packet number [149]. So the reassembly and fragmentation mechanisms should be the same as ones used by RTP, even though the details are not clearly described in [149]. Because of this similarity, the reassembly and refragmentation latencies of MTP can not be better than what RTP already provides. Regarding this latency performance, MTP has no clear gain except for the clean-slate design. Regarding the issue (ii), both RTP and MTP provides a special field to identify source entity for isolation. So there's no hard requirement to use MTP just to achieve entity isolation.

Regarding the issue (iii), this is a new feature proposed by MTP. Common transport layer protocols like TCP and UDP do not have this feature. MTP proposes to use a so-called pathlet congestion control. So instead of taking the multi-hop path between source and destination as a whole unit, this path is fragmented into multiple smaller pathlets. So conventional congestion control only track the congestion status of end hosts. Pathlet based congestion control track congestion state of each pathlets using its own type of congestion feedback. It is described in MTP that because of the heterogeneous computational power of COIN elements in the network path, the congestion state of each pathlet can be much different from each other. So MTP proposes to use pathlet based congestion control to apply different congestion control algorithms to coexist, instead of applying only one algorithm based only on end hosts. Congestion state of each pathlet needs to be stored by an Type-Length-Value (TLV) in the MTP header. So, although pathlet based congestion control provides more flexibility and granularity for network with COIN applications, additional non-negligible packet header overheads and management complexity are introduced.

In [149] the performance of the proposed pathlet congestion control algorithm is not evaluated. So whether this complexity is worth it is still an open question.

As discussed in [148], whether in-network COIN elements should be covered by congestion algorithm and participate in end-to-end flow control is an open research question. I think this change can introduce a lot of complexity and complicate the design and implementation of network functions. So in this work, the congestion control is performed only on end hosts. Computational network functions are designed and implemented to finish the ADU processing in time to not impact the end-to-end flow and congestion control.

6.4 YOLO-CT Evaluation and Measurement Results

To evaluate the end-to-end response latency performance and resource usage of the proposed system, the proposed system in Figure 6.3 is fully implemented and evaluated on the ComNetsEmu [118] network emulator introduced in Chapter 5. ComNetsEmu is deployed inside a KVM virtual machine managed by `libvirt`. The proposed network function is packaged as a Docker container and managed by the ComNetsEmu in the evaluation. The host machine has an Intel Core i7-7820HK CPU @ 2.90GHz and 32 GB DDR4 RAM. All source code needed for the evaluation is publicly available in the GitHub repository [151]. The evaluation contains two parts: (i) The evaluation of the proposed feature extraction network function. (ii) The evaluation of the end-to-end response latency of the topology presented in Figure 6.3.

6.4.1 Feature Extraction Network Function

The evaluation of the proposed feature extraction network function focuses on the data compression ratio and the resource usage.

The design goal of this offloaded network function is to have nearly zero impact on the original end-to-end object detection application. However, because the raw feature maps normally have much bigger size than the compressed image frame from source with e.g. JPEG, the raw feature maps are lossy compressed in this work to achieve a reasonable compression ratio. A trade-off need to be played between the compression ratio and the performance of the object detection application. The JPEG compression is used in this work for feature maps and the performance is evaluated with the `COCO` dataset [152] 2017 version. As analysed in my previous work [10] and illustrated in Figure 6.5, a compression ratio of 2 can be achieved with less than 2% reduction in detection accuracy. Therefore, this compression method with an average compression ratio of 2 is chosen in this work.

The resource usage comparison between the proposed feature extraction network model and the full YOLO-v2 model is listed in the Table 6.2. The CPU time is the average inference time of the neural network model of 20 iterations. The memory usage is the measured Resident Set Size (RSS) during runtime. The model file size is the size of the protobuf file used by `Tensorflow` to store the model on disk. As showed in the Table 6.2, the proposed model splitting and feature extraction can

significantly reduce the memory and disk space required by the network function. Compared to the full model, only 450 MB memory is needed, namely about 21% of the full model. The RSS memory usage is important because the high performance memory is a relative limited resource on network nodes and are shared by all deployed network functions. Also, the size of the serialized model file is very small. It only takes 696 KB to store or transport the model file, which is about 0.35% of the file size of the full model. The reduced memory, disk and network transport sizes can benefit the deployment of the proposed network function on the edges of the network. The CPU processing time of the feature extraction function takes about 64% of the full model inference time. Therefore, the processing workload of the server in the cloud in reduced by about 64% for each input RTP flow. The saved CPU processing time on the server can be used to handle more RTP flows or can be used by other services. So all in all, the proposed feature extraction network function can significantly reduce the output bandwidth and the workload of the server in the cloud with reasonable memory and disk usage.

Component	CPU Time	Memory Usage	Model File Size
Feature Extraction Model	292 ms	450 MB	696 KB
Full YOLOv2 Model	459 ms	2116 MB	199184 KB

Table 6.2: Resource usage comparison between the feature extraction model and the full YOLOv2 model.

6.4.2 End-to-end Response Latency

The end-to-end response latency is the delay between the client sends out the frame until it receives the response with object detection result from the server. So it is the RTT measured on the client side which contains all networking and computing latencies of the network and cloud server. In the latency tests, the same JPEG image (`pedestrian.jpg` in the repository [151]) with a size of 48 KB is sent by the client repeatedly to measure the response latency. For all response latency measurements are measured with the minimal dumbbell topology illustrated in Figure 6.2 with two switches and middleboxes in the network. All nodes and network links are emulated with ComNetsEmu with following parameters: (i) Client, server and middleboxes are pinned on different CPU cores to avoid contention of computing resource. (ii) The link parameters are emulated with Linux Traffic Control (TCP). To build the dumbbell bottleneck, the bandwidth and propagation latency of the link 4 is set to 100 Mbit/s and 100 ms. Links between switches and edge nodes, i.e. the client and the server, are configured with a bandwidth of 1000 Mbit/s and a latency of 10 ms. Because NFV middleboxes are usually deployed close to the switch, links between middleboxes and switches are configured with a bandwidth of 1000 Mbit/s and a latency of 1 ms.

Two tests are performed on this network topology: (i) Without background workload: In this scenario, only client 1 sends frames to server for object detection. There is no any other traffic in the network. In this scenario, a batch forwarding delay is

