Journal of Network and Systems Management (2022) 30: 75
https://doi.org/10.1007/s10922-022-09687-z

GENERAL SUBMISSION

XDP-Based SmartNIC Hardware Performance Acceleration for Next-Generation Networks

Pablo Salva-Garcia[1] · Ruben Ricart-Sanchez[1] · Enrique Chirivella-Perez[1] · Qi Wang[1] · Jose M. Alcaraz-Calero[1]

Received: 6 December 2021 / Revised: 4 August 2022 / Accepted: 10 August 2022 /
Published online: 9 September 2022

Abstract

Next-generation networks are expected to combine advanced physical and digital technologies in super-high-speed connected system infrastructures, gaining critical operation competitiveness of improved efficiency, productivity and quality of services. Towards a fully digital and connected world, these platforms will enable infrastructure virtualization and support of edge processing, making emerging sectors, such as Industry 4.0, ready to exploit its full potentials. Nevertheless, the fast growth of data-centric and automated systems may exceed the capabilities of the overall infrastructure beyond the radio access networks, becoming unable to fulfil the demands of vertical sectors and representing a bottleneck. To minimize the negative effects that could affect critical services in a heavily loaded network, it is essential for network providers to deploy highly scalable and prioritisable in-network optimisation schemes to meet industry expectations in next-generation networks. To this end, this work presents a novel framework that leverages extended Berkeley Packet Filter (eBPF) and eXpress Data Path (XDP) to offload network functions to reduce unnecessary overhead in the backbone infrastructure. The proposed solution is envisioned to be implemented as a Network Application (NetApp) service, which will greatly benefit the compatibility with next-generation networking ecosystem empowered by Artificial Intelligence (AI), advanced automation, multi-domain network slicing, and other related technologies. The achieved results demonstrate key performance improvements in terms of packet processing capacity as high as about 18 million packets per second (Mpps), system throughput up to 6.1 Mpps with 0% of packet loss, and illustrate the flexibility of the framework to adapt to multiple network policy rules dynamically on demand.

Keywords 5G/6G networks · Traffic engineering · Hardware acceleration · Network Offloading · XDP · BPF · Programmable networking

Ruben Ricart-Sanchez, Enrique Chirivella-Perez, Qi Wang and Jose M. Alcaraz-Calero have contributed equally to this work.

Extended author information available on the last page of the article

1 Introduction

The next-generation mobile networks (5G and beyond including pre-6G) entail novel management solutions to accommodate a wide range of use cases with advanced and heterogeneous requirements in terms of latency, resilience, coverage and bandwidth. It is envisioned that a radical revolution of societies is taking place, with the societies become more and more data-centric, data-dependent and automated, taking communications closer to the vision of Internet of Everything (IoE) [1]. Industrial manufacturing process (Industry 4.0), autonomous systems, and millions of Internet of Things (IoT) devices are increasingly connecting not just people, but also vehicles, devices, wearables, and a broad range of sensors [2]. The diverse requirements of future smart cities demand that current use cases will evolve to more heterogeneous implementations whilst quality assurance mechanisms such as network slicing are required to guarantee the Service-Level Agreements (SLAs) for diverging use cases over the same physical network [3]. Therefore, current 5G networks need to evolve to flexible architectures where the diversity and performance needed for these new services are assured. With the advent of virtualization and softwarisation technologies, operators expect networks to support flexible and rapid deployment of their Network Services (NSs) and Network Applications (NetApps). For example, in virtualised Mobile/Multi-access Edge Computing (MEC) infrastructures, services can be migrated/deployed at the edge, closer to the final user, to improve the overall service performance on demand. Furthermore, there have been significant advances in radio access technologies in 5G and beyond systems recently; however, the backbone infrastructure could become a new bottleneck if it is not upgraded accordingly to accommodate the rapidly growing traffic aggregated from the distributed radio access networks. Moreover, to allow service quality assurance in the backbone infrastructure, advanced packet processing capabilities for traffic engineering are entailed without compromising the super-high traffic transmission speed of the data plane, as expected in the next-generation networks.

Network Function Virtualization (NFV) and Software-Defined Networking (SDN) technologies bring numerous advantages in terms of dynamicity and flexibility to provide connectivity among the distributed NetApp deployment components. Meanwhile, they also introduce added complexities to the overall management and control of the architecture. Furthermore, network traffic traversing the data-plane of the infrastructure also becomes much more complex just when networks need to transfer much greater amounts of data, at much higher speeds. Particularly, the forwarding process of network packets is conducted by encapsulating them into different network protocols to allow differentiating multiple tenants, users and services, which makes it more complicated to perform any packet processing tasks. Packets can be processed at multiple stages on their way from the physical network interface until they reach the application service. However, high-performance packet processing in software requires very strict limits on the time spent in processing each packet, which has led to the idea of moving this tasks to lower levels by leveraging hardware support. The overhead associated to the software and virtualisation layers is imperative to address a synergy where software and hardware should work together to face the new challenges introduced by next-generation networks.

Contemporary Network Interface Cards (NICs) allow multiple receive and transmit descriptor queues (multi-queue). On reception, a NIC can forward different packets to different queues to distribute processing among CPUs, thus increasing performance uniformly and applying traffic prioritization when necessary. This mechanism is known as "Receive-side Scaling" (RSS). However, the filter used in RSS is typically a hash function over the network and/or transport layer headers, which is by far not enough to tackle the high level of network encapsulation imposed by virtualised networks. Therefore, network packets reaching any physical host of the infrastructure will be presented as only one flow, making network device drivers not capable of handling the efficient distribution across multiple CPUs. The lack of existing implementations capable of addressing the complexity of network traffic flowing through virtualised multi-tenant architectures has an immediate consequence in degrading the performance of the entire mobile network platform, invalidating any further optimization attempts in later stages. Furthermore, it is paramount to any service provider to guarantee their SLAs specially for critical transmissions, meaning that each service flow requires to be optimally accommodated and prioritized when required.

