An eBPF-XDP hardware-based network slicing architecture for future 6G front- to back-haul networks

Pablo Salva-Garcia, Ruben Ricart-Sanchez, Jose M. Alcaraz-Calero, Qi Wang, Octavio Herrera-Ruiz

***Abstract*—The heterogeneous requirements imposed by different vertical businesses have motivated a networking paradigm shift in the next generation of mobile networks (beyond 5G and 6G), leading to critical operation competitiveness of improved productivity, performance and efficiency. Furthermore, with the global digital revolution, such as Industry 4.0, and a connected world, network virtualisation together with high reliability and high performance communications have become crucial elements for mobile network operators. To minimise the negative effects that could affect critical services, network slicing is widely recognised as a key technology with the objective of meeting the Service-Level Agreements (SLAs) and Key Performance Indicators (KPIs) in future 6G networks. In this context, it is essential to introduce a programmable data plane able to enforce flexible Quality of Service (QoS) commitments, while providing high-performance packet processing and real-time monitoring capabilities. To this end, this paper is focused on designing, prototyping and evaluating a novel framework that leverages a set of hardware-based technologies including eXpress Data Path (XDP), extended Barkeley Packet Filter (eBPF) and Smart Network Interface Cards (SmartNICs) to offload network functionality with the objective of providing high-performance pre-6G front-, mid- and back-haul network communications and thus, decreasing the overhead incurs by the Linux Kernel. The proposed solution is implemented based on bypassing the Linux Kernel and accelerating the communication, while providing network slice control and real-time monitoring capabilities. The main aim of this framework is to ensure network communications in forthcoming 6G infrastructures by guaranteeing 6G KPIs and avoiding system overload. The empirical validation of this solution for Industry 4.0 services as an example use case demonstrates key performance improvements in terms of packet processing as high as about 25Gbps, 20M packet per second, 0% packet loss, 0.1ms of latency and less than 10% load on the CPUs.**

***Index Terms*—6G, Programmable Data Plane, eBPF, XDP, Network Slicing.**

I. Introduction

THE beyond 5th Generation (B5G) and 6th Generation (6G) mobile networks are expected to support advanced capabilities that exceed those of current mobile technologies, such as 4G and 5G, in order to accommodate highly demanding applications. [1]. Given the considerable challenges and diverse requirements that these applications will place on Next-Generation Networks (NGNs) – including performance, reliability, latency, connectivity, or throughput – It is essential to equip future mobile network infrastructures with the necessary capabilities to process network packets at high speeds and manage Quality of Service (QoS) in a flexible manner to ensure that specific Service Level Agreements (SLAs) can be met. The importance of preparing next-generation networks for a global digital revolution has been previously acknowledged, and significant progress has been made to address this need. The radio segment has evolved to new technologies, with a shift from downlink-centric to uplink-centric and the use of higher frequency bands from mmWave to THz. Uplink-Centric Broadband Communication (UCBC) provides a 10x increase in uplink bandwidth. These promising advances can benefit industries and manufacturers through their transition to future 6G networks, where the concept of the Internet of Things (IoT) goes beyond the traditional sensors and other technologies exchanging data, and focuses on a novel concept called Industrial Internet of Everything (IIoE), whereby an interconnected network is aggregated in IIoE to facilitate revolutionary network services [2], [3]. While numerous advances have been made in radio segments, it is crucial to similarly evolve wired infrastructure segments. These must match the growing capacity of radio networks and include mechanisms that facilitate QoS management in network services delivery, with an emphasis on front-, mid-, and back-haul network communications. In response to this, the present research adopts Transport Network Slicing, a concept which allows a set of services, each with different needs, to operate concurrently on a single network infrastructure [4]. Network slicing was widely recognised as one of the key opportunities enabled by 5G, and its exploration is projected to continue in future network generations [5], [6]. Network slicing is a method used to separate the existing network into various virtual networks tailored to meet specific use case demands. This partitioning provides network operators with the ability to split their physical infrastructure into several virtual networks (termed as multi-tenancy), where each one is allocated its own dedicated resources and configurations. Through network slicing, operators can achieve greater adaptability, increased efficiency, and improved QoS in managing and delivering network services. The slicing architecture proposed in our study presents a novel framework that enhances network packet processing and QoS management by allocating network traffic among multiple isolated network slices. Every slice functions as an independent network entity that delivers unique network

R. Ricart-Sanchez, P. Salva-Garcia, J.M. Alcaraz-Calero and Q. Wang are with the School of Computing, Engineering and Physical Sciences, University of the West of Scotland, United Kingdom, High St, Paisley PA1 2BE.

O. Herrera-Ruiz is with Netronome/Mira, 3159 Unionville Road, Suite 100, Cranberry, PA 16066, Pensilvania, USA.

Manuscript received July 24, 2023; Accepted October 27, 2023.

services and can be individually managed and scaled to meet to the particular requirements of its associated applications.

However, while promising for enabling differentiated services, network slicing also presents critical challenges in the data plane that may result in low-performance scenarios if the underlying infrastructure is not appropriately upgraded and equipped with adequate packet-processing systems. In virtualised network environments that support multi-tenancy, network services are decoupled from the underlying infrastructure by encapsulating one packet within another packet (Overlay networks), increasing the operational complexity of network traffic processing and classification tasks. The high system resource requirements for processing large amounts of data in high-speed transmissions can lead to bottlenecks and negatively impact the overall performance of the entire mobile network. Therefore, there is a pressing need to address these challenges and develop effective solutions to ensure reliable and efficient network slicing in virtualised network environments. As a response to this challenge, two key concepts have emerged. The first concept is the Data Plane Programmability (DPP), which allows network administrators to fine-tune the behavior of network devices, such as switches and routers, to better suit the needs of specific applications and workloads. By implementing custom logic in the data plane, network operators can achieve faster and more efficient packet processing, as well as more granular control over network traffic. The programmable data plane can be implemented using various technologies, such as field-programmable gate arrays (FPGAs), Smart Network Interface Cards (SmartNICs), or software-based solutions. However, it requires specialized skills and expertise to design and implement custom logic for the data plane, and can also introduce additional complexity to network operations. The second concept is network hardware offloading, which has emerged to overcome the negative impact of high workload on the host's CPUs by delegating some of the network-related tasks from a host's software to specialized hardware components. Network hardware offloading frees up CPU cycles that would otherwise be spent on network-related tasks. This can significantly reduce CPU utilization, resulting in improved system performance and scalability. The aim of this research is to develop a solution for efficient packet processing and classification, which will accelerate network slicing in future 6G front-to-back-haul communications. The primary objective is to meet the demanding Quality of Service (QoS) requirements imposed by the future 6G use cases. However, there are various challenges that need to be taken into account to achieve this goal.

A. Support for encapsulated network traffic

With regards network hardware offloading, some Network Interface Cards (NICs) are equipped with Receive Side Scaling (RSS) technology that allows the NIC to distribute incoming network traffic among multiple queues and their associated processing cores in a system. The NIC generates a hash identifier for each incoming network packet. The hash is calculated based on several packet header fields, such as the source and destination IP addresses, source and destination ports, and protocol type. The NIC then uses this hash value to assign the packet to a specific receive queue, which is associated with a particular CPU core. By spreading the workload across multiple cores, RSS can help prevent performance bottlenecks and improve the overall efficiency of a system's network processing capabilities. In the context of this research work, RSS can be also used to forward network traffic related to a specific services or applications to dedicated queues to achieve network slicing. However, the default RSS algorithms used in many network interface cards (NICs) have limitations. These algorithms can only examine a few header fields in each packet to generate a hash identifier, and are unable to perform deep packet inspection. In virtualized multi-tenant scenarios, such as those found in B5G and future 6G networks, traffic can be deeply encapsulated, making it necessary to explore inner protocols to differentiate among different network operators or service applications. This can cause all incoming network packets to be considered as a single flow by the NIC (same hash identifier), resulting in packets not being distributed across different receive queues and CPUs, which can lead to bottlenecks and CPU starvation. Thus, it is crucial to establish a mechanism for programming network cards with algorithms capable of inspecting, classifying, and processing high-depth encapsulated network traffic in order to effectively utilize multi-queuing network cards. This research utilizes the extended Berkeley Packet Filter (eBPF) to create ad-hoc network traffic filtering programs with encapsulation support, which can be offloaded to the Agilio CX SmartNIC using the eXpress Data Path (XDP) architecture.

B. Kernel Network Stack Limitations

While network hardware offload techniques can help alleviate some of the processing load on the CPU and improve overall network performance, the Linux network subsystem is still responsible for forwarding the packet to its final destination. When a packet arrives at the kernel network stack, it goes through several processing stages, including protocol processing and socket buffering. Here, a potential bottleneck can occur in the socket buffering stage, where incoming packets are stored in buffers before being processed by the application. If the buffer sizes are not properly configured, or if the application is not able to process packets fast enough, this can lead to packet drops and reduced network performance. This behaviour can be observed in our previous work [7] where authors had to overcome such limitation by forwarding packets to other servers for processing since the system's network stack was not able to cope with high-rate packet processing. In this research, we collaborated to develop, integrate, and evaluate an XDP-based kernel bypass technique (AF_XDP) for the Netronome Flow Processor (NFP) driver. This approach enables network packets already processed at the hardware level to be forwarded by the driver to their final destination without incurring processing penalties from the kernel network subsystem. While there are other smartNICs that support kernel bypass at the driver level, this is the first time, to the best of our knowledge, that both XDP-based techniques, hardware offload, and driver AF_XDP are combined into

the same smartNIC architecture to achieve high-performance packet processing and transport network slicing capabilities.

C. Next-generation Networks compatibility

As previously stated, programming the data plane is a complex task that requires specialized skills from network administrators, who would be required to create custom configuration programs with specific capabilities for each new application that is added to the system. This process can cause delays in the network operations. In contrast, NGNs are designed to provide dynamic resource allocation and on-demand service delivery through advanced technologies such as Software Defined Networking (SDN) or Network Function Virtualisation (NFV). As such, to align with the current approaches of NGNs, it is necessary to develop a solution that supports flexibility and on-demand capabilities. This research proposes a user-friendly DPP network application that simplifies the programming of the data plane. The application provides both control and monitoring capabilities and can be easily deployed on-demand. Its flexible configuration can be adjusted to meet the needs of various use cases. By hiding the complexity of programming the data plane, this application simplifies the network administration process, making it more accessible to users with different levels of expertise.

D. Flexibility to Accommodate Diverse Scenarios

This research proposes a solution that is compatible with current 5G technologies, but future 6G use cases are expected to demand even higher Key Performance Indicators (KPIs) due to their need for ultra-low latency, high reliability, and massive connectivity. Saad et al. [1] emphasise that 6G networks will converge technological trends driven by underlying services, necessitating enhanced requirements and KPIs compared to 5G and beyond 5G networks. These use cases include real-time and mission-critical applications like autonomous driving, smart factories, and remote surgeries, which require large amounts of data and complex network topologies, posing significant challenges for meeting KPIs effectively. To ensure our solution can handle overload situations in future 6G front-to back-haul networks, we have empirically validated it with pre-6G use cases, particularly those related to Industry 4.0 (I4.0) services.