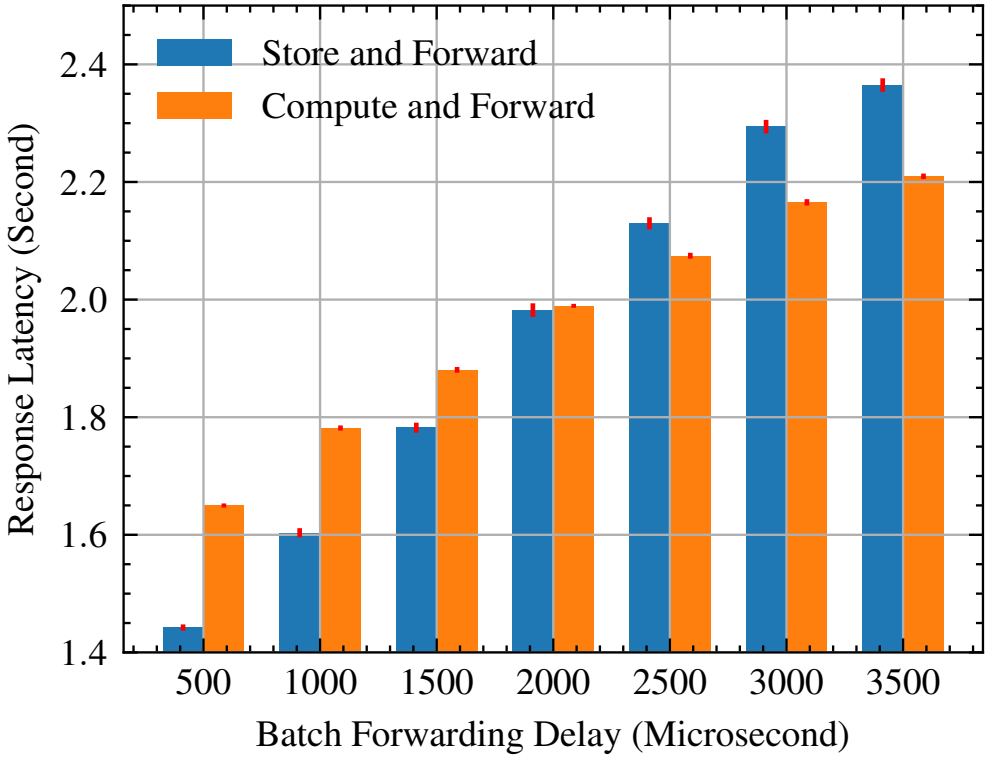


Figure 6.6: YOLO-CT: Response latency without background workload.

manually added to the forwarding CNF running on the middlebox 2 to evaluate the impact of busy level of the middlebox 2 on the end-to-end response latency. Because compared to the edge switch 1, the switch 2 is deployed closer to the cloud and needs to handle much more input links compared to switch 1. Due to dynamic workload changing or system scheduling challenge, a forwarding delay of a batch of packets can be introduced for the CNF running on middlebox 2. The detailed distribution of this latency is not clear, so this latency is manually added from 500 us with a step of 500 us. (ii) With background workload: In this scenario, client 2 uses `sockperf` tool from Mellanox to generate UDP background workload with payload size of 1400 B and different PPS. In this scenario, the batch forwarding latency of the CNF running on middlebox 2 is fixed to 2000 us. These two tests are used to evaluate the latency of the proposed approach in both good and bad network congestion conditions. In all tests, the client 1 sends RTP fragments of the `pedestrian.jpg` image to the server in a ping-pong test mode. Each test is repeated for 100 times to get the average and 99% confidence intervals.

The response latencies without background workload are plotted in Figure 6.6. As showed in the results, the *compute and forward* approach has a higher latency compared to *store and forward* when the batch forwarding delay is lower than 2000 us. So the additional reassemble, fragmentation and processing latencies introduced by the feature extraction network function are higher than the transmission, queuing and processing latencies it saved. So in this scenario, there's no gain in the latency perspective. But when the batch forward latency is higher than 2000 us, the *com-*

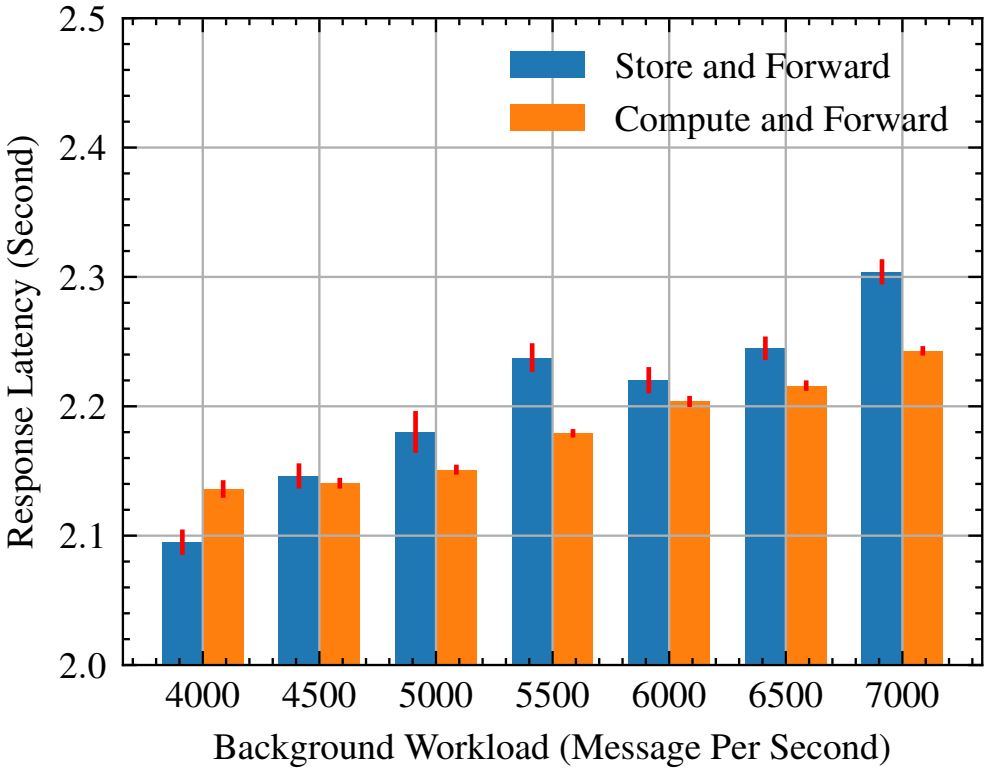


Figure 6.7: YOLO-CT: Response latency under background workload.

pute and forward approach starts to show lower latency than the conventional *store and forward*. So the CNF running on middlebox 1 can switch between two modes dynamically based on the busy level of the following network nodes.

The response latencies with background workload are illustrated in Figure 6.7. The results show that the *compute and forward* approach can provide lower and also stabler response latency performance when there is a noisy neighbor in the network aggressively generating traffic. Latency results from this test shows the important benefit that can be achieved with my approach when network resources are shared by multiple clients. So it improves the scalability of the network system with in-network computing.

6.5 Summary

In this work, for the real-time video streaming object detection application illustrated in Figure 6.2 a novel approach called You Only Look Once, but Compute Twice (YOLO-CT) is proposed to reduce the amount of traffic required to be sent through the network. All requirements are analyzed and components to build the proposed system with STOA software technologies and network protocols. The proposed approach is compatible with the standard real-time video streaming system. The initial prototype uses the RTP, which is the IETF standard used for real-time and low latency multimedia applications. A prototype of the proposed system is designed and implemented

with STOA software packet processing technologies. Source code of all components used in this work is fully open source [151]. Experiments can be easily reproduced on a single computer.

Proposed system is evaluated using network emulation with ComNetsEmu. Dumbbell topology showed in Figure 6.2 is built and emulated on ComNetsEmu testbed. According to the initial measurement results illustrated in Figure 6.6 and Figure 6.7, the described YOLO-CT approach can improve the latency performance when network nodes are busy or congested.

7 Summary

As described in the Section 1.2, this dissertation describes and summarizes four research works I have conducted to address the unprecedented and challenging open question for softwarization network of 5G and beyond:

How to significantly reduce the latency of State of the Art (STOA) softwarization network data plane to meet the 5G stringent end-to-end latency requirement of 1 ms with minimal negative impact or even improvement on other important performance metrics, especially bandwidth and energy consumption?