Latest advances in data-plane processing technologies have the potential for a breakthrough in this direction. This study shows how the extended Berkeley Packet Filter (eBPF) and the eXpress Data Path (XDP) can be explored to significantly upgrade conventional implementations and make it possible to realize high-speed custom packet processing that integrates seamlessly with existing systems, while selectively tailoring network functions in a flexible way. Specifically, this work leverages these technologies with the aim of accelerating next-generation network processing tasks, ensuring that a target system uses all its resources to their maximum capacity and avoiding packet loss by forwarding network traffic to other systems or nodes when the local system is overloaded. The proposed solution is expected to be applicable to 5G and beyond networks especially the emerging pre-6G networks. It is noted that the various network segments along the end-to-end data plane in those next-generation networks can benefit from the proposed technologies, including all the non-radio-access-network segments such as the MEC segment, the transport network and the core network. In addition, the proposed technologies can operate and are fully compatible with the so-called Fixed 5G and Beyond Networks [4].

This paper contributes to the literature by filling the gap of existing packet-processing implementations not being capable to deal with the complexity of network traffic flowing through virtualised multi-tenant architectures and addresses the following challenges: (A) To create an environment to effectively manage the execution of tailored eBPF programs directly offloaded in the hardware, before the kernel itself touches the packet data; (B) To enable packet processing to differentiate among services of various priorities, including high-priority traffic that needs to be handled as fast as possible, and non-prioritised traffic that can be forwarded when necessary to other server nodes for further processing; (C) To increase the overall system performance by distributing tasks among CPUs; (D) To design a framework to act as a glue for all interested stakeholders or higher layers of next-generation architectures, including vertical NetApp Management Platforms; and (E) To provide results of the achievable performance, throughput and packet loss of the network compared to traditional implementations.

The solution proposed envisions a decentralised management of dispatching network traffic differentiation policies that can be highly useful for network slicing implementations. Furthermore, an intensive empirical validation has been performed to demonstrate the overhead, performance, viability and scalability of the proposed framework.

The rest of the paper is organised as follows. Section 2 reviews the state of the art of existing software and hardware network data paths technologies. Section 3 outlines the eBPF infrastructure. Section 4 attempts to define the fundamentals of pre-6G networks and presents a proposed MEC architecture. Section 5 presents the architecture, lifecycle management, other technical aspects and implementation details of the proposed network accelerator framework. Section 6 depicts a holistic view of the deployment and testbed implemented. This section also presents performance and scalability test results to illustrate the findings. Finally, Sect. 7 summarises the paper and outlines future research work.

2 Literature Review

Achieving high throughput in packet processing is a key element in networks. Packets can be processed at different levels on their way from the physical network interface until they reach the application. This section reviews the existing approaches and techniques and highlights key technologies relevant to this work.

2.1 Kernel Bypass Solutions

Programmable packet processing plays a key role to provide the capability to enable the new business models expected in the pre-6G era. To avoid expensive context switches between kernel and user space, there is increasing popularity of special-purpose toolkits such as DPDK [5] for software packet processing, where a user-space application takes the complete control of the networking hardware with the while operating system (OS) bypassed. Similarly, there are also other frameworks such as Netmap [6] and PF_RING [7], which offer high packet processing performance without bypassing the kernel completely, whilst partially aiming to lower the overhead of transporting packet data from the network device to a user-space application. While these approaches can significantly increase performance, they lack of easy integration with the existing system since applications have to re-implement functionality, otherwise exposed by the OS network stack. Instead, this paper focuses on providing a solution that works in conjunction with the kernel networking stack, thereby taking advantage of eBPF-XDP-based technologies, as these are part of the mainline Linux kernel.

2.2 Software-Based Approaches

If no kernel bypass is performed, the driver builds new socket buffer (*sk_buffer*) instances and packets are sent to the network stack. In the first level of this network stack, the Traffic Control (TC) [8] is placed. TC is a packet scheduler that provides hooking points in the Linux kernel and allows configuring different queues disciplines for scheduling and shaping purposes. Apart from built-in packets processing and classification capabilities, TC can also use u32 modules for classifying and allows the attachment of eBPF programs. After TC, packets reach Linux bridges or virtual switches for further processing. One of the most commonly used tools at the bridging level is Open vSwitch (OVS) [9]. OVS is an open-source virtual multi-layer distributed switch integrated in the mainstream in the Linux kernel, which provides a switching stack for virtualised hardware environments. After this, packets that need to be processed at the network level will traverse multiple Netfilter hooks. Here, diverse packet filtering and mangling expressions can be enforced by using the well-known user-space tool iptables [10]. Iptables provides excellent performance for traditional networks especially for firewalling purposes. However, it does not scale for high traffic rates and lacks native support for tackling deeply encapsulated traffic in 5G as well as in the new generation of mobile networks as expected.

Through the literature review, there are manifold studies that approach software-level solutions to address this topic. In [11], an efficient virtual video-optimization mechanism which maintains QoS in critical services is proposed. This paper uses iptables to control and optimise video traffic in multi-tenant 5G networks. Similarly, Matencio-Escolar et al. [12] presents an adaptive network slicing solution for multi-tenant 5G IoT network. The implementation is based on OVS and provides good performance for different IoT scenarios. Meanwhile, the solution is also limited by the Linux kernel stack performance, which does not scale up for more than 1-2M packets per second. Kurtz et al. [13] propose and implement an SDN/NFV network slicing tool for 5G networks. They utilise OVS and SDN controllers to control the network traffic. However, the data path they propose does not fulfill the 5G requirements in terms of virtualisation; therefore, they do not use real 5G traffic for the experimentation. In [14], an algorithm to provide information regarding the paths traversed by the network traffic in a multi-tenant architecture is presented. This implementation uses P4 language to create a flexible and programmable network data path that allows its scaling using different network topologies. Although it provides a promising network algorithm with strong impact in the creation of multi-tenant architectures, the data presented is based on simulations without performance evaluation with real network traffic. Despite there are manifold software solutions in the related work, in this particular area, any software approach may come too late because any degradation in network traffic performance may have already occurred.

2.3 Hardware-Based Programmability

Programmable hardware presents multiple advantages and disadvantages in comparison with software-based solutions. First, the use of Content-Addressable Memory (CAM) provides constant access to memory and therefore, the number of rules inserted does not necessarily implies more memory access time. Second, hardware devices provide superior processing capabilities and thus, compared with software implementations, higher performance and bandwidths. Nonetheless, these devices provide less flexibility in terms of programmability and also the commercial cost is usually higher.

Intel has been producing in recent years several NICs with novel firewalling and QoS capabilities [15–18]. The most advanced [18] provides a QoS-aware mechanism based on a sophisticated scheduling algorithm composed of two different levels of queues. However, this card only allows data paths based on the 5-tuple (Source IP, Destination IP, Source Port, Destination Port, L4 protocol) filtering and Virtual LAN (VLAN) classification. Moreover, the network rules granularity provided is too limited, which is a strong limitation to perform the Deep Packet Inspection (DPI) in the virtualised 5G/6G networks where transmissions are deeply encapsulated.