This paper demonstrates how combining eBPF-XDP with AF_XDP and eBPF-XDP hardware offloading technologies, all of which are supported by the Agilio CX SmartNIC, can substantially enhance network performance in NGNs architectures while simultaneously enabling the selective customization and classification of network traffic in a flexible manner. To be concrete, this work leverages these technologies to accelerate the wired network segments inside the cloud service provider with the aim of accelerating network communications in the forthcoming 6G networks, maximising the resources of the system without overloading the CPUs to ensure an excellent optimisation of network processing tasks. This work provides a solution that meets the demanding requirement of I4.0 in future 6G networks, and presents several use cases where this solution could be successfully applied to. To this end, the solution proposed has been validated by performing an intensive empirical validation, which demonstrates the viability of this work and also fulfils demanding KPIs expected in future 6G networks.

The rest of the paper is organised in the following sections. Section II presents an exhaustive analyses of the current literature review of different software and hardware network data path technologies. Section III attempts to define the key components of a pre-6G network in a I4.0 scenario. Section IV outlines several novel 6G networks use cases. Section V depicts a detailed representation of the framework architecture created to fulfill the ambitious requirements of the next-generation of networks. Section VI presents a holistic view of the testbed implemented and also an empirical validation of the solution to illustrate the findings. Finally, Section VII summarises the present work and outlines the future research work.

II. Literature Review

In this section, a comprehensive review of the relevant literature is presented, focusing on packet processing techniques. Specifically, and aligned with the scope of this research work, this review categorizes these technologies and implementations into two main groups: kernel bypass and hardware offloading technologies, providing an in-depth analysis of each.

A. Kernel Bypass Technologies

The interface between the kernel of the system and the user-space applications are usually the main sources of performance bottlenecks. Vanilla Linux can only efficiently handle around 1Mpps, which is far from the expected 6G KPIs, where minimum throughput required is 10Mpps [8]. Kernel bypass technologies aim to tackle this by avoiding the use of expensive context switches introduced by the kernel network stack. One of the most popular kernel bypass solutions is Data Plane Development Kit (DPDK) [9] that consists of a set of libraries to accelerate packet processing workloads and available on a wide range of CPU architectures. In this solution, a user-space application takes control of the networking hardware while the Operating System (OS) is bypassed. Snabbswitch [10] is a networking framework written in Lua language and mainly indented for writing L2 applications. This framework is very similar to DPDK since both are full framework and rely on Universal Input Output (UIO). Netmap [11] is not implemented using UIO techniques, but using a couple of kernel modules. However, it is not integrated with networking hardware and it requires drivers customisation. PACKET_MMAP [12] is not strictly a kernel bypass technique, however it is used to efficiently capture network traffic using very limited buffers and one system call improving the efficiency of packet raw transmission. PF_RING [13] is a type of network socket that allows to speed up the packet capture. This tool polls the packets from the NIC using the Linux NAPI allowing interrupt mitigation for networking devices. EF_VI [14] allows to send and receive raw Ethernet frames directly from user-level, reducing the CPU overhead and thus, increasing the performance. However, the user has

to implement the upper-layer protocols by himself. AF_XDP [15] uses XSK sockets to transmit the raw network frames from the NIC to the user application layer and it is optimised for high performance packet processing. In this tool, each socket is associated with a RX and a TX ring. To transmit the packets, these rings use a data buffer in a memory area called a UMEM, which is described as a region of virtual contiguous memory divided into equal-sized frames. Contrary to the previous solutions, AF_XDP is integrated with eBPF-XDP, which is part of the mainline Linux kernel. In addition, eBPF-XDP allows to program the data plane in different contexts such as hardware offloading or network driver, among others. However, to the best of our knowledge, as of now, the driver of Agilio SmartNICs (Netronome), the only SmartNICs that currently support eBPF-XDP hardware offload, did not support AF_XDP. This made it impossible to combine both implementations of hardware offloading and kernel bypass in the driver via AF_XDP. For the purpose of this research work, the driver of this SmartNIC has been extended to integrate the requiredd AF_XDP functionalities.

B. Hardware Offloading Technologies

Programmable hardware provides constant access to memory, which implies that when number of filtering rules are increased there is no extra access memory time overhead associated, thus allowing more specific latency control. Furthermore, genuinely hardware devices have superior processing speed than software data planes, which implies faster transmission rates. In recent years, Xilinx has provided different FPGA and SmartNICs with high-performance capabilities for both trading environments and enterprise data centers, including artificial intelligence, big data, analytics, hyperscale, machine learning, storage, and telco applications [16], [17], [18], [19], [20], [21]. These cards are designed to meet the demanding requirements of modern data centers with internal pipeline Static Random-Access Memory (SRAM) bandwidth of up to 30TB/s and physical network interfaces of up to 100GbE. However, there are two important concerns with these cards, the first one is the high price and the second one is the private property of the Software Development Kit (SDK) and driver. Intel has introduced SmartNIC solutions to provide high-performance and low-latency connectivity for NFV infrastructures, including Virtualised Radio Access Network (vRAN) solutions [22], [23], [24]. But again, due to their high cost and proprietary source code, network administrators and academics in the networking field may not view them as the most attractive option. NetFPGA [25] is an open source, line-rate and flexible platform, which is able to provide network transmissions up to 100Gbps. Furthermore, this card can be programmed using P4 [26] or VHDL [27]. However, as it has been demonstrated in [28], [29], [30], [31] and due to the SUME_RIFFA driver drawbacks, the performance achieved by NetFPGA based solutions does not meet the future 6G networks KPIs. Netronome[32] and Corigine [33] offer an Agilio SmartNIC and software that delivers unmatched efficiency for server networking, including functions such as overlays, telemetry or security. Netronome and Corigine offer

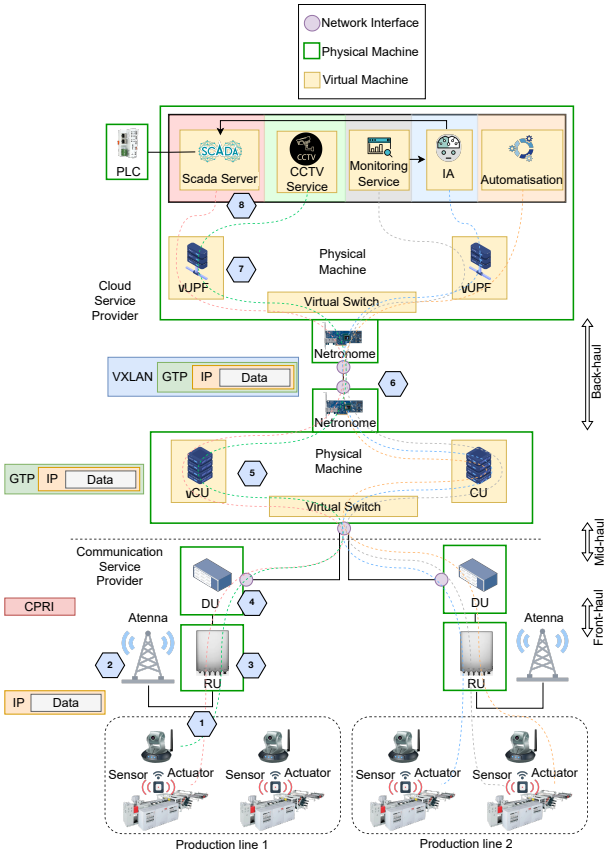


Fig. 1: Multi-tenant beyond 5G and 6G architecture prototype

an SDK that enables programming of their SmartNICs using P4 or constraint C language. These SmartNICs are reasonably in price and their firmware, as well as their NFP drivers, are open-source. In addition, this hardware supports eBPF-XDP hardware offloading, making it highly desirable in terms of hardware programmability and flexibility. Although the NFP driver does not provide genuine AF_XDP support to bypass the Linux kernel, through collaboration with Netronome and Corigine, this capability has been integrated into the NFP driver. As a result, here is proposed a novel approach that combines both hardware offload and kernel bypass (via AF_XDP) techniques for processing complex overlay networks in pre-6G scenarios.

III. Multi-tenant Pre-6G Architecture Overview

Figure 1 shows a multi-tenant beyond 5G and pre-6G architecture for an I4.0 setting. To enhance the readability of the following description, various labels with numbers have been inserted throughout this figure. Label 1 shows that multiple production lines are connected to a communication service provider (CSP). These production lines belong to different virtual mobile network operators. The CSP facilitates wireless communication between sensors and actuators through the use of Multiple Input Multiple Output (MIMO) antennas, Radio

Units (RU), and Decentralised Units (DU), which are labeled in figure 1 as 2, 3, and 4, respectively. RUs are responsible for converting digital data into radio signals that can be transmitted wirelessly through the air to user devices. DU is a physical component allocated close to the RU and runs the Radio Link Control (RLC), MAC and also some parts of the physical layer, and its operation is controlled by the Centralised Unit (CU). The connections from the RU to the DU, from the DU to the virtual CU (vCU), and from the vCU to the core network are referred to as the front-haul, mid-haul and back-haul, respectively.

The core of the architecture is controlled by the cloud service provider, which offers a distributed and virtualised network architecture where physical computers are shared among various virtual network operators. The cloud service provider's segment comprises various components, including virtualized CUs (vCUs), virtualized User Plane Functions (vUPFs), and vertical services. These components are labeled 5 and 7 in Figure 1, respectively. vCUs allow the users communication between the communication service provider and the cloud and therefore, allowing users to access to the core network segment. The vUPFs, apart from forwarding the user data through the network data path, are also in charge of user mobility, authentication, handover management, user registry, and so on. The vUPF forwards the user traffic from the vCU to different vertical services (as seen in label 8), which have different user needs and demanding SLAs which are accomplished by the implementation of advanced network slicing policies. An extended and comprehensive description of the operations in each of the components described is provided in more detail in [34].

The architecture depicted presents a hybrid software/hardware approach, based on separated forwarding devices with the objective of achieving high performance connectivity and advanced network slicing support between hardware and software components of a pre-6G infrastructure. Label 6 in Figure 1 identifies the SmartNIC responsible for forwarding network traffic between the vCUs and the vUPFs. This SmartNIC offloads network data path handling, control, and monitoring functions, creating a slicing-friendly environment with support for pre-6G networks. By doing so, the aim is to optimize network communication performance while preserving the flexibility provided by software components.