As graphically illustrated as a Triforce in Figure 1.4, the works and achievements included in this dissertation contribute to the improvement of the STOA software network data plane systems in three different directions:

1. Ultra-reliable low-latency: As described in Chapter 3 and in my journal paper [4], the Chain bAsed Low latency VNF ImplemeNtation (CALVIN) system is designed, implemented and evaluated for ultra-reliable low-latency tactile Internet applications. In contrast to conventional solutions, CALVIN implements classified Virtualized Network Functions (VNFs) either fully in Linux kernel space or in user space in order to fully avoid the context-switching and data transmission between these two spaces. According to rigorous measurements, for the elementary forwarding function, the eXpress Data Path (XDP) based VNF implementation is able to achieve a packet forwarding performance ranging from 120 μs to 180 μs. CALVIN is able to achieve an end-to-end RTT of only 0.32 ms for large packets with the size of 1400 bytes on an OpenStack-based practical cloud testbed. Therefore, the proposed CALVIN approach can achieve the 1 ms latency budget required by tactile Internet applications at the cost of reduced bandwidth support.

2. Energy-efficient: As described in Chapter 4 and in my journal paper [7], the novel XDP-Monitoring energy-Adaptive Network functions (X-MAN) framework is designed, implemented and evaluated for power management of Cloud-native Network Functions (CNFs). X-MAN combines the non-intrusive in-band in-kernel traffic monitoring for each individual CNF with the responsive and adaptive core frequency management in user space with a global view of all

running CNFs. According to rigorous measurements on practical testbed, the proposed X-MAN is much more responsive than the STOA Hardware Counter (HC) approach. Measurement results also indicate that X-MAN is able to save much more energy compared to the Code Instruction with Heuristic power management (CIH) approach, while has negligible impact on the latency performance.

3. Computing-centric: As described in Chapter 6 and partly in **my journal paper** [10], a novel approach named as You Only Look Once, but Compute Twice (YOLO-CT) is designed and implemented which utilizes the COmputing In Network (COIN) paradigm supported by the softwarized network to significantly reduce the amount of data required to be sent through the network by offloading part of the Convolutional Neural Network (CNN) model directly into the network nodes with computing power and functionalities. According to the evaluation using the ComNetsEmu network emulator, the proposed YOLO-CT is able to improve the end-to-end response latency performance when network nodes are overwhelmed or congested.

Acronyms

3GPP	3rd Generation Partnership Project.
ADU	Application Data Unit.
AES	Advanced Encryption Standard.
AIMD	Additive-Increase Multiplicative-Decrease.
AMD	Advanced Micro Device.
ANN	Artificial Neural Network.
API	Application Programming Interface.
APM	Adaptive Polling Mechanism.
ARM	Advanced RISC Machines.
ARP	Address Resolution Protocol.
BCC	BPF Compiler Collection.
BESS	Berkeley Extensible Software Switch.
BIOS	Basic Input/Output System.
BMAP	Batch Markovian Arrival Process.
BS	Base Station.
CALVIN	Chain bAsed Low latency VNF ImplemeNtation.
CAPEX	Capital Expense.
CBR	Constant Bit Rate.
CDF	Cumulative Distribution Function.
CI	Code Instruction.
CIH	Code Instruction with Heuristic power management.
CNF	Cloud-native Network Function.
CNN	Convolutional Neural Network.
COIN	COmputing In Network.
COINRG	COIN Research Group.
ComNets	The Deutsche Telekom Chair of Communication Networks.

ComNetsEmu	Communication Networks Emulator.
COTS	Commercial Off-The-Shelf.
CPU	Central Processing Unit.
CRUD	Create, Read, Update and Delete.
CV	Computer Vision.
DHCP	Dynamic Host Configuration Protocol.
DIND	Docker-IN-Docker.
DL	Deep Learning.
DMA	Direct Memory Access.
DPDK	Data Plane Development Kit.
DPI	Deep Packet Inspection.
DuT	Device under Test.
DVFS	Dynamic Voltage and Frequency Scaling.
eBPF	extended Berkeley Packet Filter.
ENVI	Elastic resource flexing for Network function VIrtualization.
ETSI	European Telecommunications Institute.
FC	Flow Classifier.
FFPP	Fast Forward Packet Processing.
GF	Galois Field.
GPU	Graphics processing unit.
GRE	Generic Routing Encapsulation.
HC	Hardware Counter.
HTTPS	Hypertext Transfer Protocol Secure.
ICA	Independent Component Analysis.
ICMP	Internet Control Message Protocol.
ICN	Information-centric Networking.
ID	identification Number.
IETF	Internet Engineering Task Force.
IMIX	Internet Mix.
INC	In-Network Computing.
IO	Input/Output.
IOMMU	Input-Output Memory management Unit.
IoT	Internet of Things.
IP	Internet Protocol.
IPC	Inter-Process Communication.
IPG	Interpacket Gap.
ISG	Inter-Stream Gap.
ISP	Internet Service Provider.

IT	Information Technology.
JPEG	Joint Photographic Experts Group.
KNI	Kernel Network Interface.
KVM	Kernel-based Virtual Machine.
LAN	Local Area Network.
LKF	Linux Kernel IP Forwarding.
M2M	Machine to Machine.
MAC	Media Access Control.
MEC	Multi-access Edge Computing.
ML	Machine Learning.
MTP	Message Transport Protocol.
MTU	Maximum Transmission Unit.
NAT	Network Address Translator.
NC	Network Coding.
NCKernel	Network Coding Kernel Library.
NF	Network Function.
NFV	Network Function Virtualization.
NFVI	NFV Infrastructure.
NIC	Network Interface Card.
NJICA	Network Joint Independent Component Analysis.
NPM	No Power Management.
NTP	Network Time Protocol.
OPEX	Operating Expense.
OS	Operating System.
OVS	Open vSwitch.
OVS-DPDK	Open vSwitch with DPDK Datapath.
OWD	One-Way Delay.
PacketGen	Packet Generator.
PDU	Protocol Data Unit.
PM	Power Manager.
PMD	Poll Mode Driver.
pNIC	Physical Network Interface Controller.
POSIX	Portable Operating System Interface.
PPS	Packet-per Second.
PSTN	Public Switched Telephone Network.
QoS	Quality of Service.

QUIC	Quick UDP Internet Connections.
R-CNN	Regions with CNN.
RAN	Radio Access Network.
RLNC	Random Linear Network Coding.
RSS	Resident Set Size.
RTC	Run-To-Completion.
RTP	Real-time Transport Protocol.
RTT	Round-trip Time.
RX	Receive.
SCTP	Stream Control Transmission Protocol.
SDN	Software-Defined Networking.
SF	Service Function.
SFC	Service Function Chaining.
SFC-OStack	Service Function Chaining on OpenStack.
SFI	Service Function Instance.
SFP	Small Form-factor Pluggable.
SIMD	Single Instruction Multiple Data.
SMA	Simple Moving Average.
SSE2	Streaming SIMD Extension 2.
STOA	State of the Art.
TCP	Transmission Control Protocol.
TCP	Traffic Control.
TLV	Type-Length-Value.
TM	Traffic Monitor.
ts	Timestamp.
TUD	Technische Universität Dresden.
UDP	User Datagram Protocol.
UDS	Unix Domain Socket.
UE	User Equipment.
URLLC	Ultra-Reliable Low Latency Communication.
vCPU	virtual CPU.
veth	Virtual Ethernet Device.
VM	Virtual Machine.
VNF	Virtualized Network Function.
vNIC	Virtual Network Interface Controller.
VPP	Vector Packet Processing.
VXLAN	Virtual Extensible LAN.
WEB	World Wide Web.