Therefore, the market is moving from conventional NICs to NICs based on Field-Programmable Gate Arrays (FPGAs) and Network Processing Units (NPUs). These technologies overcome the lack of flexibility and programmability of the network data path in traditional NICs and they also reduce the overheads induced by the network data processing in software implementations. FPGAs and NPUs are commonly called as SmartNICs. They reduce CPU cycles, and save CPU cores and power, by offloading CPU-intensive tasks to dedicated hardware. Furthermore, SmartNICs provide the possibility of programming complex data planes and highly granular network rules, both required in the next generation of mobile networks.

The FPGA [19] launched by Intel proposed open-source platform supported by DPDK and targeted for different use cases, such as cyber security or Network Functions Virtualisation (NFV). The programmable data path provided by this card is based on Verilog [20] and VHDL [21]. Bittware [22] presents another FPGA-based solution with P4 language support, which provides high flexibility and easy programmability of the network data plane. However, no framework is provided to control the creation of new rules or hardware-based filters. The Netcope FPGA [23], although providing good performance, does not provide open source modules and therefore it is very difficult to extend the native capabilities of the card beyond the existing programmability of the data path. NetFPGA-SUME [24] is an open source FPGA, which provides a programmable network data path with P4 support. This card provides high flexibility and has an extensible framework that allows the control and definition of the tables content. However, it has been demonstrated [25, 26] that it does not provide superior performance in high-rate network transmissions.

Ricart-Sanchez et al. [27, 28] propose a NetFPGA-based network slicing solution implemented in P4 language [29] for 5G MEC architectures. Moreover, an analysis and evaluation of the performance is presented. However, these empirical validations do not present any detailed evaluation about the queues or CPUs performance. In [30], Yan et al. propose a SmartNIC-based implementation to enable network slicing for scalable cloud systems, with the objective of meeting 5G/B5G network requirements. This solution also employs P4 language to program the network data path for L2/L3/L4 classification and action. Although P4 is very flexible and it is supported by several network technologies, the range of actions provided is limited and difficult to extend in comparison with other data plane development technologies, such as eBPF or XDP. P4 language presents some limitations [31]: it does not support internal methods, P4 only support external functions or methods which are implemented outside P4 and they are called from the P4 pipeline; there is no iterations, loops are not supported; there is no dynamic memory allocation; there are no pointers or references; P4 has no built-in support for scheduling, multiplexing or queuing; there is no standard communication channel between data plane and control plane, this is usually provided by external methods; among others. Although these are promising solutions, they do not expose any accessible API through which the network policy rules can be dynamically enforced.

2.4 XDP Approaches

The improvement of the packet processing performance is a key enabling element for the deployment of virtualised pre-6G architectures, where a high performance is required. eBPF and XDP enable the implementation of high-performance networking applications based on Linux kernel and hardware offloading. In [32], an eBPF-based prototype using bpf-iptables is proposed. It presents an empirical comparison between the eBPF-based solution and the current implementation of iptables, showing improved performance especially when a high number of rules are involved. Scholz et al. [33] focus on the study of two different eBPF XDP scenarios, based on the Linux space and the application layer respectively. Nevertheless, none of the previous work has been implemented or tested in hardware offloading mode. Enberg et al. [34] propose a combined application and hardware packet steering implementation using eBPF and XDP. It provides a practical approach for accelerating network-intensive applications. Although it presents an extended description of different XDP scenarios and modes, it does not provide any empirical validation on packet processing performance. In [35] a hybrid DDoS mitigation pipeline architecture is proposed, leveraging the flexibility of eBPF and XDP to handle different types of traffic and attackers. Although achieving a dropping rate of approximately 15 Mpps in the SmartNIC CPU, it does not present packet processing scalability when network traffic is sent to other network applications. A design, prototyping and empirical validation of a network slicing approach based on eBPF, XDP and Netronome over a pre-6G infrastructure is proposed in [36]. Although this is a promising approach to provide hardware-based network slicing, they do not define any congestion control mechanism of the network node and therefore, it becomes a non suitable solution for scenarios with high data transmissions. Furthermore, none of the solution has provided a classification mechanism to identity and take decisions over virtualised multi-tenant network traffic. Despite the considerable number of related work that exist in eBPF and XDP technologies, they are mostly to introduce the concept to the research community by demonstrating promising performance results.

However, there has not yet been sufficient research from the virtualised infrastructure data path perspective on how to: (a) Satisfy the performance demands of use cases with diverging requirements and prioritisation levels; (b) Analyse the system status to avoid packet losses by proposing a reactive solution.

3 Background of the Proposed Approach

This section provides brief and essential background information on eBPF together with XDP to facilitate the understanding of the proposed approach, where the combination of eBPF and XDP serves as the basis. Full details of the related background can be found in [37–39].

eBPF enables programmed code to be executed in the kernel space in a more secure and restricted environment, which allows creating tools that otherwise would require modifying kernel's source code or implementing new kernel modules. eBPF employs a highly flexible and efficient virtual machine (VM) construct in the Linux kernel to execute bytecode at multiple hook points safely. Thanks to those hooks including XDP hooks, eBPF programs can be designed for manifold use cases, most prominently networking, tracing and security. eBPF programs are written in *restricted* C code and compiled to eBPF bytecode, which is injected from the user space into the kernel, where it is verified before attached. eBPF is able to call a fixed set of in-kernel helper functions (via BPF_CALL) and access shared data structures such as eBPF maps, which act as efficient key/value stores. It offers helper functions to communicate with and to take advantage of the kernel functionality tail calls to interact with other eBPF programs, security capabilities, object pinning (maps, programs), and infrastructure for allowing eBPF to be offloaded to Smart Network Cards (SmartNICs). In the following, eBPF VM, XDP hooks and eBPF maps are outlined.

3.1 eBPF Virtual Machine

eBPF in-kernel VM allows injecting and executing programs from the user space by attaching them to specific hooks. These programs run in a restricted sandbox environment with access only to a limited set of functions. The VM consists of 11 64-bit registers, a program counter and a 512 byte fixed-size stack. Registers are named *r0–r10*.