The infrastructure presented provides flexible data paths with hardware-based control and monitoring capabilities, and software sockets using kernel-bypass mechanisms, which allow network traffic acceleration from the hardware device to the Network Function Virtualisation (NFV). This paper uses an Agilio CX SmartNIC-based high-performance control and monitoring programmable data path to support 6G advanced network slicing and offload generic storage designed for the cloud service provider with the objective of meeting the demanding KPIs imposed by 6G networks vertical use cases. In the cloud service provider network segment, the network packets are received and sent from and to sensor and actuators, respectively, are encapsulated by the CU using GPRS (General Packet Radio Service) Tunneling Protocol (GTP). This protocol allows network mobility along different locations.

TABLE I: Use cases key performance requirements

Use case	E2E Latency	Reliability	Bandwidth	Packet Size	Slice config
(1)	<0.1ms	10^{-8}	<100Mbps	<1500B	<1s
(2)	<33ms	10^{-6}	<3Gbps	<1500B	<1s
(3)	<0.15ms	10^{-6}	<10Gbps	<300B	<1s
(4)	<0.5ms	10^{-5}	<10Gbps	<132B	<1s
(5)	<0.1ms	10^{-6}	<10Gbps	>200B	<1s

In addition, the virtual switch also creates a second tunnel to provide network traffic isolation among virtual network operators, and therefore to allow logical separation of network resources. While there are various network protocols that can be utilized to create such tunnels, the Virtual Extensible LAN (VXLAN) was specifically chosen for the experiments conducted in this publication. The proposed architecture outlined in this manuscript offers several advantages, such as the ability to use different logical paths using AF_XDP and Agilio CX. Additionally, it provides real-time monitoring metrics while meeting the demanding KPIs required by future 6G networks.

IV. Pre-6G Use cases

This research targets to develop a high-performance and programmable data plane that is capable of dynamic control, monitoring, and network slicing. To validate its feasibility, a set of demanding and novel use cases, specifically those in the context of Industry 4.0, have been selected. Through this validation, KPIs such as data latency, reliability, and bandwidth will be carefully considered. The objective is to ensure that the data plane (from the front-haul to the back-haul) of future 6G networks can support these advanced use cases with the necessary performance and efficiency. These use cases are based on those available in the deliverable 2.1 of the 6G-BRAINS project[35] and their requirements are listed accordingly in Table I.

A. Use Case (1): Offloading of Programmable Logic Controller (PLC) Control Functions to the Edge

Fixed machine controllers like PLCs are aim to be replaced by *soft* controllers running in virtualised environments to provide the level of flexibility of the software and hardware components involved in factories. The communication network between novel virtualised components and machined components is a critical element for the successful traffic transmission in 6G scenarios and the control methods associated. The possibility of offloading the PLC control function to an edge, by running these as containers or virtual machines, provides more flexibility for controlling the production process in industry 4.0. The communication between the virtualised controllers and the machine components requires low latency and high reliability provided by applying network slicing strategies in order to guarantee deterministic communication that should support low industrial application cycle times and very precise synchronicity. Key performance requirements associated to this use case are described in Table I, row (1).

B. Use Case (2): Smart Transportation Vehicles: Localisation and Video Processing Offloading

Automated Guided Vehicles (AGVs) are key contributors for the I4.0. While state of the art factories are still based on

hand labour or semi-automated machines, the factories of the future are investing on novel AI-based AGVs. This approach entails an increment of the overall factory performance, which is associated with a positive outlook for the company's profits. AGVs are great assets in I4.0, taking care of such tasks like driver-less and autonomous transport of both goods and materials from and to production lines. Furthermore, these vehicles can carry robot equipment to facilitate the assembly process by picking and placing parts using a gripper arm thanks to the sensor mounted on the AGV. It means that with low initial and recurring effort, AGVs significantly improves factory efficiency.

This use case can be divided into two important AGV features: *a)* Video capturing using a AGV on-board camera streaming the video to an edge computing node for object detection and decision making; *b)* Indoor and outdoor autonomous navigation based on a real time decisions algorithms.

High-quality video cameras are used to transmit the video from the AGV to the Edge node of the 6G network architecture, where the video is analysed and the decisions are made. This video transmission requires an efficient network architecture behind which ensures an optimal communication with high data rate requirements, consuming up to 3Gbps per camera in a transmission of approximately 130 uncompressed HD frames. With this use case the data rate requirements, the ultra-high precision and the decision making are analysed with respect to the current state of the art technologies in order to meet the KPIs described in table I, row (2).

C. Use Case (3): Advanced Network Slicing

The uses cases previously described mostly represent a single type of traffic and no prioritisation techniques a strictly required apart from isolating the management and control traffic from the application communication. However, it is common to share communication resources of different applications in the same environment. In some I4.0 environments, the requirement between different use cases can dramatically vary and therefore, advance network slicing techniques should be established in order to efficiently guarantee the KPIs of the different use cases. These techniques address the highly dynamic and heterogeneous traffic control requirements and also guarantee real-time QoS for the I4.0.

Advanced network slicing provides flexible and customised E2E guaranteed QoS for divergent customer requirements. It enables the definition of on-demand highly flexible network slices with several granularity levels based on the network data communication. In order to be highly available and effective, the network slice instantiation time should be less than 1 second. In a complex industrial environment, where different use cases co-exist over the same network, intent-based network slicing management facilitates the management tasks and guarantees the corresponding QoS for independent network traffic communications. This allows the compliant of the SLAs in complex and critical I4.0 scenarios using 6G network architectures. The demanding requirement of advanced network slicing are described in table I, row (3).

D. Use Case (4): Animal Tracking in Indoor Farming Scenarios

Smart farming refers to the evolution from conventional farms to IoT-based farming, using modern information and communication technologies to increase both, quality and quantity of the production while human interaction is reduced to optimal levels. This optimisation can be carried out in different aspects; however, this use case is focus on animal tracking in indoor farming scenarios. Farm animals, such as cows, pigs or chickens, wear collar that monitors animal's activity, well-being and health and if anything is that could affect to animal welfare is detected, the farmer and regulatory bodies are immediately notified. Monitoring data is transmitted to the Edge computer node, where an AI agent processes the mobility and health data of the animal in order to guarantee its well-being and health. The KPIs associated with this use case are described in table I, row (4).

E. Use Case (5): Airports Service and Baggage Handling Robots

This use case presents a 6G automated baggage handling system using Automated Guided Vehicle (AGV) robots for carrying passenger luggage between baggage handling conveyor belt locations in airports. Video live streaming of objects in path of service robots and AGVs is used in order to perform object recognition and collision avoidance of objects in their paths. When the passenger arrives at an airport, the luggage is dropped in the check-in line and an AGV takes the luggage that has just been self-checked in and carries it to the right conveyor belt for it to be transported to the destination aircraft producing an automated and sustainable baggage sorting system with minimal amount of baggage handling staff. High reliability and low latency communication is necessary to perform this use case, with the purpose of live streaming communication and real time AI detection and actuation. The demanding communication requirements needed by this use case are described in table I, row (5).

V. Prototyping Platform and Implementation

The following subsections present a detailed description about the proposed framework architecture, the algorithm implemented and the prototyped implementation used to guarantee high-performance and high-reliability communications in 6G front- to back-haul networks.

A. XDP-based Agilio CX SmartNIC reference data-path

This study is based on the Agilio CX SmartNIC platform, which provides a transparent offload SmartNIC with a programmable data path widely integrated with XDP and DPDK via a Netronome Flow Processor (NFP) driver, with a maximum processing rate of 25Gbps. The proposed solution has been created using XDP, eBPF and a custom version of the NFP driver. The Agilio CX SmatNIC has a NFP-4000 processor and also includes 60 programmable flow processing cores, 48 packet processing cores, PCIe Gen3 2x8 interface with RX and TX channels, and 2xSFP28 25GbE physical

interfaces. The SmartNIC uses the Agilio eBPF firmware in order to implement the XDP both offloaded and at the driver level. The solution implemented works with up to 16 hardware queues, directly attached each of them to a XSK socket, which delivers the traffic from the card to the user space. For this implementation, a customized version of the NFP driver provided by Netronome and Coregine was used specifically for the scope of this research, which provides AF_XDP support. AF_XDP is an address family that is optimised for high performance packet processing by creating XSK sockets [15]. The novelty of this implementation comes from different aspects: (1) the offloaded and programmable pre-6G data path implemented with multi-tenancy support, (2) The offloaded high-performance real-time monitoring tool, (3) The offloaded control capabilities to provide advance network slicing and traffic filtering and (4) a custom version of the NFP driver to provide XSK sockets support and therefore, improving the performance of the Linux kernel based implementation. To the best of our understanding, this is the first time that it is approached a solution that combines hardware offloaded capabilities and kernel by-pass techniques using the same network card to enhance high-performance communications in the transport network segment in pre-6G network architectures.

B. Proposed framework architecture

Figure 2 represents the internal architecture of the framework implemented in this work. This framework is divided into three parts: the hardware, using a Agilio CX SmartNIC; the Linux kernel space, where hardware and software interruptions are handle; and the user space, which is the closest part to the end user. In this figure, the dotted red lines represent the control plane and the blue line refers to the data plane.

The SmartNIC contains a custom XDP offloaded firmware and two different eBPF hardware maps: one for control and one for monitoring purposes. These key/value maps are able to allocate over to 2 million rules each and have constant memory access time. In the control map, each rule contains a 32 bits key and a 32 bits value, which is divided into 16 bits to represents the action and 16 bits to indicate the output value of the action if required (E.g., set to a specific rx_queue). Each rule of the monitoring map also contains a 32 bits key and a 64 bits value, divided into 32 bits for the packet length and 32 bits to represent the number of packets matched by this specific rule. The XDP offloaded firmware, represented in grey in the figure, can be divided into three different components, the pre-6G parser, the logic where the hardware maps are check and the output decision is made. The pre-6G parser is in charge of processing the network flows and extracting the required meta-data. This information is used to calculate a hash identifier, which will be used later by the matching module. The matching module compares the hash identifier with those stored in the ePBF control map to determine the appropriate action to take for the network packet. Depending on the comparison result, the matching module can take one of three actions. It can forward the network packet to a specific hardware queue, drop the packet, or pass the packet to the kernel network stack through the default queue.