WMA	Weighted Moving Average.
X-MAN	XDP-Monitoring energy-Adaptive Network functions.
X-MAN-C1	X-MAN with C1 State Management.
X-MAN-FB	X-MAN with FeedBack.
XDP	eXpress Data Path.
YOLO	You Only Look Once.
YOLO-CT	You Only Look Once, but Compute Twice.

Bibliography

[1] Frank Fitzek, Fabrizio Granelli, and Patrick Seeling. *Computing in Communication Networks: From Theory to Practice*. Academic Press, 2020.

[2] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. "Network function virtualization: State-of-the-art and research challenges". In: *IEEE Communications surveys & tutorials* 18.1 (2015), pp. 236–262.

[3] Kamal Benzekki, Abdeslam El Fergougui, and Abdelbaki Elbelrhiti Elalaoui. "Software-defined networking (SDN): a survey". In: *Security and communication networks* 9.18 (2016), pp. 5803–5833.

[4] Zuo Xiang, Frank Gabriel, Elena Urbano, Giang T Nguyen, Martin Reisslein, and Frank HP Fitzek. "Reducing latency in virtual machines: Enabling tactile Internet for human-machine co-working". In: *IEEE Journal on Selected Areas in Communications* 37.5 (2019), pp. 1098–1116.

[5] Stuart Cheshire. "Latency and the Quest for Interactivity". In: *White paper commissioned by Volpe Welty Asset Management, LLC, for the Synchronous Person-to-Person Interactive Computing Environments Meeting*. 1996.

[6] *Technical Specification Group Services and System Aspects; Study on Communication for Automation in Vertical Domains (Release 16)*. 3GPP TR 22.804. 22.804 TR, V2.0.0. 3GPP. May 2018.

[7] Zuo Xiang, Malte Höweler, Dongho You, Martin Reisslein, and Frank HP Fitzek. "X-MAN: A Non-intrusive Power Manager for Energy-adaptive Cloud-native Network Functions". In: *IEEE Transactions on Network and Service Management* (2021).

[8] Harshit Gupta, Abhigyan Sharma, Alex Zelezniak, Minsung Jang, and Umakishore Ramachandran. "A {Black-Box} Approach for Estimating Utilization of Polled {IO} Network Functions". In: *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*. 2019.

[9] *DPDK Official Documentation,Sample Applications User Guides, L3 Forwarding with Power Management Sample Application*. https://doc.dpdk.org/guides-20.11/sample_app_ug/l3_forward_power_man.html. [Online; accessed 2022-02-18]. 2020.

[10] Zuo Xiang, Patrick Seeling, and Frank HP Fitzek. "You only look once, but compute twice: Service function chaining for low-latency object detection in softwarized networks". In: *Applied Sciences* 11.5 (2021), p. 2177.

[11] Zuo Xiang, Frank Gabriel, Giang T Nguyen, and Frank HP Fitzek. "Latency measurement of service function chaining on OpenStack platform". In: *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*. IEEE. 2018, pp. 473–476.

[12] Joel M. Halpern and Carlos Pignataro. *Service Function Chaining (SFC) Architecture*. RFC 7665. Oct. 2015. DOI: 10.17487/RFC7665. URL: https://www.rfc-editor.org/info/rfc7665.

[13] Deval Bhamare, Raj Jain, Mohammed Samaka, and Aiman Erbad. "A survey on service function chaining". In: *Journal of Network and Computer Applications* 75 (2016), pp. 138–155.

[14] Juliver Gil Herrera and Juan Felipe Botero. "Resource allocation in NFV: A comprehensive survey". In: *IEEE Transactions on Network and Service Management* 13.3 (2016), pp. 518–532.

[15] Ahmed M Medhat, Tarik Taleb, Asma Elmangoush, Giuseppe A Carella, Stefan Covaci, and Thomas Magedanz. "Service function chaining in next generation networks: State of the art and research challenges". In: *IEEE Communications Magazine* 55.2 (2016), pp. 216–223.

[16] Barbara Martini, Federica Paganelli, AA Mohammed, Molka Gharbaoui, Andrea Sgambelluri, and Piero Castoldi. "SDN controller for context-aware data delivery in dynamic service chaining". In: *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. IEEE. 2015, pp. 1–5.

[17] Ying Zhang, Neda Beheshti, Ludovic Beliveau, Geoffrey Lefebvre, Ravi Manghirmalani, Ramesh Mishra, Ritun Patneyt, Meral Shirazipour, Ramesh Subrahmaniam, Catherine Truchan, et al. "Steering: A software-defined networking for inline service chaining". In: *2013 21st IEEE international conference on network protocols (ICNP)*. IEEE. 2013, pp. 1–10.

[18] João Soares, Carlos Gonçalves, Bruno Parreira, Paulo Tavares, Jorge Carapinha, Joao Paulo Barraca, Rui L Aguiar, and Susana Sargento. "Toward a telco cloud environment for service functions". In: *IEEE Communications Magazine* 53.2 (2015), pp. 98–106.

[19] Zuo Xiang. *SFC-Ostack: A Simple Research Framework for SFC on OpenStack*. https://github.com/stevelorenz/sfc-ostack. [Online; accessed 2022-02-18]. 2018.

[20] Konstantinos Antonakoglou, Xiao Xu, Eckehard Steinbach, Toktam Mahmoodi, and Mischa Dohler. "Toward haptic communications over the 5G tactile Internet". In: *IEEE Communications Surveys & Tutorials* 20.4 (2018), pp. 3034–3059.

[21] Kwang-Cheng Chen, Tao Zhang, Richard D Gitlin, and Gerhard Fettweis. "Ultra-low latency mobile networking". In: *IEEE Network* 33.2 (2018), pp. 181–187.

[22] Oliver Holland, Eckehard Steinbach, R Venkatesha Prasad, Qian Liu, Zaher Dawy, Adnan Aijaz, Nikolaos Pappas, Kishor Chandra, Vijay S Rao, Sharief Oteafy, et al. "The IEEE 1918.1 "tactile internet" standards working group and its standards". In: *Proceedings of the IEEE* 107.2 (2019), pp. 256–279.

[23] Zhanwei Hou, Changyang She, Yonghui Li, Tony QS Quek, and Branka Vucetic. "Burstiness-aware bandwidth reservation for ultra-reliable and low-latency communications in tactile Internet". In: *IEEE Journal on Selected Areas in Communications* 36.11 (2018), pp. 2401–2410.

[24] Ahmed Nasrallah, Akhilesh S Thyagaturu, Ziyad Alharbi, Cuixiang Wang, Xing Shao, Martin Reisslein, and Hesham ElBakoury. "Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research". In: *IEEE Communications Surveys & Tutorials* 21.1 (2018), pp. 88–145.

[25] Yang Yang and Anthony M Zador. "Differences in sensitivity to neural timing among cortical areas". In: *Journal of Neuroscience* 32.43 (2012), pp. 15142–15147.

[26] Xian-Ming Zhang, Qing-Long Han, and Xinghuo Yu. "Survey on recent advances in networked control systems". In: *IEEE Transactions on industrial informatics* 12.5 (2015), pp. 1740–1752.