- *r0* Contains the return value of a helper function call.
- *r1–r5* Hold arguments from the BPF program to the kernel helper function.
- *r6–r9* Are called saved registers that will be preserved on helper function call.
- *r10* Read-only frame pointer to access stack.

Some SmartNICs have already mapped this well constrained VM for offloading to lightweight Network Processing Unit (NPU) general purpose cores [40], whilst this paper leverages these capabilities for the kernel to execute eBPF programs on the network interface instead of on the host CPU.

3.2 XDP Hooks

eBPF employs a number of hooks for attaching programs, including those concerned at the lower end of the datapath. From Linux 4.8 +, new hooks have been added for XDP, a new programmable high-performance networking datapath that works in conjunction with the Linux stack, and relies on eBPF to perform very fast packet processing. The key difference is that XDP hooks allow executing programs to process packets at very early stages, before they arrive at the Linux network stack. The execution can happen in three different places (1) **Generic Mode** where the XDP hook is called from *netif_ receive_ skb()*, after the packet Direct Memory Allocation (DMA) and Socket Buffer (SKB) allocation are finished, thereby loosing most of the performance benefits; (2) **Native Mode** where the execution takes place in the driver before the kernel allocates an SKB; (3) **Offloaded Mode** which attaches eBPF programs into hardware network devices, and thus is the fastest mode of all. This paper focuses on the Offloaded Mode to offload eBPF program to the network card itself.

A valid eBPF program attached to a XDP hook must return a *xdp_action* indicating the decision on what to do with the packet after it has been processed. Available values, which are defined in *bpf.h*, are listed below:

- *XDP_ABORTED* Error, Drop packet.
- *XDP_DROP* Drop packet.
- *XDP_PASS* Allow further processing by kernel stack.
- *XDP_TX* Transmit from the interface the packet came from.
- *XDP_REDIRECT* Transmit the packet from another interface.

As further explained in Sect. 5.4, this paper uses a combination of *XDP_PASS* and *XDP_TX* to both send priority traffic up to the kernel network stack for regular processing, and to forward non-prioritised network traffic to other available physical hosts in the infrastructure.

3.3 eBPF Maps

eBPF utilises maps as generic key/value data structure for data transfer between Kernel/hardware and user space. The maps are managed by using file descriptor and they are accessed from user space via BPF *syscalls*. eBPF provides multiple useful data structures that we can explore to store persisted data or even to exchange data from/to the user space. Each map type has a distinct functionality, with some being used globally and others having specific applications. Although a full list is defined in the *enum bpf_map_type*, from */usr/include/linux/bpf.h*, some of the most important are listed below:

- *BPF_MAP_TYPE_HASH* A map with items indexed by a hash function.
- *BPF_MAP_TYPE_ARRAY* A map with items indexed by a number.
- *BPF_MAP_TYPE_PROG_ARRAY* A map that contains references to other eBPF programs.
- *BPF_MAP_TYPE_SOCKMAP* A map with socket references.

- *BPF_MAP_TYPE_CPUMAP* A map that can redirect raw XDP frames to another CPU.
- *BPF_MAP_TYPE_XSKMAP* A map that can redirect XDP frames to an AF_XDP socket.
- *BPF_MAP_TYPE_QUEUE* A map with a queue behaviour.
- *BPF_MAP_TYPE_STACK* A map that uses last-in, first-out (LIFO) to keep elements in the map.

As further explained in Sect. 5.3, this paper explores two different types of maps for sharing data between eBPF offloaded programs and user-space applications. Specifically *BPF_MAP_TYPE_HASH* and *BPF_MAP_TYPE_ARRAY* are employed to store packet data structures of priority traffic and a list of alternative server addresses, respectively.

4 Pre-6G Network Architecture

Whilst the research into the next-generation networks towards 6G is emerging, there is no official architecture proposal yet. Figure 1 envisions a pre-6G MEC architecture, which attempts to present a view of a pre-6G architecture, following an evolution of the MEC paradigm. MEC moves part of the service processing and data storage from the core of the network (central cloud) to the edge nodes. This physical and logical movement of the services to the last miles implies several benefits already seen in current 5G networks, such as performance improvements and traffic optimisation. In pre-6G networks, it is expected that intelligent edge processing will be more pervasive and powerful, being built upon 5G MEC yet significantly enhance and extend the benefits, empowered by advanced traffic control and Artificial Intelligence (AI) techniques at the edge, beyond the advances in 5G networks. This paper

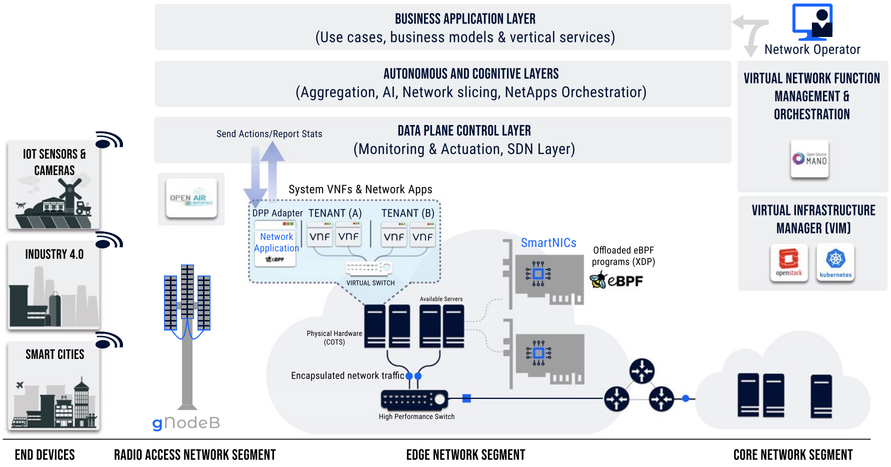

BUSINESS APPLICATION LAYER
(Use cases, business models & vertical services)
Network Operator
AUTONOMOUS AND COGNITIVE LAYERS
(Aggregation, AI, Network slicing, NetApps Orchestratior)
VIRTUAL NETWORK FUNCTION MANAGEMENT & ORCHESTRATION
MANO
Send Actions/Report Stats
DATA PLANE CONTROL LAYER
(Monitoring & Actuation, SDN Layer)
IOT SENSORS & CAMERAS
INDUSTRY 4.0
SMART CITIES
OPEN AIR
System VNFs & Network Apps
DPP Adapter
TENANT (A)
TENANT (B)
Network Application
eBPF
vnf
vnf
vnf
vnf
VIRTUAL SWITCH
SmartNICs
Offloaded eBPF programs (XDP)
eBPF
VIRTUAL INFRASTRUCTURE MANAGER (VIM)
openstack
kubernetes
Available Servers
Physical Hardware (COTS)
Encapsulated network traffic
High Performance Switch
gNodeB
END DEVICES
RADIO ACCESS NETWORK SEGMENT
EDGE NETWORK SEGMENT
CORE NETWORK SEGMENT