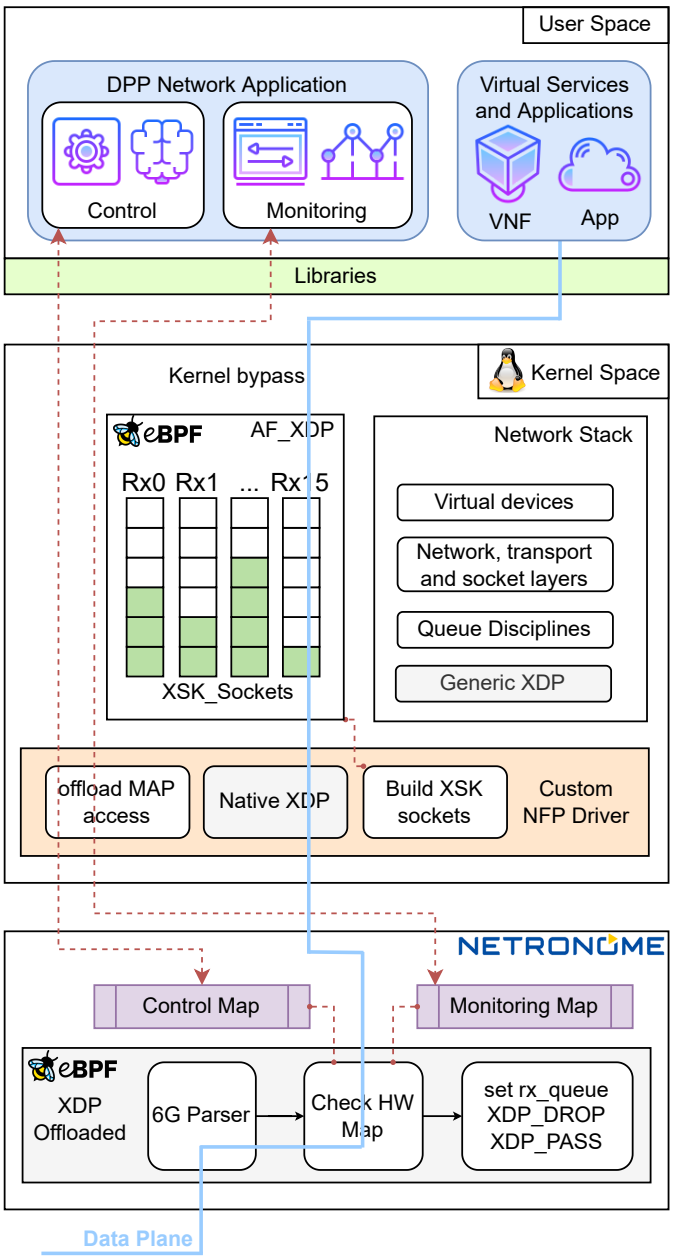


Fig. 2: Proposed framework architecture

Once the network packet reaches the kernel space, it is processed by the network stack, which contains a custom version of the NFP driver that allows for socket forwarding at the driver level. Typically, the public NFP driver uses the network stack to transmit data from the hardware card to the user space. However, after analyzing the results obtained in [7], which used an RSS queue discipline to transmit data through the network stack, it became clear that the performance of this solution did not meet the KPIs of future 6G networks. The bottleneck in the network stack was identified as the cause of this poor performance. To address this issue, the research team proposed a novel and customized version of the NFP driver. This driver leverages AF_XDP and XSK_sockets to bypass the network stack and accelerate communication between the SmartNIC and the user space. The customized

driver implementation supports up to 16 sockets and allows access to the hardware control and monitoring maps. It also provides a flexible data path with Native XDP support. By using this customized driver, the research team was able to achieve improved performance and efficiency compared to a more generic driver, enabling the data plane to meet the demanding requirements of 6G networks.

At the user space level, two main components are present: the Data Plane Programmability (DPP) network application and the virtual services and applications. The DPP network application provides control and monitoring tools, while the virtual services and applications comprise various network services and applications included in the network data plane. Several libraries, such as Libbpf [36] and XSK [37], facilitate communication between these user applications and the kernel space. Control applications are responsible for inserting and removing network rules to and from the hardware control map. In contrast, the monitoring user space application not only inserts and removes rules from the monitoring map but also collects real-time statistics and useful information stored in this map by the hardware pipeline. The monitoring application provides real-time metrics of the network traffic processed by the Agilio CX SmartNIC, such as packets per second processed per socket, the number of packets received/-transmitted, and packet size. This information is vital for network administrators and users alike, providing insight into the performance of the network.

C. Slice definition: End-user and Slice management overview

While this study explores eBPF-XDP based solutions for controlling and programming the data-plane of Transport Network Slices, it is important to note that this is part of a larger architecture design with capabilities to monitor, manage, and orchestrate network slices [35]. In-depth information on the necessary components to support the management and life-cycle of slices can be found in [38]. Moreover, our research in [39] presents the integration process between the Open Network Automation Platform (ONAP) and this slice manager within the European 6G Brains project. This research comprehensively covers the workflows for creating Service Instances and the instantiation of Slices, providing a practical perspective for end-users. For a deeper understanding, we recommend referring to the 6G Brains' project deliverable D5.2 [40]. For the sake of simplicity, we have omitted those steps here and instead, we directly illustrate the message received at the Data Plane Programmability (DPP) application. To illustrate, Listing 1 shows a slice definition message received by the DPP (refer to Figure 2). The message details a specific service, which is defined as a set of network properties outlined in the "*Resources*" section. The message also contains slice configuration attributes such as service priority, Maximum Allowed Bandwidth (MAB), and Minimum Guaranteed Bandwidth (MGB). With this message, the DPP has enough information to process and create the necessary packet metadata structures (see listing 2) and their respective hash identifiers. Similarly, the property *"priority"* is used in the eBPF program (see algorithm 1, line 10) to to indicate the target *rx_queue* for this service.

Listing 1: Intent-based Message (simplified version)

```
{
  "Resources": [{
        "resourceId": "F8A4C949",
        "encapsulationID1": "00000445",
        "encapsulationType1": "vxlan",
        "srcIP": "146.191.50.26",
        "dstPort": "4789",
},{
        "resourceId": "B6E6B2B3",
        "encapsulationID2": "8894D0D4",
        "encapsulationType2": "gtp",
        "srcIP": "10.100.0.19",
        "dstPort": "2152",
}],
  "Intent": {
    "flowAgentName": "XDP-based"
    "actionType": "INSERT",
    "actionName": "CREATE_SLICE",
    "slice_id": "03E8",
    "priority": "1",
    "MAB" : "12000000",
    "MGB" : "10000000"
 },
  "Params": [{
    "paramName":  "interfaceName",
    "paramValue": "eth0"
    }]
}
```

D. Slice definition: Proposed model, policies and logical implementation

In this study, we adopt the Transport Slice definition put forth by the Internet Engineering Task Force (IETF) [4]. According to this definition, the underlying transport network must be capable of dynamically configuring its network devices through controllers to provide transport transmissions that meet some or all of the Service Level Objectives (SLOs) for all traffic or specific flows in the slice. To achieve this, we utilize the DPP Network Application (Figure 2). This application enables us to configure network devices on demand by programming the underlying smartNIC with a customized eBPF algorithm and control maps. This approach allows for control communication between user space, kernel, and hardware via eBPF syscalls. The Linux source code defines eBPF maps using *enum bpf_map_type* in */usr/include/linux/bpf.h*. To store information about the slice definition of each network flow, including its pre-calculated flow hash identifier and the specific queue where it should be associated, we use the BPF_MAP_TYPE_HASH map type. Each entry in this data structure is referred to as a "control rule" in this study. This ensures that the eBPF maps are optimized to handle the specific characteristics of the network traffic, thus enabling efficient control and management of network flows.

Listing 2: Packet metadata structure

```
struct pkt_meta {
        __u16 out_macsrc[3];
        __u16 out_macdst[3];
        __u16 in_macsrc[3];
        __u16 in_macdst[3];
        __u16 out_ethproto;
        __u16 in_ethproto;
        __be32 out_ip4src;
        __be32 out_ip4dst;
        __be32 in_ip4src;
        __be32 in_ip4dst;
        __u8 out_ip4proto;
        __u8 in_ip4proto;
        union {
```

```
                __u32 out_ports;
                __u16 out_port16[2];
        };
        union {
                __u32 in_ports;
                __u16 in_port16[2];
        };
        __u32 vni;
        __be32 teid;
        __u32 ethercatid
};
```

Algorithm 1 eBPF-XDP packet steering algorithm for 6G networks

```
1:  struct pkt_meta;
2:  procedure XDP_PROG(pkt)
3:      pkt_meta ← parse_headers(pkt);
4:      hash ← calculate_hash(pkt_meta);
5:      control_entry ← control_map.lookup(hash);
6:      if control_entry != NULL then
7:          if control_entry.action == 0 then
8:              return XDP_DROP
9:          if control_entry.action == 1 then
10:             pkt.rx_queue ← control_entry.queue
11:     else
12:         return XDP_PASS
13:     monitor_entry ← monitor_map.lookup(hash);
14:     if monitor_entry != NULL then
15:         monitor.update(hash,pkt_meta,1)
16:     return XDP_PASS;
```

Algorithm 1 shows the XDP offloaded pipeline logic implemented inside the Agilio CX SmartNIC using C language. This algorithm allows traffic classification, filtering and steering while accelerating the network segment it represents. This algorithm allows to dissect double encapsulated pre-6G network traffic extracting the metadata required to create a unique hash, which is used to distribute the network flows among the different XSK sockets, guarantying the acceleration of the traffic between the network card and the user space. From the point of view of the SmartNIC, when a packet arrives to the physical interface, the offloaded eBPF program captures the packet and starts extracting the packet headers (lines 3). This routine parses and analyses every single bit of the incoming packet and extracts the information required to complete the pkt_meta structure shown in listing 2. This data structure contains the information required to identify a pre-6G network flow, including inner and outer MAC, IP addresses, ports, link protocols, transport protocols, and also, VXLAN and GTP identifiers. This data is used to calculate a 32 bits long hash (line 4), which unequivocally and individually identify an incoming network packet. This hash is used to check if there is any entry in the control map that matches it (line 5 and 6). If so, the rule is checked and the action value is obtained. If the action value is 0 then the packet is dropped (line 7 and 8). However, if the action value is 1, it means that the packet is transmitted to a XSK socket to be sent to user space (line 9). To do so, the value of the output queue/socket is obtained from the control rule and the packet configuration metadata is updated with this value (line 10). In the event that there is no control rule that matches the hash value, the packet is sent to the user space using the default queue/socket (XDP_PASS), which is the 0 (line 12). If the packet has not been dropped, the monitoring map is checked to see if there is any rule that matches with the hash of the incoming packet (line 13). If there is a rule that matches with the hash of the incoming packet (line 14), then the rule is updated by adding the information of the current packet processed, and increasing by 1 the packet counter (line 15). Finally, the packet is transmitted to user space (line 16) using the output queue previously selected in line 5.