[27] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, and Haiyong Xie. "A survey on software-defined networking". In: *IEEE Communications Surveys & Tutorials* 17.1 (2014), pp. 27–51.

[28] Frank Fitzek, Gerrit Schulte, and Martin Reisslein. "System architecture for billing of multi-player games in a wireless environment using GSM/UMTS and WLAN services". In: *Proceedings of the 1st workshop on Network and system support for games*. 2002, pp. 58–64.

[29] Noa Zilberman, Matthew Grosvenor, Diana Andreea Popescu, Neelakandan Manihatty-Bojan, Gianni Antichi, Marcin Wójcik, and Andrew W Moore. "Where has my time gone?" In: *International Conference on Passive and Active network measurement*. Springer. 2017, pp. 201–214.

[30] Nathan Hanford, Vishal Ahuja, Matthew K Farrens, Brian Tierney, and Dipak Ghosal. "A survey of end-system optimizations for high-speed networks". In: *ACM Computing Surveys (CSUR)* 51.3 (2018), pp. 1–36.

[31] Paul Emmerich, Daniel Raumer, Sebastian Gallenmüller, Florian Wohlfart, and Georg Carle. "Throughput and latency of virtual switching with open vswitch: A quantitative analysis". In: *Journal of Network and Systems Management* 26.2 (2018), pp. 314–338.

[32] Giuseppe Lettieri, Vincenzo Maffione, and Luigi Rizzo. "A survey of fast packet I/O technologies for network function virtualization". In: *International Conference on High Performance Computing*. Springer. 2017, pp. 579–590.

[33] Linquan Zhang, Shangqi Lai, Chuan Wu, Zongpeng Li, and Chuanxiong Guo. "Virtualized network coding functions on the Internet". In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2017, pp. 129–139.

[34] Michail-Alexandros Kourtis, Michael J McGrath, Georgios Gardikis, Georgios Xilouris, Vincenzo Riccobene, Panagiotis Papadimitriou, Eleni Trouva, Francesco Liberati, Marco Trubian, Josep Batallé, et al. "T-nova: An open-source mano stack for nfv infrastructures". In: *IEEE Transactions on Network and Service Management* 14.3 (2017), pp. 586–602.

[35] Hyame Assem Alameddine, Sanaa Sharafeddine, Samir Sebbah, Sara Ayoubi, and Chadi Assi. "Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing". In: *IEEE Journal on Selected Areas in Communications* 37.3 (2019), pp. 668–682.

[36] Hassan Halabian. "Distributed resource allocation optimization in 5G virtualized networks". In: *IEEE Journal on Selected Areas in Communications* 37.3 (2019), pp. 627–642.

[37] Hassan Hawilo, Manar Jammal, and Abdallah Shami. "Network function virtualization-aware orchestrator for service function chaining placement in the cloud". In: *IEEE Journal on Selected Areas in Communications* 37.3 (2019), pp. 643–655.

[38] Long Qu, Chadi Assi, Khaled Shaban, and Maurice J Khabbaz. "A reliability-aware network service chain provisioning with delay guarantees in NFV-enabled enterprise datacenter networks". In: *IEEE Transactions on Network and Service Management* 14.3 (2017), pp. 554–568.

[39] Satyam Agarwal, Francesco Malandrino, Carla Fabiana Chiasserini, and Swades De. "VNF placement and resource allocation for the support of vertical services in 5G networks". In: *IEEE/ACM Transactions on Networking* 27.1 (2019), pp. 433–446.

[40] Ilias Benkacem, Tarik Taleb, Miloud Bagaa, and Hannu Flinck. "Optimal VNFs placement in CDN slicing over multi-cloud environment". In: *IEEE Journal on Selected Areas in Communications* 36.3 (2018), pp. 616–627.

[41] Abdelquoddouss Laghrissi, Tarik Taleb, and Miloud Bagaa. "Conformal mapping for optimal network slice planning based on canonical domains". In: *IEEE Journal on Selected Areas in Communications* 36.3 (2018), pp. 519–528.

[42] Toke Høiland-Jørgensen, Jesper Dangaard Brouer, Daniel Borkmann, John Fastabend, Tom Herbert, David Ahern, and David Miller. "The express data path: Fast programmable packet processing in the operating system kernel". In: *Proceedings of the 14th international conference on emerging networking experiments and technologies*. 2018, pp. 54–66.

[43] Sebastiano Miano, Matteo Bertrone, Fulvio Risso, Massimo Tumolo, and Mauricio Vásquez Bernal. "Creating complex network services with ebpf: Experience and lessons learned". In: *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*. IEEE. 2018, pp. 1–8.

[44] Zaafar Ahmed, Muhammad Hamad Alizai, and Affan A Syed. "Inkev: In-kernel distributed network virtualization for dcn". In: *ACM SIGCOMM Computer Communication Review* 46.3 (2018), pp. 1–6.

[45] Sebastian Gallenmüller, Dominik Scholz, Florian Wohlfart, Quirin Scheitle, Paul Emmerich, and Georg Carle. "High-performance packet processing and measurements". In: *2018 10th International Conference on Communication Systems & Networks (COMSNETS)*. IEEE. 2018, pp. 1–8.

[46] Sebastian Gallenmüller, Paul Emmerich, Florian Wohlfart, Daniel Raumer, and Georg Carle. "Comparison of frameworks for high-performance packet IO". In: *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE. 2015, pp. 29–38.

[47] Nguyen Van Tu, Kyungchan Ko, and James Won-Ki Hong. "Architecture for building hybrid kernel-user space virtual network functions". In: *2017 13th International Conference on Network and Service Management (CNSM)*. IEEE. 2017, pp. 1–6.

[48] Chuanpeng Li, Chen Ding, and Kai Shen. "Quantifying the cost of context switch". In: *Proceedings of the 2007 workshop on Experimental computer science*. 2007, 2–es.

[49] Morten V Pedersen, Janus Heide, and Frank HP Fitzek. "Kodo: An open and research oriented network coding library". In: *International Conference on Research in Networking*. Springer. 2011, pp. 145–152.

[50] Danilo Cerović, Valentin Del Piccolo, Ahmed Amamou, Kamel Haddadou, and Guy Pujolle. "Fast packet processing: A survey". In: *IEEE Communications Surveys & Tutorials* 20.4 (2018), pp. 3645–3676.

[51] David Barach, Leonardo Linguaglossa, Damjan Marion, Pierre Pfister, Salvatore Pontarelli, and Dario Rossi. "High-speed software data plane via vectorized packet processing". In: *IEEE Communications Magazine* 56.12 (2018), pp. 97–103.

[52] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. "The Design and Implementation of Open {vSwitch}". In: *12th USENIX symposium on networked systems design and implementation (NSDI 15)*. 2015, pp. 117–130.

[53] Michail-Alexandros Kourtis, Georgios Xilouris, Vincenzo Riccobene, Michael J McGrath, Giuseppe Petralia, Harilaos Koumaras, Georgios Gardikis, and Fidel Liberal. "Enhancing VNF performance by exploiting SR-IOV and DPDK packet processing acceleration". In: *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. IEEE. 2015, pp. 74–78.

[54] *BPF Compiler Collection (BCC) - Tools for BPF-based Linux IO analysis, networking, monitoring, and more*. `https://github.com/iovisor/bcc`. [Online; accessed 2022-02-18].