Fig. 1 Pre-6G architecture overview

focuses on advanced edge traffic control. Moreover, such improved MEC will play an increasingly important role to support new functions and services. Beyond the 5G use case requirements in terms of eMBB (enhanced Mobile Broadband), URLLC (Ultra Reliable Low Latency Communications) and mMTC (massive Machine Type Communications), it is expected that pre-6G networks would allow a flexible combination and expansion of the requirements in 5G to provide improved support for complex use cases such as Industry 4.0+ and Factories and Cities of Future [41].

To this end, the Autonomous and Cognitive layers provide intelligent network management, comprising aggregation, AI, network slicing and NetApps Orchestration modules [42]. Intelligent network management intents are then transformed into actions by the Data Plane Control layer and inserted into the network applications. These actions are enforced to control the network traffic in the data plane and thus to guarantee the service-level agreements to fulfil the requirements of the various demanding use cases, and new business models and vertical services at the Business Application layer.

The infrastructure presented in Fig. 1 consists of three main network segments, the Radio Access Network (RAN), the Edge and the Core. These segments are distributed in different geographical locations, managed by virtual infrastructure managers (VIMs), e.g., using OpenStack [43] and Kubernetes [44]. The RAN segment is composed of gNodeBs (gNBs), which represent the logical radio nodes and allow the wireless communications between the end devices and the RAN. For a proof-of-concept implementation, these gNBs can be built upon the OpenAirInterface (OAI) platform [45] as part of the pre-6G architecture using Ettus B210 [46] software-defined radio. In the edge segment of the network, Commercial Off-The-Shelf (COTS) hardware is used to deployed logical services using Network Function Virtualisation (NFV) technologies, which allow the massive deployment of virtualised network applications closer to final users. These COTS computers also provide offloading mechanisms, such as XDP and eBPF programs, employed to execute network functions of using the hardware directly. Virtualisation technologies imply a considerable reduction in the Capital Expenditure (CAPEX) for mobile network operators, while offloading tools also provide the mechanisms to achieve the demanding performance required by the next-generation services. The different edges of the network are connected to the core network segment, where the data centre of the architecture with different network functions are deployed. These functions are responsible for mobility and session management, user authentication, authorisation and accounting, among others.

4.1 Network Traffic in Virtualised Architectures

Pre-6G is a major evolution over previous technologies in many senses. Apart from introducing performance improvements of orders of magnitude over today's networks, 6G infrastructures provide native support for multi-tenancy, mobility and knowledge discovery, automatic network adjustment, smart resource management and intelligent service provisioning. In order to provide these features, Pre-6G data packets follow a nested structure, which is illustrated as it follows: (1) MAC/IP/L4, (2) VXLAN/MAC/IP/L4 and (3) GTP/IP/L4/SERVICE.

The first group of headers **(1) MAC/IP/L4** is related to the communication between physical machines including Medium Access Control (MAC), IP and the transport protocol (TCP or UDP). The second group **(2) VXLAN/MAC/IP/L4** includes a VXLAN encapsulation protocol, MAC, IP and the transport layer (TCP or UDP). This second first encapsulation layer is used to isolate tenant traffic, especially for mobile network operators sharing the same physical 5G infrastructure as tenants. The group three of headers, **(3) GTP/IP/L4/SERVICE**, includes GTP, IP and TCP or UDP, and it is introduced to allow the management of the user mobility across different gNB in a transparent way without losing connectivity. Finally, the application header contains the data being transmitted by the end users, for example RTP for video communications or HTTP for web services. The VXLAN encapsulation protocol is used to achieve tenant isolation, however other alternative protocols like GRE or GENEVE can be employed to fit the same purpose. Normal IP network packets uses a very limited subset of these headers, for instance, MAC/IP/UDP/SERVICE, nonetheless, compared to this simple case, several additional headers have been added to achieve both multi-tenancy and mobility. These encapsulation protocols are applied by both ends of the data path, i.e. edge and core.

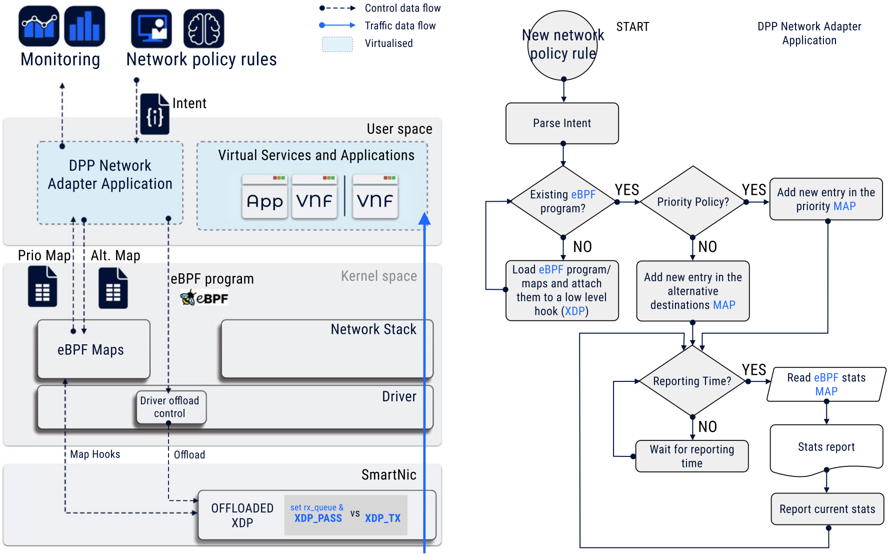
The parsing and classification of this complex network packages is one of the problems that current NICs face. Commercial-off-the-shelf (COTS) NICs do not provide classification and control support for the double encapsulated network traffic required for most of the novel 5G/6G network architectures. It can lead to an issue when the final user information is contained in the nested headers and packet steering protocol is based on this information. In this context, when COTS NICs receive 5G/6G network data flows they are not able to inspect inner headers and the packet steering is made based on the outer headers information. This can end in a CPU bottleneck, since the destination queue and CPU decision is based on the physical machine, which is static for our Pre-6G architecture proposed and not based on the final user information. However, these final user addresses are totally dynamic, based on the flow, and will vary depending on the final service. The SmartNIC-based solution proposed in this paper supports this complex data path, as well as to offload the network functionality using eBPF and XDP. The flexibility and high performance provided by SmartNICs is implying a needed change from traditional NICs to SmartNICs in the new generation of mobile networks.