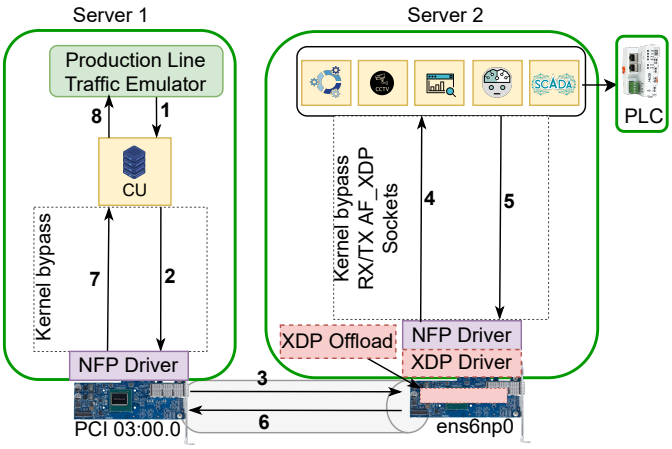


Fig. 3: Implementation of the performance evaluation testbed

VI. Empirical Validation

A. Experimental setup

This section describes the experimental setup used to carry out the empirical validation of the framework proposed, as illustrated in Figure 3. For the development of this experimental environment, a cloud service provider architecture has been replicated using two physical servers, one allocating the CU and the second one with the UPF and vertical services. The two separate physical servers have the same specifications: CyberServe XE5-212S v4, X10DRi-T, Dual Intel 10GbE LAN, 2TB SATA 7200RPM and Intel Solid-State Drive DC P3600 Series 1.2TB. In addition, both server 1 and 2 have a Agilio CX 2x25GbE SmartNIC. With regards to software specifications, both servers have an operating system Ubuntu 21.04 (hirsute) with a Linux kernel 5.11.0-40-generic. Furthermore, a customised version of the NFP driver with AF_XDP sockets support is running in both servers. Server 1 has a CU developed in a Linux Container and using OperAirInterface [41]. In addition, it also contains a product line traffic emulator using DPDK 21.02.0 and Pktgen 21.03.1. Server 2 has a customised XDP offload firmware instantiated in the SmartNIC, the AF_XDP sockets by-passing the kernel and different emulated vertical services using Linux containers. The XDP firmware implements a new pipeline with support for 6G networks and with control and real-time monitoring capabilities. Both servers are connected using an Avago 25GBase-SR SFP28.

The following text summarises the steps followed to perform the execution of the experimental setup shown in Figure 3:

1) On server 1, there is a tool called the product line traffic emulator that acts as a sensor and generates network traffic by transferring video from a physical device to the vertical service. To emulate the sender, the tool uses Pktgen, which is deployed inside the CU. The traffic is emulated using double-encapsulated and transmits Real-Time Protocol (RTP) video frames..
2) On Server 1, network traffic generated in the CU is transmitted directly to the SmartNIC, bypassing the Linux kernel for transmission. To make this communication possible, DPDK has been used.
3) On Server 1, the SmartNIC consumes traffic from the CU via the PCI slot 03:00.0 and forwards it to Server 2 through its physical interface. The traffic can be transmitted at speeds of up to 25Gbps in optimal conditions.
4) After being transmitted through the PCI slot on Server 1, the traffic is received on Server 2 through the interface ens6np0. The SmartNIC then processes the traffic using an offloaded pipeline, matching the packet metadata against rules stored in the control maps to determine the appropriate action. Additionally, the monitoring hardware map is updated with information about the incoming traffic, which is used later for performance analysis. If the traffic doesn't match any rules, it is discarded. However, if a rule is matched, the traffic is forwarded to the vertical service using an AF_XDP socket, bypassing the Linux kernel for improved performance. It's worth noting that this behavior is not supported by the upstreamed NFP driver, and is an innovation of this research work.
5) Next, the network traffic is finally received by the vertical service, where the target application processes the traffic accordingly. Additionally, the vertical service also gathers metrics from the SmartNIC hardware monitoring maps and updates control rules as needed.
6) Once a decision has been made and the network flow has been modified by the vertical service introducing the action parameter in the payload of the packet, this network flow is sent back to the SmartNIC using the same AF_XDP socket and bypassing the Linux kernel.
7) The network flow is forwarded to the Server 1 using the physical interface ens6np0.
8) The SmartNIC allocated in Server 2 sends the network traffic to the CU.
9) Finally, the network flow, which contains the decision taken by the vertical service, is received by the production line emulator which perform the associated action accordingly.

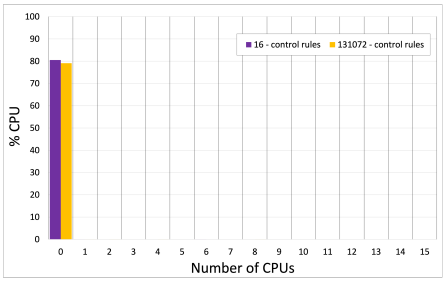
B. Results

This section empirically validates the accelerated data path for pre-6G networks proposed in this manuscript. The goal here is to demonstrate that the demanding vertical services related to future 6G networks and I4.0 use cases meet the necessary quality of service (QoS) and performance requirements. This is achieved by providing high reliability and low-latency communication, even in highly congested scenarios where real-time communication is crucial for safe and optimal operations. To maximise the performance of this communication is critical to isolate the communication of the different product lines of a factory and to do so, this paper proposes isolated AF_XDP sockets to guarantee this communication between the physical server and the product line. Each socket has dedicated CPU resources in order to obtain the maximum performance of the system. In addition, the irq_affinity is set so that one CPU is in charge of processing a particular system interrupt so the number of CPUs is directly proportional to the number of network receiving queues reserved in the system. To validate the performance, reliability, and latency of the solution proposed, different charts have been elaborated and are presented in this manuscript. These graphs present statistics about percentage of CPU usage and packet loss when number of control rules, packet size and sockets are modified. And also Round Trip Time (RTT) latency based on the packet size. The experiments have been executed transmitting up to 25Gbps and 20M packets per second, depending on the packet size. To test the requirements of the different use cases presented in section IV, specially advanced network slicing, packet size ranges from 132 bytes, which is the minimum required for a 6G double encapsulated packet, and 1500 bytes, as can be seen in Table I. Furthermore, the maximum number of sockets supported is 16, and thus, the maximum number of CPU used by the framework is 16, divided into five different executions: 1, 2, 4, 8 and 16 sockets. Notice that the experiments have been executed five times and the values obtained are the average of those five executions. Although the optimal use of these sockets can prioritize specific network services with different SLAs, this research primarily focuses on validating the scalability of the proposed solution. Therefore, to ensure a fair validation, all transmitted service flows are distributed equally across the available queues/sockets. Moreover, to align with the underlying architecture and facilitate their matching within the network traffic injected into the system, control eBPF maps have been prepopulated with specific control rules.

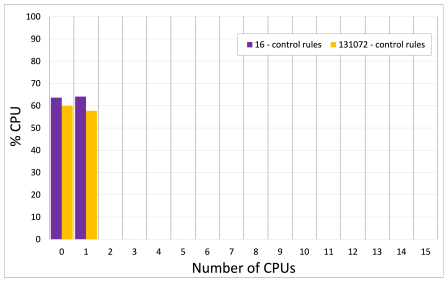
1) CPU Load: Figure 4 shows the percentage of CPU used when up to 16 CPUs are performing simultaneously and up to 131072 rules are inserted in the SmartNIC offload map. (a), (b), (c), (d) and (e) represent different experiments with 1, 2, 4, 8 and 16 sockets, respectively. The number of execution per experiment carried out is:

$$16 * 2^n$$

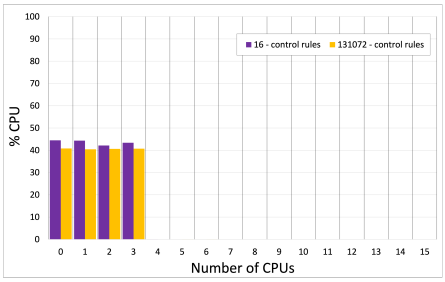
where n=$\{0,..,13\}$. It is noted that each CPU is exclusively in charge of processing a particular system interrupt and therefore a particular CPU. For this experiment, a packet sizes ranging from 132 to 1500 bytes have been used and the results shown are an average of all the traffic equally distributed per socket using the control rules. Graph (a) in Figure 4 can be considered the baseline scenario where there is only one socket and therefore, all network traffic is processed by the same CPU. This CPU is performing with an average load of 70% and as it can be appreciated in the graph, the CPU load does



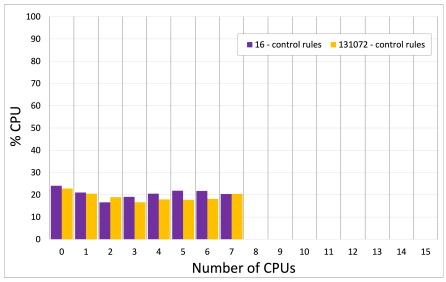
(a) 1: %CPU Usage vs Control Rules with 1 Sockets.



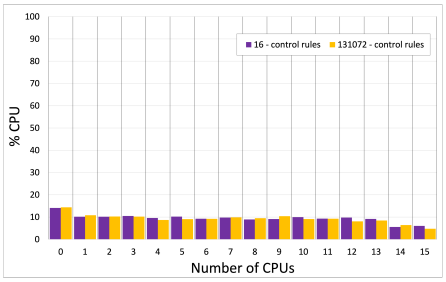
(b) 2: %CPU Usage vs Control Rules with 2 Sockets.



(c) 3: %CPU Usage vs Control Rules with 4 Sockets.



(d) 4: %CPU Usage vs Control Rules with 8 Sockets.



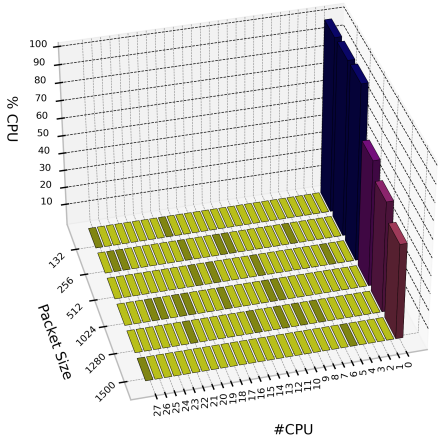
(e) 5: %CPU Usage vs Control Rules with 16 Sockets.

Fig. 4: Relation between % CPU used, number of CPU and number of control rules with different number of sockets. The irq_affinity is set so that one CPU is in charge of processing a particular system interrupt so the number of CPUs is directly proportional to the number of network receiving queues reserved in the system.

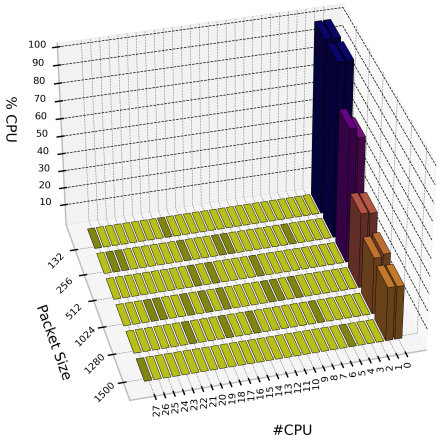
not increase when the number of rules is incremented. Graph (b) represents an scenario where two sockets are transmitting network traffic simultaneously and as it is shown in the graph, the CPU load has decreased to an average of 50%. Similarly, in (c), (d) and (e) the CPU load decreases when the number of sockets performing increases. In the best case scenario, when 16 sockets are processing traffic, the CPU load is below 10%. It is noted that per each of the flows processed the control map needs to be checked to see if there is any rule that matches with the incoming network packet. In this context, when 131072 control rules are inserted in the SmartNIC hardware map, the CPU load remains constant, which demonstrates the scalability of the system in terms of system load. Over the 90% of the idle CPU load could be used for any other task while up to 25Gbps of network traffic is processed, controlled and monitored thanks to the solution proposed in this paper.