[55] *Libxdp - Library for attaching XDP programs and using AF_XDP sockets*. https://github.com/xdp-project/xdp-tools/tree/master/lib/libxdp. [Online; accessed 2022-02-18].

[56] Jagmohan Chauhan, Dwight Makaroff, and Anthony Arkles. "Is doing clock synchronization in a VM a good idea?" In: *Proc. IEEE Int. Perform. Comput. Commun. Conf.* 2010, pp. 1–2.

[57] *iPerf - The ultimate speed test tool for TCP, UDP and SCTP*. https://iperf.fr/. [Online; accessed 2022-02-18].

[58] Long Qu, Chadi Assi, and Khaled Shaban. "Delay-aware scheduling and resource optimization with network function virtualization". In: *IEEE Transactions on communications* 64.9 (2016), pp. 3746–3758.

[59] Christian Sieber, Raphael Durner, Maximilian Ehm, Wolfgang Kellerer, and Puneet Sharma. "Towards optimal adaptation of nfv packet processing to modern cpu memory architectures". In: *Proceedings of the 2nd Workshop on Cloud-Assisted Networking*. 2017, pp. 7–12.

[60] Wolfgang Hahn, Borislava Gajic, Florian Wohlfart, Daniel Raumer, Paul Emmerich, Sebastian Gallenmüller, and Georg Carle. "Feasibility of compound chained network functions for flexible packet processing". In: *European Wireless 2017; 23th European Wireless Conference*. VDE. 2017, pp. 1–6.

[61] Alfred Bratterud, Alf-Andre Walla, Hårek Haugerud, Paal E Engelstad, and Kyrre Begnum. "IncludeOS: A minimal, resource efficient unikernel for cloud services". In: *2015 IEEE 7th international conference on cloud computing technology and science (cloudcom)*. IEEE. 2015, pp. 250–257.

[62] Rudolf Ahlswede, Ning Cai, S-YR Li, and Raymond W Yeung. "Network information flow". In: *IEEE Transactions on information theory* 46.4 (2000), pp. 1204–1216.

[63] Tracey Ho, Muriel Médard, Ralf Koetter, David R Karger, Michelle Effros, Jun Shi, and Ben Leong. "A random linear network coding approach to multicast". In: *IEEE Transactions on information theory* 52.10 (2006), pp. 4413–4430.

[64] Vu Nguyen, Giang T Nguyen, Frank Gabriel, Daniel E Lucani, and Frank HP Fitzek. "Integrating sparsity into Fulcrum codes: Investigating throughput, complexity and overhead". In: *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE. 2018, pp. 1–6.

[65] Frank Gabriel, Simon Wunderlich, Sreekrishna Pandi, Frank HP Fitzek, and Martin Reisslein. "Caterpillar RLNC with feedback (CRLNC-FB): Reducing delay in selective repeat ARQ through coding". In: *IEEE Access* 6 (2018), pp. 44787–44802.

[66] Daniel E Lucani, Morten Videbæk Pedersen, Diego Ruano, Chres W Sørensen, Frank HP Fitzek, Janus Heide, Olav Geil, Vu Nguyen, and Martin Reisslein. "Fulcrum: Flexible network coding for heterogeneous devices". In: *Ieee Access* 6 (2018), pp. 77890–77910.

[67] Alexandros G Dimakis, P Brighten Godfrey, Yunnan Wu, Martin J Wainwright, and Kannan Ramchandran. "Network coding for distributed storage systems". In: *IEEE transactions on information theory* 56.9 (2010), pp. 4539–4551.

[68] Joao Barros, Rui A Costa, Daniele Munaretto, and Joerg Widmer. "Effective delay control in online network coding". In: *IEEE INFOCOM 2009*. IEEE. 2009, pp. 208–216.

[69] Sreekrishna Pandi, Frank Gabriel, Juan A Cabrera, Simon Wunderlich, Martin Reisslein, and Frank HP Fitzek. "PACE: Redundancy engineering in RLNC for low-latency communication". In: *IEEE Access* 5 (2017), pp. 20477–20493.

[70] Mohammad Karzand and Douglas J Leith. "Low delay random linear coding over a stream". In: *2014 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE. 2014, pp. 521–528.

[71] Simon Wunderlich, Frank Gabriel, Sreekrishna Pandi, Frank HP Fitzek, and Martin Reisslein. "Caterpillar RLNC (CRLNC): A practical finite sliding window RLNC approach". In: *IEEE Access* 5 (2017), pp. 20183–20197.

[72] Wolfgang Kellerer, Patrick Kalmbach, Andreas Blenk, Arsany Basta, Martin Reisslein, and Stefan Schmid. "Adaptable and data-driven softwarized networks: Review, opportunities, and challenges". In: *Proceedings of the IEEE* 107.4 (2019), pp. 711–731.

[73] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. "Measuring {HTTPS} adoption on the web". In: *26th USENIX security symposium (USENIX security 17)*. 2017, pp. 1323–1338.

[74] Thomas Hoeschele, Christoph Dietzel, Daniel Kopp, Frank HP Fitzek, and Martin Reisslein. "Importance of Internet Exchange Point (IXP) infrastructure for 5G: Estimating the impact of 5G use cases". In: *Telecommunications Policy* 45.3 (2021), p. 102091.

[75] Cisco. *Cloud-Native Network Functions (CNFs) White Paper*. `https://www.cisco.com/c/en/us/products/collateral/routers/cloud-native-broadband-router/white-paper-c11-740841.html`. [Online; accessed 2022-02-18]. 2018.

[76] Cornelius Diekmann, Johannes Naab, Andreas Korsten, and Georg Carle. "Agile network access control in the container age". In: *IEEE Transactions on Network and Service Management* 16.1 (2018), pp. 41–55.

[77] Motassem Al-Tarazi and J Morris Chang. "Performance-aware energy saving for data center networks". In: *IEEE Transactions on Network and Service Management* 16.1 (2019), pp. 206–219.

[78] Ruben Milocco, Pascale Minet, Eric Renault, and Selma Boumerdassi. "Evaluating the upper bound of energy cost saving by proactive data center management". In: *IEEE Transactions on Network and Service Management* 17.3 (2020), pp. 1527–1541.

[79] Thang Le Duc, Rafael Garcia Leiva, Paolo Casari, and Per-Olov Ö stberg. "Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey". In: *ACM Computing Surveys (CSUR)* 52.5 (2019), pp. 1–39.

[80] Amit Sheoran, Sonia Fahmy, Lianjie Cao, and Puneet Sharma. "AI-Driven Provisioning in the 5G Core". In: *IEEE Internet Computing* 25.2 (2021), pp. 18–25.

[81] Lianjie Cao, Puneet Sharma, Sonia Fahmy, and Vinay Saxena. "ENVI: Elastic resource flexing for Network function Virtualization". In: *9th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 17)*. 2017.

[82] Hui Yu, Jiahai Yang, and Carol Fung. "Fine-grained cloud resource provisioning for virtual network function". In: *IEEE Transactions on Network and Service Management* 17.3 (2020), pp. 1363–1376.

[83] Xuesong Li, Wenxue Cheng, Tong Zhang, Fengyuan Ren, and Bailong Yang. "Towards power efficient high performance packet I/O". In: *IEEE Transactions on Parallel and Distributed Systems* 31.4 (2019), pp. 981–996.