4.2 Focused Use Case Scenario

Future 6G networks are expected to process 10 to 100 times more data volume than their predecessors [47]. However, more and more, it is the network end system, instead of the network, that is responsible for degraded performance of network applications. The proposed use case presents a scenario where some services have been migrated/deployed at the edge, closer to the final user, to improve the overall services performance on demand. However, a situation with an ultra-high rate of data traffic passing through a single edge node can lead to performance bottleneck e.g., causing massive packet loss. This paper addresses this problem by proposing a dynamic multi-queue discipline mechanism that avoids the congestion of the edge node by distributing traffic processing among different CPUs. Although this multi-queue discipline allows prioritisation of the traffic and offers better performance in terms of throughput, the Linux Network Stack still presents some limitations in terms of scalability when high volumes of data need to be processed. To tackle this, this paper also proposes the forwarding of non-prioritised packets to a different edge node or even to the core segment of the network for further processing, when the receiving system exceeds its processing capacity. This allow flexible balancing the network traffic between different nodes and therefore improve the overall performance of the network.

5 The Proposed Framework

This section provides more details on the proposed pre-6G architecture and an implementation of a network accelerator framework towards realising the architecture. Figure 2a shows the system diagram to establish the position, connections, and different roles of the framework components.



(a) Proposed Framework. **(b)** DPPnApp Lifecycle overview.

Fig. 2 **a** Framework diagram to establish the position, connections, and different roles among the different modules; **b** Lifecycle overview of Data Plane Programmability Network Application containing the logic to offload eBPF programs into SmartNICs and access eBPF maps to both add new entries, and collect stats

```
{
"Resources": [{
  "resourceId": "F8A4C949",
  "encapsulationID1": "00000445",
  "encapsulationType1": "vxlan"
},{
  "resourceId": "B6E6B2B3",
  "encapsulationID2": "8894D0D4",
  "encapsulationType2": "gtp"
},{
  "resourceId": "5A07C580",
  "srcIP": "192.188.0.140",
  "dstPort": "5004",
}],
"Action": {
    "actionType": "INSERT",
    "actionName": "PRIO_QUEUE",
    "flow_hash": "E918D704",
    "rx_queue": "1"},
"Params": [{
    "paramName":  "interfaceName",
    "paramValue": "eth0"}]}
```

Listing 1 Intent-based message

5.1 Control Layer

On top of Fig. 2a it shows how network policy rules can be enforced through the Northbound Interface (NBI) of the Data Plane Programmability Network Adapter Application (DPPnApp). This enables any authorised entity of higher layers of the described pre-6G architecture to send intent-based messages to perform specific network actions over the system. An *Intent* defines what type of traffic should be controlled and the action that needs to be enforced over packets matching its network specifications. As it is represented in Listing 1 Intent-based messages reaching this NBI are composed of one or more *resources* describing the flows to take action to, one *action* that defines which kind of action has to be applied over such flows (e.g. send to rx_queue), and a list of *parameters* for fine-grained specifications (e.g, network interface, mode, time, duration, etc.). This gives enough information for the DPPnApp to parse a given intent parameters into a *C* structure which will later be added as a new entry on the eBPF-MAP (see Sect. 3.3). Listing 1 gives a simple example of an intent-based message where a specific flow service (identified by its *resourceId*) needs to be prioritised. For classifying tasks, the packet structure (network protocol stack) is provided in the *Resources* section. In this example, the flow service is encapsulated in two different network protocols, VXLAN and GTP , which will need to be added as part of the filter in the network policy rule.

In a similar manner, higher layers of the pre-6G architecture can also monitor the performance and implications of existing network policy rules in the system (e.g, throughput, packet loss, cpu-usage, etc.). This offers fine-grained control of the platform and paves the way for network operators or even AI-based models to update,

```
struct prio_value {
  uint8_t   is_active;
  uint32_t  target_rx_queue_index;
  struct    flow_stats  f_stats;
  struct    flow_struct  f_struct;
};
```

Listing 2 Data structure used as value for the key/value *BPF_MAP_TYPE_HASH* prio-MAP

remove or create new network policy rules based on the current behaviour of the overall system or a particular service.

5.2 DPPnApp: Data Plane Programmability Network Adapter Application

The DPPnApp is the core control module. Its NBI exposes functions to both enforce and monitor network policy rules. DPPnApp provisions the system by offloading maps and programs in its first execution, parses intent-based messages and inserts them as new entries of the eBPF maps, reads from those maps to generate statistical reports, and provides feedback to the higher layers of the architecture by sending periodical informs. Figure 2b depicts the logical representation of the DPPnAPP's lifecycle flow, which provides desegregated policy management and statistics. The lifecycle starts when a network policy is received by the intent parser, transforms it into eBPF rules and inserts them into the eBPF program maps. Statistics about the rules inserted and traffic processed are collected and stored in different eBPF maps. These statistics are periodically collected from the maps in order to generate statistics reports used for further analysis of the eBPF-XDP framework deployed.

5.3 eBPF Maps Framework Requirements

As displayed in Fig. 2a, eBPF-maps are used for the communication (by sharing data) of the user space application with the eBPF program that is offloaded in the SmartNIC. This paper envisions two different types of maps. The first eBPF-map (Prio-Map in Fig. 2a) is a *BPF_MAP_TYPE_HASH* and contains all the required information of the network traffic that has to be accepted in the system. The second eBPF-map (Alt-Map in Fig. 2a) is a *BPF_MAP_TYPE_ARRAY* and contains a list of alternative server addresses to which to forward traffic that has not been accepted by the system (non-prioritised traffic). When a new network policy rule is enforced, the DPPnApp will parse and accommodate the intent-based message as a new entry of a eBPF-map structure. By doing so, the eBPF program can look up for new entries on such maps and perform actions to every packet matching their network protocol specifications.