Figure 5 depicts % of CPU used when the packet size changes from 132 up to 1500 bytes. In the same way as in the previous graphs, number of sockets range with a magnitude of $16 * 2^n$ where n=$\{0,..,13\}$. Graph (a) can be considered the baseline scenario where only one socket is processing the traffic. In other words, the baseline scenario mimics the behavior of a NIC that lacks the ability to generate an RSS hash identifiers for multi-encapsulated network traffic. It can be seen that CPU load is almost 100% when the size of the packet processed is below 512 bytes, due to the number of software interruptions generated in the system per each of the packet processed. However, this value decreased up to 35% when the packet size increases to 1500 bytes. When the number of sockets is increased, see graphs (b), (c), (d) and (e), the percentage of CPU load decreases and thus, allows the system to be more idle. In the scenario where 16 CPUs are processing the network traffic, the percentage of CPU usage with a packet size of 132 bytes is around 25%, however it decreases to 10 % with 256 bytes packet size and below 5% when packet size if bigger than 512 bytes. It demonstrates the scalability of the eBPF/XDP-based framework proposed even when the size of the packet processed is small and the number system interruptions is very high.

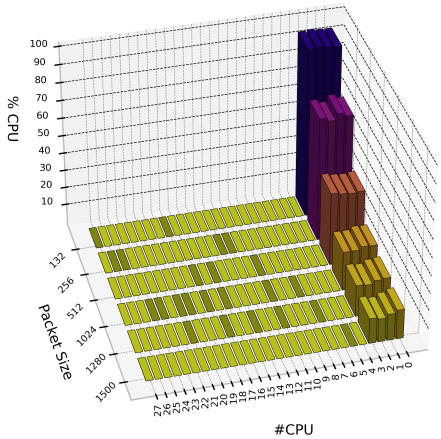
2) Packet Loss: Percentage of packet loss in different scenarios can demonstrate the reliability of the system tested. Figure 6 depicts the percentage of packet loss when the packet size is changed and the number of control rules inserted in the hardware map is modified. The experiments carried out to plot this validation are the same that the ones used for the previous experiments, however it is now focused on packet loss. Graph (a) shows the baseline scenario with only one socket performing. In this case, the percentage of packet loss is around 70% when packet size is 132 bytes. In this experiment, almost 0% packet loss is achieved when packet size is 1024, 1280 and 1500 bytes. In graph (b), with two sockets processing traffic, the solution still loses around 40% of the packets when packet size is 132 bytes, however with a packet size of 512 bytes it decreases close to 0%. The network traffic processed by the SmartNIC is equitably distribute between the different sockets. Graphs (c), (d) and (e) shows how when number of sockets created in the framework is 4 or higher, the reliability is close to 100%, due to packet loss is almost 0% even in worst case scenario when packet size is 132 bytes and number of control rules is 131072. Meaning that the access time to the hardware maps is constant and also demonstrates the reliability of the system when the number of control rules is increased.



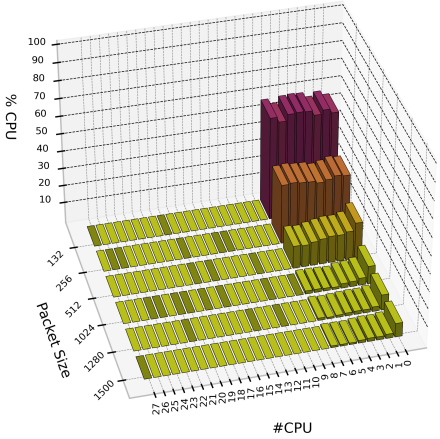
(a) 1: %CPU Usage vs Packet size(Bytes) with 1 socket.



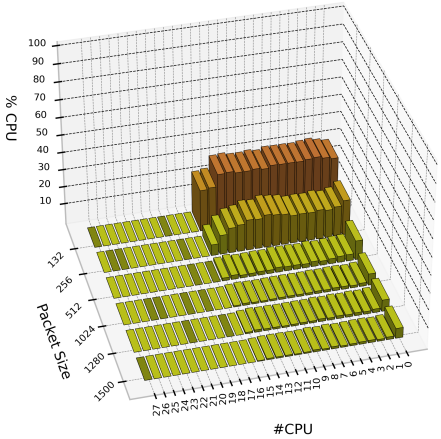
(b) 2: %CPU Usage vs Packet size(Bytes) with 2 sockets.



(c) 3: %CPU Usage vs Packet size(Bytes) with 4 sockets.



(d) 4: %CPU Usage vs Packet size(Bytes) with 8 sockets.



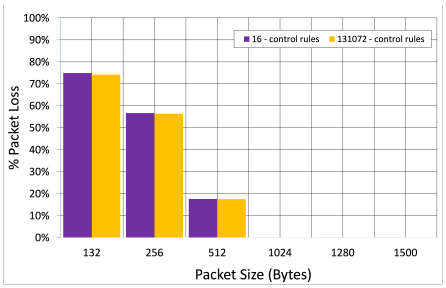
(e) 5: %CPU Usage vs Packet size(Bytes) with 16 sockets.

Fig. 5: Relation between % CPU used, number of CPU and packet size in bytes with different number of sockets. The irq_affinity is set so that one CPU is in charge of processing a particular system interrupt so the number of CPUs is directly proportional to the number of network receiving queues reserved in the system.

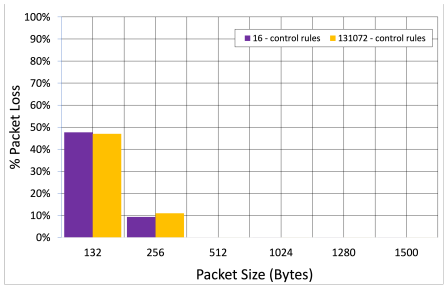
In these scenarios where the CPUs are equally distributed and attached to the different sockets and also where these socket are totally isolated between them, it is interesting to study the performance of each socket itself. In this case, the traffic has been distributed equally between the different sockets, so theoretically the behaviour of the different sockets should be the same. The distribution of the different network flows transmitted is done using up to 131072 control rules inserted in the hardware card. As it is represented in Figure 7, the tendency of the different graphs is very similar to the graphs depicts in Figure 6, in fact it is the same data but showing different information. In this figure, it has been calculated the average of all control rules and it is focused in showing a comparison between the packet size and the number of sockets. Again graph (a) is considered the baseline scenario where almost 70% of the packets are lost when packet size is 132 bytes. This value becomes 0 when the number of socket used in the framework presented is 4 or higher.

The high reliability and high performance demonstrated through the empirical validation of this paper verify the suitability of the solution to cover the demanding requirements of 6G use cases shown in Table I. Although this work is focused on providing an advanced network slicing solution, depending on the number of queues created and the rules inserted in the control and monitoring map, this solution can be used to successfully deployed all the use cases previously commented.

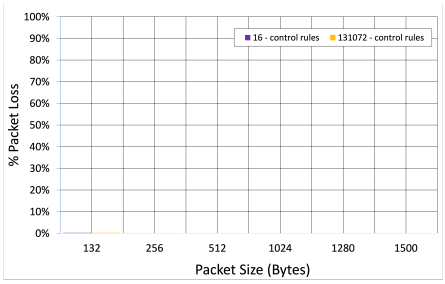
3) Bandwidth: The scalability test of the framework deployed has been tested by calculating the Packet Per Second (PPS) received while the packet size and number of sockets is modified, as it is shown in Figure 8. In order to evaluate the performance of a system, three different concepts need to be considered: (a) Pps, which represents the number of packets per second transmitted through the network; (b) Bandwidth, which refers to the maximum data volume capacity of a network; and (c) Throughput, which make reference to the amount of data traveling successfully across a network. Notice that in this article the value shown in Figure 8 is the number of PPS received by the sockets (bandwidth) and not the PPS processed (throughput). To calculate the total PPS processed



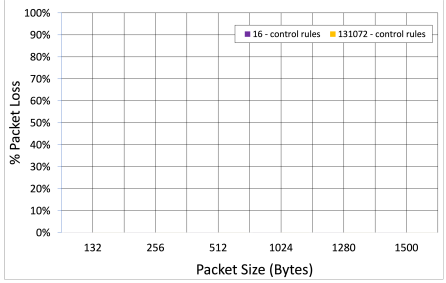
(a) 1: %Packet loss vs Control Rules with 1 socket.



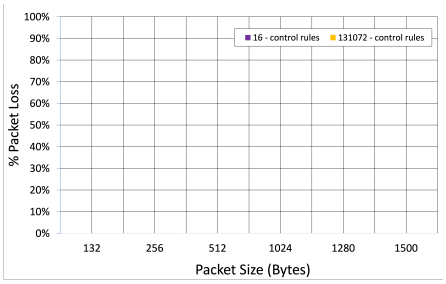
(b) 2: %Packet loss vs Control Rules with 2 sockets.



(c) 3: %Packet loss vs Control Rules with 4 sockets.

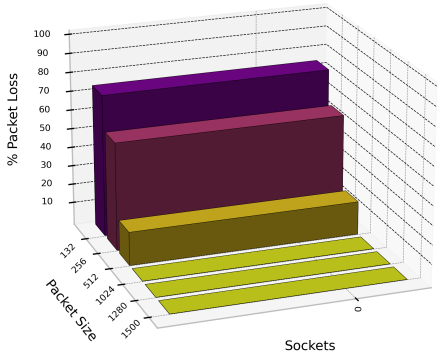


(d) 4: %Packet loss vs Control Rules with 8 sockets.

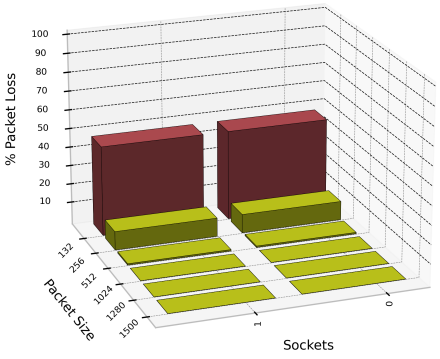


(e) 5: %Packet loss vs Control Rules with 16 sockets.