[84] Jons-Tobias Wamhoff, Stephan Diestelhorst, Christof Fetzer, Patrick Marlier, Pascal Felber, and Dave Dice. "The TURBO diaries: Application-controlled frequency scaling explained". In: *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. 2014, pp. 193–204.

[85] Yan Liu. "Optimizing PAPI for Low-Overhead Counter Measurement". PhD thesis. University of Maine, 2017.

[86] Sangjin Han, Keon Jang, Aurojit Panda, Shoumik Palkar, Dongsu Han, and Sylvia Ratnasamy. "SoftNIC: A software NIC to augment hardware". In: *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-155* (2015).

[87] Tom Barbette, Cyril Soldani, and Laurent Mathy. "Fast userspace packet processing". In: *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE. 2015, pp. 5–16.

[88] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. "Moongen: A scriptable high-speed packet generator". In: *Proceedings of the 2015 Internet Measurement Conference*. 2015, pp. 275–287.

[89] Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, and Scott Shenker. "NetBricks: Taking the V out of NFV". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 203–216.

[90] Leonardo Linguaglossa, Dario Rossi, Salvatore Pontarelli, Dave Barach, Damjan Marjon, and Pierre Pfister. "High-speed data plane and network functions virtualization by vectorizing packet processing". In: *Computer Networks* 149 (2019), pp. 187–199.

[91] Vinicius Fulber Garcia, Leonardo da C Marcuzzo, Alexandre Huff, Lucas Bondan, Jeferson C Nobre, Alberto Schaeffer-Filho, Carlos RP dos Santos, Lisandro Z Granville, and Elias P Duarte. "On the design of a flexible architecture for virtualized network function platforms". In: *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2019, pp. 1–6.

[92] Hristo Georgiev Trifonov. “Traffic-aware adaptive polling mechanism for high performance packet processing”. In: (2017).

[93] Shihabur Rahman Chowdhury, Mohammad A Salahuddin, Noura Limam, and Raouf Boutaba. “Re-architecting NFV ecosystem with microservices: State of the art and research challenges”. In: *IEEE Network* 33.3 (2019), pp. 168–176.

[94] Shihabur Rahman Chowdhury, Haibo Bian, Tim Bai, Raouf Boutaba, et al. “A disaggregated packet processing architecture for network function virtualization”. In: *IEEE Journal on Selected Areas in Communications* 38.6 (2020), pp. 1075–1088.

[95] Prateek Shantharama, Akhilesh S Thyagaturu, and Martin Reisslein. “Hardware-accelerated platforms and infrastructures for network functions: A survey of enabling technologies and research studies”. In: *IEEE Access* 8 (2020), pp. 132021–132085.

[96] Adel Bouridah, Ilhem Fajjari, Nadiib Aitsaadi, and Hacene Belhadef. “Optimized Scalable SFC Traffic Steering Scheme for Cloud Native based Applications”. In: *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE. 2021, pp. 1–6.

[97] Maciej Gawel and Krzysztof Zielinski. “Analysis and Evaluation of Kubernetes based NFV management and orchestration”. In: *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE. 2019, pp. 511–513.

[98] Leila Abdollahi Vayghan, Mohamed Aymen Saied, Maria Toeroe, and Ferhat Khendek. “Deploying microservice based applications with kubernetes: Experiments and lessons learned”. In: *2018 IEEE 11th international conference on cloud computing (CLOUD)*. IEEE. 2018, pp. 970–973.

[99] Ahmad Faisal Sani and Mukhammad Andri Setiawan. “DNS tunneling Detection Using Elasticsearch”. In: *IOP Conference Series: Materials Science and Engineering*. Vol. 722. 1. IOP Publishing. 2020, p. 012064.

[100] Tianzhu Zhang, Leonardo Linguaglossa, Massimo Gallo, Paolo Giaccone, and Dario Rossi. “FlowMon-DPDK: Parsimonious per-flow software monitoring at line rate”. In: *2018 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE. 2018, pp. 1–8.

[101] Tianzhu Zhang, Leonardo Linguaglossa, Massimo Gallo, Paolo Giaccone, and Dario Rossi. “FloWatcher-DPDK: Lightweight line-rate flow-level monitoring in software”. In: *IEEE Transactions on Network and Service Management* 16.3 (2019), pp. 1143–1156.

[102] *Packetbeat documentation: Configure traffic capturing options*. `https://www.elastic.co/guide/en/beats/packetbeat/current/configuration-interfaces.html`. [Online; accessed 2022-02-18]. 2020.

[103] Jesper Dangaard Brouer and Toke Høiland-Jørgensen. “XDP: challenges and future work”. In: *Proc. Linux Plumbers Conference*. 2018.

[104] Nikolai Pitaev, Matthias Falkner, Aris Leivadeas, and Ioannis Lambadaris. "Characterizing the performance of concurrent virtualized network functions with OVS-DPDK, FD. IO VPP and SR-IOV". In: *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*. 2018, pp. 285–292.

[105] *Mizar project documentation*. https://mizar.readthedocs.io/en/latest/. [Online; accessed 2022-02-18]. 2020.

[106] Daniel Raumer, Florian Wohlfart, Dominik Scholz, Paul Emmerich, and Georg Carle. "Performance exploration of software-based packet processing systems". In: *Leistungs-, Zuverlässigkeits-und Verlä sslichkeitsbewertung von Kommunikationsnetzen und verteilten Systemen* 8 (2015).

[107] Danish Sattar and Ashraf Matrawy. "An empirical model of packet processing delay of the Open vSwitch". In: *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. IEEE. 2017, pp. 1–6.

[108] Muhammad Faisal Iqbal, Muhammad Zahid, Durdana Habib, and Lizy Kurian John. "Efficient prediction of network traffic for real-time applications". In: *Journal of Computer Networks and Communications* 2019 (2019).

[109] SO Abdulsalam, Kayode S Adewole, and RG Jimoh. "Stock trend prediction using regression analysis–a data mining approach". In: (2011).

[110] Ethan Blanton, Dr. Vern Paxson, and Mark Allman. *TCP Congestion Control*. RFC 5681. Sept. 2009. DOI: 10.17487/RFC5681. URL: https://www.rfc-editor.org/info/rfc5681.

[111] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. "The macroscopic behavior of the TCP congestion avoidance algorithm". In: *ACM SIGCOMM Computer Communication Review* 27.3 (1997), pp. 67–82.

[112] *TRex: Realistic Traffic Generator*. https://trex-tgn.cisco.com/. [Online; accessed 2022-02-18]. 2022.

[113] Len Brown. *turbostat-Report processor frequency and idle statistics*. 2019.

[114] Vincent M Weaver, Matt Johnson, Kiran Kasichayanula, James Ralph, Piotr Luszczek, Dan Terpstra, and Shirley Moore. "Measuring energy and power with PAPI". In: *2012 41st international conference on parallel processing workshops*. IEEE. 2012, pp. 262–268.

[115] A Morton. "Imix genome: Specification of variable packet sizes for additional testing". In: *AT&T Labs, July* (2013).

[116] *veth - Virtual Ethernet Device*. https://man7.org/linux/man-pages/man4/veth.4.html. [Online; accessed 2022-02-18]. 2022.

[117] Colin Ian King. "Stress-ng". In: *URL: http://kernel. ubuntu. com/git/cking/stressng. git/(visited on 28/03/2018)* (2017).