Listing 2 shows the data structure used as value for any (key/value) entry of the prio-MAP. It is worth highlighting that the *key* would be the flow hash that is passed in the Intent message. The attribute *is_active* lets the eBPF program know that there is a pending action to be performed, and it lets the user space application know that this rule has been performed successfully and is ready to provide statistical data. *target_rx_queue_index* indicates to which rx_queue packets matching the network specifications have to be sent. *f_stats* is a structure that contains up-to-date statistics (number of packets, bytes, packet size, etc.) of the network policy rule. Finally, *f_struct* is also a structure that represents an abstraction of the Intent message (Resources in 1) defining the network protocol fields that traffic reaching the network interface must accomplish to be part of this flow.

5.4 Offloaded eBPF Program Details

Algorithm 1 shows the eBPF program implemented, using the constraint C language, which allows packet classification, processing and accelerating tunnelled network traffic in the architecture presented. This algorithm also guarantees a fair distribution of the tunnelled traffic between the different RSS queues allocated in the kernel space and controlled by the Netronome Flow Processor (NFP) driver. From the point of view of the NIC, when a packet arrives, the eBPF-program starts by parsing packet headers to extract the metadata information it will react on (see lines 3 and 4). It then generates a hash by using an adapted version of the */include/linux/jhash.h* library, which is distributed with the Linux kernel source code; the main difference here is that the proposed packet parser also covers network tunnelling protocols required for 5G and pre-6G network traffic, and for this reason the *jhash* library needs to be adapted to include more entries in its hash generator function (see line 5). Next, the eBPF program reads from the prio-MAP to look up for the key with the same hash value (see line 6). If successful, and the associated statistics do not indicate an overloaded system, it extracts the *target_rx_queue_index* attribute its map entry and rewrites the current packet metadata with that value (see lines 7 and 8). After the packet is changed, the final verdict is given in the form of a eBPF program return code (see Sect. 3.2). In this case, the return code is XDP_PASS that will allow packets to be sent to the kernel network stack for regular processing (see line 13). If the hash value does not match with any prio-MAP entry and congestion is detected in the system, the eBPF program will look up of the Alt-MAP to select an alternative destination where non-prioritised traffic can be processed (see line 10). The eBPF program will change the MAC and IP destination addresses of such a packet, and return a XDP_TX code to send it back to the same interface (see lines 11 and 12). The MAC address value retrieved from the Alt-MAP represents the direction of the alternative network node that is used to process the network packet.

This behaviour aims to increase performance uniformly by forwarding different packets to different queues to distribute processing among CPUs. It also prevents the system from being overloaded by forwarding non-critical traffic to alternative destinations for further processing.

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

```
struct pkt_meta;
procedure XDP_PROG(pkt)
    parse_headers(pkt);
    pkt_meta ← pkt;
    hash ← calculate_hash(pkt_meta);
    prio_entry ← prio_map.lookup(hash);
    if prio_entry != NULL && !cpu_overloaded() then
        pkt.rx_queue ← prio_entry.target_rx_queue;
    else
        alt_entry ← alt_map.lookup(hash);
        update_dst_address(alt_entry.dst_address)
        return XDP_TX;
    end if
    return XDP_PASS;
end procedure
```

6 Empirical Performance Evaluation

6.1 Experimental Design and Implementation

Figure 3 shows a holistic view of our experimental deployment. The deployment used a Dell T5810 machine equipped with an Intel Xeon CPU E5-2630 v4, 32 768 MB of RAM, and 512-GB solid-state drive (SSD) with support of hardware Direct Memory Access (DMA) system to place packet data directly in the CPU cache. The test machine is equipped with two Netronome cards Agilio CX 2x25GbE network adapters, which are supported by the *NFP* driver. These cards have one single port each, connected to a 25G SPF28 cable. They are depicts in in Fig. 3 as Netronome 1 with port A associated and Netronome 2 with port B. The test computer runs Linux kernel version 5.4 supporting XDP for Netronome NICs, with the *hyperthreading* disabled and *irq_affinity* adjusted to match one queue per CPU. Each CPU is in C-State C0, meaning that they are fully turned on. For traffic generation, Pktgen was employed. With a total of 10 CPUs, for all the experiments, two CPUs (8–9) were reserved for the traffic generator while the remaining (0–7) were attached to the network receiving queues. For the CPU usage, the *mpstat* system utility was utilised; in addition, the */proc/net/softnet_stat* file is periodically inspected to monitor stats from network devices.

The traffic generated by pktgen is transmitted through Netronome 1 using port A and it is received in Netronome 2 through port B. Once the network traffic is received in Netronome 2, the eBPF programm determines the action to apply over