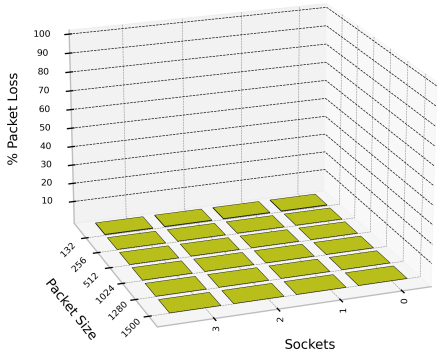
Fig. 6: Relation between % packet loss, packet size in bytes and number of control rules with different number of sockets.



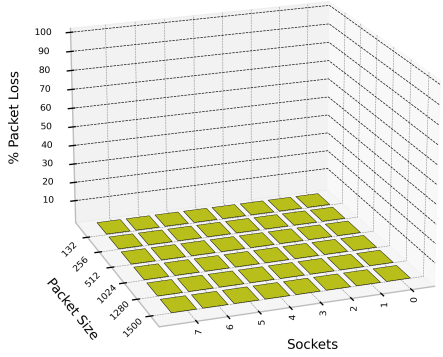
(a) 1: %Packet loss distribution with 1 socket.



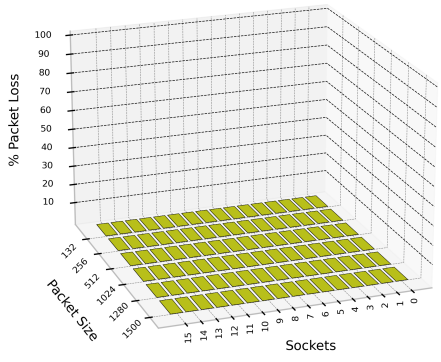
(b) 2: %Packet loss distribution with 2 socket.



(c) 3: %Packet loss distribution with 4 socket.



(d) 4: %Packet loss distribution with 8 socket.



(e) 5: %Packet loss distribution with 16 socket.

Fig. 7: Relation between % packet loss, packet size in bytes and number of sockets.

per socket, the reader has to consider that the sample taken to carry out each of the experiments is 30 seconds long and also the number of packet loss per socket shown in Figure 7. Since the total throughput or bandwidth vary based on the packet size and thus, it does not really represent the real performance of the solution, these graphs have not been included in this

manuscript, however these values can be directly calculated based on the PPS results, bandwidth received per socket:

$$BW(Gbps) = (PPS * PacketSize)/10^9$$

or throughput processed per socket

$$TP(Gbps) = ((PPS * PacketSize)/10^9) - PacketLoss$$

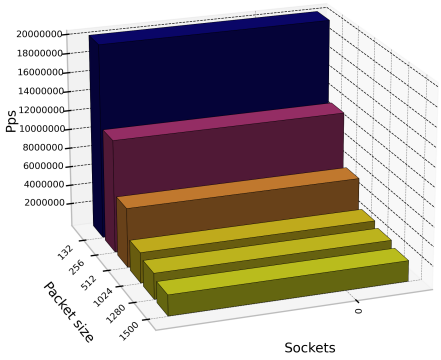
For this experimentation, the same quantity of packet per second have been sent from the packet generator to the different sockets created and therefore, almost the same quantity of network traffic is simultaneously processed by each of them. As it can be seen in 8 graph (a), the number of PPS received by the framework proposed is almost 20M when packet size transmitted is 132B. However, this is the baseline and worst case scenario and almost 70% of these packets are lost while they are processed. When the number of sockets increase the quantity of traffic received is practically the same however the number of packet loss decreases. Notice that when the packet size increases the number of packets received per socket decreases and it is because in these experiments the maximum bandwidth supported by the SmartNIC (25Gbps) is achieved with fewer packets. As it is demonstrated in graph (c) the maximum PPS processed by each socket without losing almost any packet is around 5M PPS. In the experimentation carried out, with 25Gbps of peak bandwidth, the maximum performance is achieved when the number of sockets used is 4 or more, having almost 0% of packet loss in all the experiments executed. Furthermore, the bandwidth values achieved meet the average bandwidth required by the different pre-6G vertical use cases shown in table I and it demonstrates the viability of the solution proposed in terms of bandwidth.

C. Latency

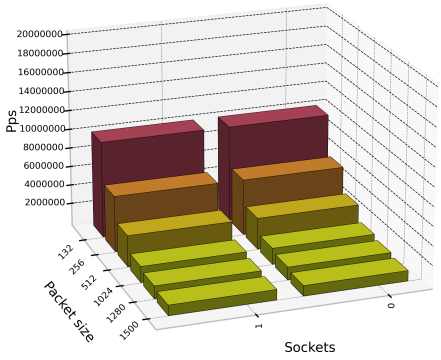
A critical key performance indicator in novel pre-6G networks is latency, which is associated with network efficiency, connection speed, and bandwidth. Reduced network transmission latency directly affects the Quality of Experience (QoE) for the end user. Figure 9 presents average RTTs for diverse pre-6G mobile network communications between the two servers portrayed in Figure 3. These servers form the production line and the cloud service provider infrastructure. The experiments have used the same packet sizes as in previous executions, with a maximum processing rate of 25Gbps. The displayed values represent the delay of a packet traveling from the packet generator server, crossing the network via a fiber cable, passing the XDP hardware data path with offloaded maps, and reaching the vertical application through the AF_XDP driver sockets. This process also includes traffic processing in the vertical application and return to origin via the same socket and cable. Please note, this research focuses on stages up to the packet's arrival at the AF_XDP sockets. The reported RTT times include the entire process, even the processing in user space (vertical application), which is beyond the scope of the proposed solution. The graph reveals an increase in RTT as packet size enlarges, largely attributed to more time needed for processing in the vertical application, producing a new packet, and re-injecting it back into the network. However, at the hardware and driver level, the most demanding scenario arises when packets of 132B are sent at 25Gbps, implying that a total of 20 million packets need to be processed each second. In this scenario, the RTT is approximately 0.13 ms. As per table I, the maximum allowable delay for the critical services described in this article in pre-6G networks is 0.1 ms for end-to-end communication. Given the proposed solution can achieve a round-trip transmission near 0.1ms, it's clear the framework can meet the stringent latency KPIs of pre-6G networks. Nevertheless, challenges may arise in specific scenarios, such as Use Case 5, requiring large packet sizes and extremely low latency. Figure 9 shows that with a packet size of 1500 bytes, the average RTT increases to 0.55 ms. In such cases, a smarNIC with higher capabilities, more hardware queues to distribute the traffic, and increased server CPU for dequeuing and processing network traffic at the user-space level would be needed. Another alternative would be to distribute traffic among different servers, similar to a load balancer, an approach we explored in our previous research article [7].

VII. Conclusions

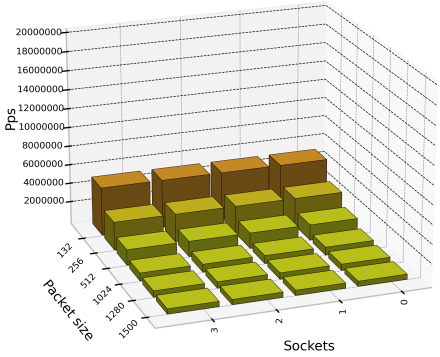
The demanding performance requirements of the next generation mobile networks and the novel use cases entail high-performance packet processing, control and monitoring capabilities. This paper has proposed a new ultra-high packet processing and monitoring network framework for future 6G networks. The proposed novel eBPF/XDP-based network filtering mechanism is able to dynamically identify multi-tenant flows and take decision over the traffic processed in order to fulfil the requirements of a concrete pre-6G network Industry 4.0 use case. In parallel with this, this framework is able to report real-time network metrics by using eBPF hardware maps, which provide constant access time to memory. The proposed programmable network data plane explores a hardware-based offloading approach using an Agilio CX smart network interface card that combines novel technologies such as eBPF, XDP and AF_XDP to bypass the Linux kernel, and thus accelerates the communication between the network card and the user applications. In order to support this, the NFP driver in charge of controlling the SmartNIC has been modified to provide AF_XDP support. This framework has been implemented and tested in a realistic beyond 5G architecture, obtaining experimental results that demonstrate the viability of the solution proposed. Moreover, the advantageous performance results in terms of packet loss and percentage of CPU used across various scenarios, modifying the number of AF_XDP sockets and the control rules, clearly show the scalability and suitability of the solution to meet the demanding KPIs of diverse pre-6G use cases. Moreover, we plan to address certain limitations in our current research. For instance, we will investigate factors such as geographical distance between connection points to identify new challenges in complex network topologies. By expanding our research in these areas, we hope to contribute to the development of robust and effective network solutions for the future.



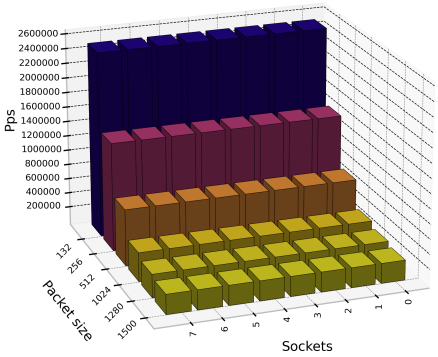
(a) 1: Pps vs Packet size(Bytes) vs Sockets.



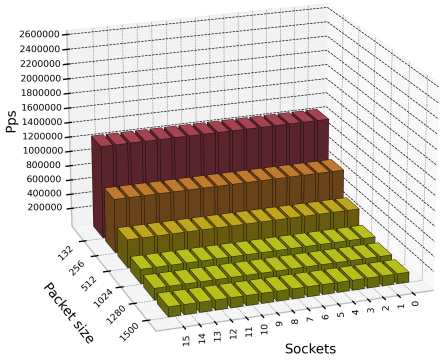
(b) 2: Pps vs Packet size(Bytes) vs Sockets.



(c) 3: Pps vs Packet size(Bytes) vs Sockets.



(d) 4: Pps vs Packet size(Bytes) vs Sockets.



(e) 5: Pps vs Packet size(Bytes) vs Sockets.

Fig. 8: Relation between packet per second received, packet size in bytes and number of sockets.

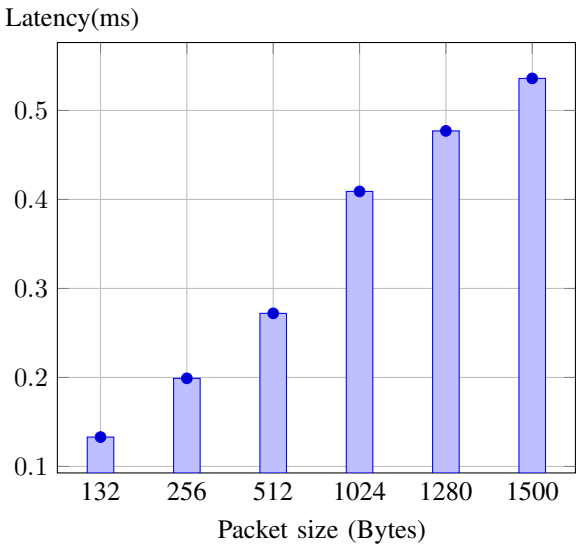


Fig. 9: RTT latency time in milliseconds when packet size varies.