[118] Zuo Xiang, Sreekrishna Pandi, Juan Cabrera, Fabrizio Granelli, Patrick Seeling, and Frank HP Fitzek. "An open source testbed for virtualized communication networks". In: *IEEE Communications Magazine* 59.2 (2021), pp. 77–83.

[119] *ComNetsEmu: A virtual emulator/testbed designed for the book: Computing in Communication Networks: From Theory to Practice*. `https://git.comnets.net/public-repo/comnetsemu`. [Online; accessed 2022-02-18].

[120] Albert Banchs, David M Gutierrez-Estevez, Manuel Fuentes, Mauro Boldi, and Silvia Provvedi. "A 5G mobile network architecture to support vertical industries". In: *IEEE Communications Magazine* 57.12 (2019), pp. 38–44.

[121] Pang-Wei Tsai, Francesco Piccialli, Chun-Wei Tsai, Mon-Yen Luo, and Chu-Sing Yang. "Control frameworks in network emulation testbeds: A survey". In: *Journal of computational science* 22 (2017), pp. 148–161.

[122] Bob Lantz and Brian O'Connor. "A mininet-based virtual testbed for distributed SDN development". In: *ACM SIGCOMM Computer Communication Review* 45.4 (2015), pp. 365–366.

[123] Jürgen Cito, Gerald Schermann, John Erik Wittern, Philipp Leitner, Sali Zumberi, and Harald C Gall. "An empirical analysis of the docker container ecosystem on github". In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE. 2017, pp. 323–333.

[124] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. "OpenFlow: enabling innovation in campus networks". In: *ACM SIGCOMM computer communication review* 38.2 (2008), pp. 69–74.

[125] FUJITA Tomonori. "Introduction to ryu sdn framework". In: *Open Networking Summit* (2013), pp. 1–14.

[126] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, et al. "ONOS: towards an open, distributed SDN OS". In: *Proceedings of the third workshop on Hot topics in software defined networking*. 2014, pp. 1–6.

[127] Jan Medved, Robert Varga, Anton Tkacik, and Ken Gray. "Opendaylight: Towards a model-driven sdn controller architecture". In: *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*. IEEE. 2014, pp. 1–6.

[128] Manuel Peuster, Johannes Kampmeyer, and Holger Karl. "Containernet 2.0: A rapid prototyping platform for hybrid service function chains". In: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE. 2018, pp. 335–337.

[129] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. "Mobile edge computing—A key technology towards 5G". In: *ETSI white paper* 11.11 (2015), pp. 1–16.

[130] Robert Pepper. *Cisco visual networking index (VNI) global mobile data traffic forecast update*. Tech. rep. Cisco, Tech. Rep., Feb. 2013. Accessed: Jul. 10, 2019.[Online]. Available …, 2013.

[131] Jinsoo Kim and Jeongho Cho. "Exploring a multimodal mixture-of-YOLOs framework for advanced real-time object detection". In: *Applied Sciences* 10.2 (2020), p. 612.

[132] Peter Wei, Haocong Shi, Jiaying Yang, Jingyi Qian, Yinan Ji, and Xiaofan Jiang. "City-scale vehicle tracking and traffic flow estimation using low frame-rate traffic cameras". In: *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*. 2019, pp. 602–610.

[133] Alex Dominguez-Sanchez, Miguel Cazorla, and Sergio Orts-Escolano. "Pedestrian movement direction recognition using convolutional neural networks". In: *IEEE transactions on intelligent transportation systems* 18.12 (2017), pp. 3540–3548.

[134] Jonathan Hui. "Real-time object detection with yolo, yolov2 and now yolov3". In: *Available online: medium. com/@ jonathan_hui /real-time-object-detection-with-YOLO-YOLOv2-28b1b93e2088 (accessed on 24 February 2019)* (2018).

[135] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. "Efficient processing of deep neural networks: A tutorial and survey". In: *Proceedings of the IEEE* 105.12 (2017), pp. 2295–2329.

[136] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[137] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems* 28 (2015).

[138] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[139] Li Lin, Xiaofei Liao, Hai Jin, and Peng Li. "Computation offloading toward edge computing". In: *Proceedings of the IEEE* 107.8 (2019), pp. 1584–1607.

[140] Jim Gettys. "Bufferbloat: Dark buffers in the internet". In: *IEEE Internet Computing* 15.3 (2011), pp. 96–96.

[141] Justus Rischke, Peter Sossalla, Sebastian Itting, Frank HP Fitzek, and Martin Reisslein. "5G Campus Networks: A First Measurement Study". In: *IEEE Access* 9 (2021), pp. 121786–121803.

[142] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. "P4: Programming protocol-independent packet processors". In: *ACM SIGCOMM Computer Communication Review* 44.3 (2014), pp. 87–95.

[143] Ike Kunze, Klaus Wehrle, Dirk Trossen, Marie-Jose Montpetit, Xavier de Foy, David Griffin, and Miguel Rio. *Use Cases for In-Network Computing*. Internet-Draft draft-irtf-coinrg-use-cases-02. Work in Progress. Internet Engineering Task Force, Mar. 2022. 54 pp. URL: https://datatracker.ietf.org/doc/html/draft-irtf-coinrg-use-cases-02.

[144] Davide Sanvito, Giuseppe Siracusano, and Roberto Bifulco. "Can the network be the AI accelerator?" In: *Proceedings of the 2018 Morning Workshop on In-Network Computing*. 2018, pp. 20–25.

[145] René Glebke, Johannes Krude, Ike Kunze, Jan Rüth, Felix Senger, and Klaus Wehrle. "Towards executing computer vision functionality on programmable network devices". In: *Proceedings of the 1st ACM CoNEXT Workshop on Emerging in-Network Computing Paradigms*. 2019, pp. 15–20.

[146] Huanzhuo Wu, Zuo Xiang, Giang T Nguyen, Yunbin Shen, and Frank HP Fitzek. "Computing Meets Network: COIN-Aware Offloading for Data-Intensive Blind Source Separation". In: *IEEE Network* 35.5 (2021), pp. 21–27.

[147] *Computing in the Network Research Group (coinrg)*. https://datatracker.ietf.org/rg/coinrg/about/. Accessed: 2022-02-16.

[148] Ike Kunze, Klaus Wehrle, and Dirk Trossen. *Transport Protocol Issues of In-Network Computing Systems*. Internet-Draft draft-kunze-coinrg-transport-issues-05. Work in Progress. Internet Engineering Task Force, Oct. 2021. 22 pp. URL: https://datatracker.ietf.org/doc/html/draft-kunze-coinrg-transport-issues-05.

[149] Brent E Stephens, Darius Grassi, Hamidreza Almasi, Tao Ji, Balajee Vamanan, and Aditya Akella. "TCP is Harmful to In-Network Computing: Designing a Message Transport Protocol (MTP)". In: *Proc. Twentieth ACM Workshop on Hot Topics in Networks*. 2021, pp. 61–68.

[150] David D Clark and David L Tennenhouse. "Architectural considerations for a new generation of protocols". In: *ACM SIGCOMM Computer Communication Review* 20.4 (1990), pp. 200–208.

[151] Zuo Xiang and Renbing Zhang. *COIN-DL*. https://github.com/stevelorenz/build-vnf/tree/master/coin_dl. 2022.

[152] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. "Microsoft COCO: Common objects in context". In: *Proc. European Conference on Computer Vision*. Springer, Cham, Switzerland. 2014, pp. 740–755.