ACKNOWLEDGMENT

This work was funded in part by the European Commission Horizon 2020 5G-PPP Program under Grant Agreement Number H2020-ICT-2020-2/101017226 “6G BRAINS: Bringing Reinforcement learning Into Radio Light Network for Massive Connections” and under Grant Agreement Number H2020-ICT-2020-2/101016941 “5G INDUCE: Open Cooperative 5G Experimentation Platforms for The Industrial Sector NetApps”. We thank Simon Horman and Niklas Soderlund (Corigine Inc.) for providing the NFP driver that enabled this research.

REFERENCES

[1] W. Saad, M. Bennis, M. Chen, and A. V. Vasilakos, “A vision of 6g wireless systems: Applications, trends, technologies, and open research problems,” *IEEE Network*, vol. 35, no. 1, pp. 186–192, 2021.

[2] P. Singh, A. Nayyar, A. Kaur, and U. Ghosh, “Blockchain and fog based architecture for internet of everything in smart cities,” *Future Internet*, vol. 12, no. 4, p. 61, 2020.

[3] P. K. Padhi and F. Charrua-Santos, “6g enabled industrial internet of everything: towards a theoretical framework,” *Applied System Innovation*, vol. 4, no. 1, p. 11, 2021.

[4] R. R. Tantsura, S. Homma, K. Makhijani, L. M. Co, ntreras, and Jeff, “IETF Definition of Transport Slice,” 2020. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-nsdt-teas-transport-slice-definition

[5] X. Li, M. Samaka, H. A. Chan, D. Bhamare, L. Gupta, C. Guo, and R. Jain, “Network slicing for 5g: Challenges and opportunities,” *IEEE Internet Computing*, vol. 23, no. 5, pp. 16–23, 2019.

[6] X. Foukas, G. Patounas, and M. K. Marina, “Network slicing in 5g: Survey and challenges,” *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1669–1713, 2020.

[7] P. Salva-Garcia, R. Ricart-Sanchez, E. Chirivella-Perez, Q. Wang, and J. M. Alcaraz-Calero, “Xdp-based smartnic hardware performance acceleration for next-generation networks,” *Journal of Network and Systems Management*, 2022.

[8] M. Majkowski, “Kernel bypass,” *https://blog.cloudflare.com/kernel-bypass/*, 2015.

[9] H. Zhu, *Data Plane Development Kit (DPDK): A Software Optimization Guide to the User Space-based Network Applications*. CRC Press, 2020.

[10] M. Paolino, N. Nikolaev, J. Fanguede, and D. Raho, “Snabbswitch user space virtual switch benchmark and performance optimization for nfv,” in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. IEEE, 2015, pp. 86–92.

[11] V. Maffione, L. Rizzo, and G. Lettieri, "Flexible virtual machine networking using netmap passthrough," in *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, 2016, pp. 1–6.

[12] L. Kernel, "Linux packet mmap," *https://www.kernel.org/doc/Documentation/networking/packet_mmap.txt*, accessed: July 15,2023.

[13] ntop, "PF_RING: High-speed packet capture, filtering and analysis," 2023, accessed: July 23,2023. [Online]. Available: https://www.ntop.org/products/packet-capture/pf_ring/

[14] Xilinx, "Ef_vi user guide," *https://china.xilinx.com/content/dam/xilinx/publications/solarflare/onload/openonload/packages/SF-114063-CD-10_ef_vi_User_Guide.pdf*, accessed: July 15,2023.

[15] Kernel, "Af_xdp - af_xdp socket (xsk)," *https://www.kernel.org/doc/html/latest/networking/af_xdp.html*, accessed: July 15,2023.

[16] A. Xilinx, "X2 series ethernet adapters - xtremescale x2522, x2541," *https://www.xilinx.com/products/boards-and-kits/x2-series.html*, accessed: July 20,2023.

[17] A. Xilinx, "8000 series ethernet adapters - 10/40gbe network adapters," *https://www.xilinx.com/products/boards-and-kits/8000-series.html*, accessed: July 20,2023.

[18] A. Xilinx, "Alveo u50 data center accelerator card," *https://www.xilinx.com/products/boards-and-kits/alveo/u50.html*, accessed: July 20,2023.

[19] A. Xilinx, "Alveo u200 data center accelerator card," *https://www.xilinx.com/products/boards-and-kits/alveo/u200.html*, accessed: July 20,2023.

[20] A. Xilinx, "Alveo u250 data center accelerator card," *https://www.xilinx.com/products/boards-and-kits/alveo/u250.html*, accessed: July 20,2023.

[21] A. Xilinx, "Alveo u280 data center accelerator card," *https://www.xilinx.com/products/boards-and-kits/alveo/u280.html*, accessed: July 20,2023.

[22] Intel, "Intel fpga smartnic n6000-pl platform," *https://www.intel.com/content/www/us/en/products/details/fpga/platforms/smartnic/n6000-pl-platform.html*, accessed: July 20,2023.

[23] Silicom, "Silicom fpga smartnic n5010 series," *https://www.silicom-usa.com/pr/fpga-based-cards/fpga-intel-based/fpga-intel-stratix-based/silicom-fpga-smartnic-n5010_series/*, accessed: July 20,2023.

[24] Intel, "Intel fpga programmable acceleration card n3000," *https://www.intel.com/content/www/us/en/products/details/fpga/platforms/pac/n3000.html*, accessed: July 20,2023.

[25] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "Netfpga sume: Toward 100 gbps as research commodity," *IEEE micro*, vol. 34, no. 5, pp. 32–41, 2014.

[26] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[27] J. Cavanagh, *Digital design and Verilog HDL fundamentals*. CRC press, 2017.

[28] R. Ricart-Sanchez, A. C. Aleixo, Q. Wang, and J. M. A. Calero, "Hardware-based network slicing for supporting smart grids self-healing over 5g networks," in *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2020, pp. 1–6.

[29] R. Ricart-Sanchez, P. Malagon, J. M. Alcaraz-Calero, and Q. Wang, "P4-netfpga-based network slicing solution for 5g mec architectures," in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 2019, pp. 1–2.

[30] R. Ricart-Sanchez, P. Malagon, J. M. Alcaraz-Calero, and Q. Wang, "Netfpga-based firewall solution for 5g multi-tenant architectures," in *2019 IEEE International Conference on Edge Computing (EDGE)*. IEEE, 2019, pp. 132–136.

[31] Q. Wang, J. Alcaraz-Calero, R. Ricart-Sanchez, M. B. Weiss, A. Gavras, N. Nikaein, X. Vasilakos, B. Giacomo, G. Pietro, M. Roddy *et al.*, "Enable advanced qos-aware network slicing in 5g networks for slice-based media use cases," *IEEE transactions on broadcasting*, vol. 65, no. 2, pp. 444–453, 2019.

[32] Netronome, "Netronome agilio cx smartnics," *https://www.netronome.com/products/agilio-cx/*, accessed: July 21,2023.

[33] Corigine, "Corigine agilio cx smartnics," *https://www.corigine.com/smartnicdetail-31.html*, accessed: July 15,2023.

[34] H. Tataria, M. Shafi, A. F. Molisch, M. Dohler, H. Sjöland, and F. Tufvesson, "6g wireless systems: Vision, requirements, challenges, insights, and opportunities," *Proceedings of the IEEE*, vol. 109, no. 7, pp. 1166–1199, 2021.

[35] A. Artemenko, Y. Zhang, U. Wostradowski, J. Cosmas, Q. Wang, J. M. Alcaraz-Calero, E. C. Perez, P. Salva-Garcia, and R. Ricart-Sanchez, "6G BRAINS: D2.1 Definition and Description of the 6G Primary Use Cases and Derivation of User Requirements," Jun. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.5185388

[36] L. Kernel, "Libbpf library," *https://www.kernel.org/doc/html/latest/bpf/libbpf/index.html*, accessed: July 11,2023.

[37] L. Kernel, "Xsk library," *https://github.com/torvalds/linux/blob/master/tools/lib/bpf/xsk.c*, accessed: July 11,2023.

[38] E. Chirivella-Perez, P. Salva-Garcia, I. Sanchez-Navarro, J. M. Alcaraz-Calero, and Q. Wang, "E2e network slice management framework for 5g multi-tenant networks," *Journal of Communications and Networks*, pp. 1–13, 2023.

[39] J. Fonseca, M. Khadmaoui-Bichouna, B. Mendes, P. Duarte, M. Araujo, D. Corujo, I. Sanchez-Navarro, A. Matencio-Escolar, P. Salva-Garcia, J. M. Alcaraz-Calero, and Q. Wang, "6g brains topology-aware industry-grade network slice management and orchestration," in *2023 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*. IEEE, 2023, pp. 341–346.

[40] A. Kazmierowski, J. Kodjabachian, P. Salva-Garcia, A. Matencio-Escolar, and I. Sanchez-Navarro, "6G Brains: D5.2 Preliminary integration for AI-based E2E network slicing control and MANO," Dec. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.7468007

[41] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, "Openairinterface: A flexible platform for 5g research," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 33–38, 2014.

Pablo Salva-Garcia is a Lecturer at the University of the West of Scotland. Pablo is Co-investigator in several Horizon 2020 EU projects, such as 6G-BRAINS, 5G-INDUCE and ARCADIAN-IoT and member of the B5G-Hub. His main interests are Network management, data plane programmability, SDN, and Beyond 5G Networks.

Ruben Ricart-Sanchez is a researcher at the University of the West of Scotland, where he obtained his PhD. Ruben is Co-investigator in several Horizon 2020 EU projects, such as 6G-BRAINS, 5G-INDUCE and ARCADIAN-IoT. His main interests include 5G/6G Networks, Network management, programmable hardware, and network security.

Jose M. Alcaraz-Calero is a Professor in next-generation networks and security at the University of the West of Scotland. He is the technical co-coordinator of the EU H2020 5G-PPP SELFNET and SliceNet projects, and co-principal investigator of EU H2020 5G INDUCE and 6G BRAINS projects. His professional interests include network cognition, management, security and control, service deployment, automation and orchestration, and 5G mobile networks.

Qi Wang is a Professor at the University of the West of Scotland. He is the technical co-coordinator of EU H2020 5G-PPP SELFNET and SliceNet projects, and co-principal investigator of EU H2020 5G INDUCE and 6G BRAINS projects. He is a Board Member of the Technology Board of EU 5G-PPP. His research primarily focuses on 5G mobile networks, video networking and artificial intelligence.

Octavio Herrera-Ruiz obtained a PhD in Telecommunications by the University of Pittsburgh, USA in 2011. He is Manager of Mira/Netronome Global Support. His research interests include network performance and survivability.