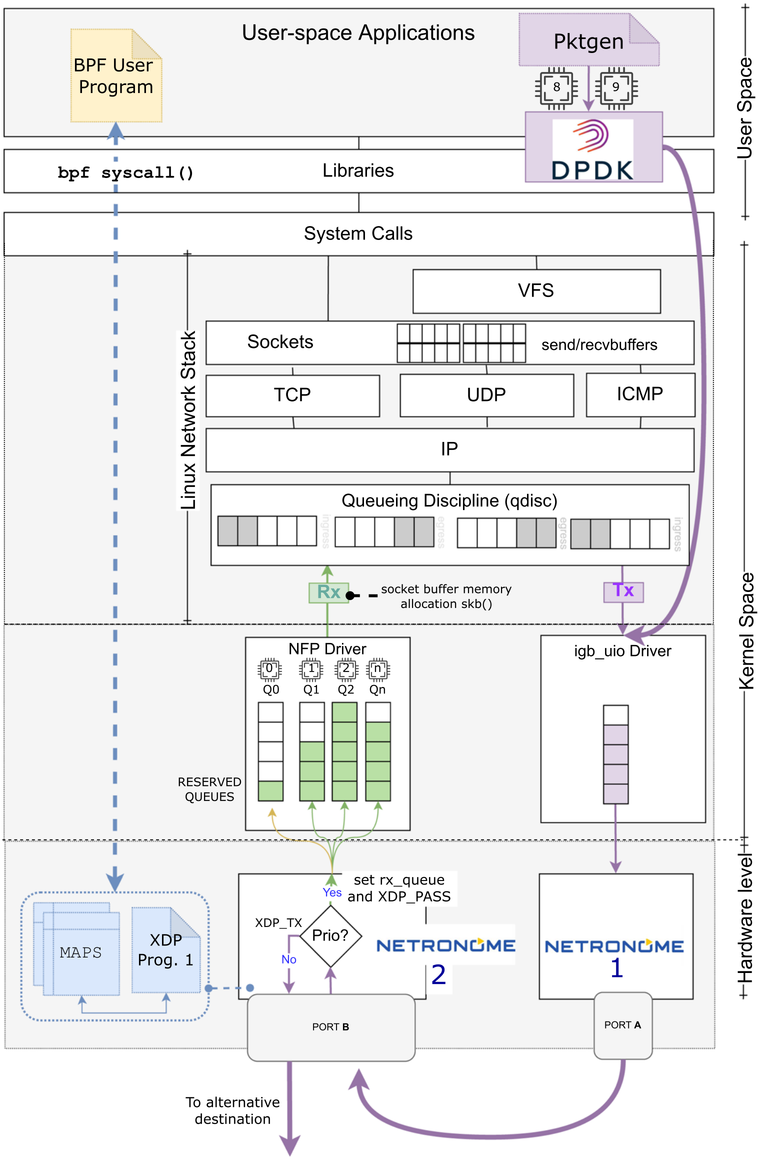


Fig. 3 Implementation of the performance evaluation testbed

this traffic. The solution implemented provides two operational modes depending on the XDP action invoked in the network data plane. These actions, *XDP_PASS and XDP_TX*, define, respectively, if the packet is sent to the kernel of the system or if it is forwarded to a different physical machine, load balancing the traffic. Therefore, when XDP_PASS is invoked, Netronome 2 Port B actuates only as a reception interface (Rx), while, when XDP_TX is invoked this Netronome card actuates as both, a reception (Rx) and a transmission (Tx) interface.

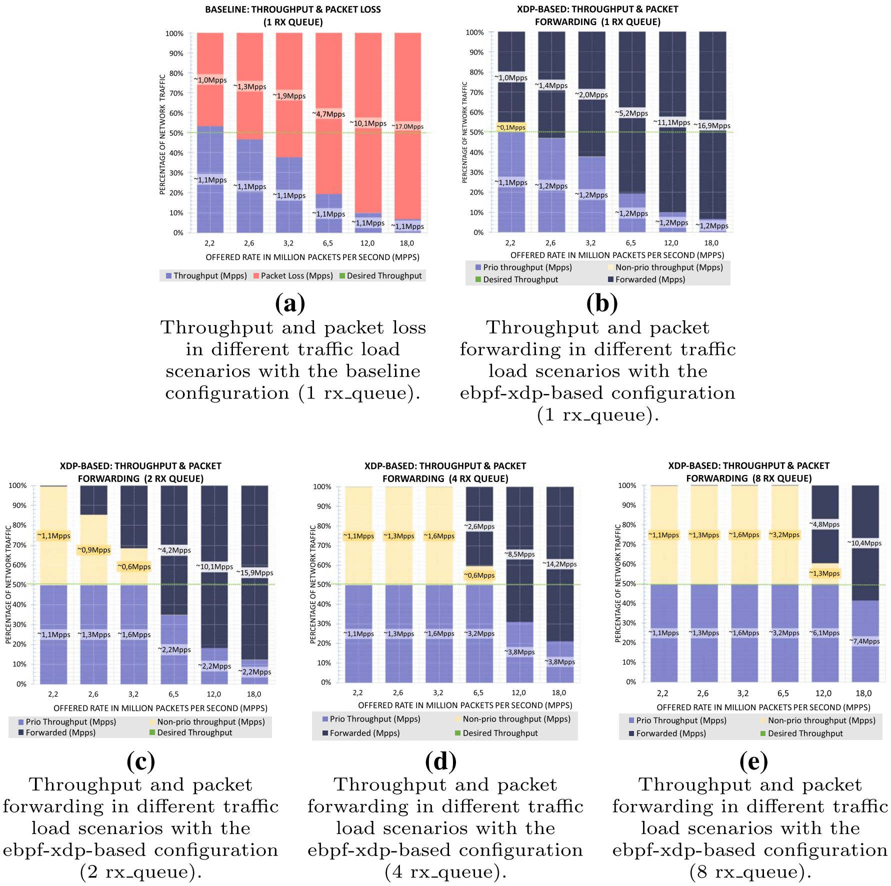
6.2 Empirical Evaluation

This section presents the empirical performance evaluation of the proposed solution. A set of experiments has been conducted in order to validate its functionality and assess its performance. Table 1 shows the different experiments conducted. Packets of multiple sizes have been generated to provide different data plane traffic loads; for each of these scenarios, an increasing number of receiving queues were implemented to evaluate the system behaviour. For all experiments, the traffic injected in the network had a total of 16 services of which 8 were identified as prioritised services. The prioritised services are selected depending on their Tunnel endpoint identifier (TEID) of the GTP Network protocol, that allowed to emulate different mobile users that require to be prioritised. In the experiments, all services sent packets at the same packet rate. Thus, the optimal performance point was reached when half of the injected network traffic (the prioritised traffic) reaches the network stack. This value is indicated as *Desired Throughput* in Table 1 and it is also shown as a horizontal line, located at 50% value, in Fig. 4.

The performance evaluation focuses on three metrics:

- Packet loss. In contrast to many other studies on the efficiency of XDP, this paper also offers packet loss results to show the maximum throughout performance of the overall system. Although packet processing in hardware can be really fast, subsequent levels of kernel network processing tasks can present a bottleneck, causing packets to be discarded before they reach the application in the user space. This effectively measures the overhead of the system as a whole, and serves as a key indicator to assess the proposed solution under different traffic load conditions.
- CPU usage. On reception, a NIC can forward different packets to different queues to distribute processing among CPUs, thereby increasing performance uniformly. This is quantified by measuring how CPU usage scales with the different queue implementations.
- Packet forwarding performance. As mentioned in the introduction section, the objective is to prioritise traffic belonging to specific services. However, it is equally important that no traffic is lost under normal conditions. This is achieved by redirecting non-prioritised traffic to other distributed network nodes for further processing.

The following results attempt to find the optimal performance point in terms of the highest number of network packets per second that manage to be processed by the Linux network stack. To this end, hardware packet pre-processing was executed to distribute the prioritised traffic over different queues/CPUs while non-prioritised



(a) Throughput and packet loss in different traffic load scenarios with the baseline configuration (1 rx_queue).

(b) Throughput and packet forwarding in different traffic load scenarios with the ebpf-xdp-based configuration (1 rx_queue).

(c) Throughput and packet forwarding in different traffic load scenarios with the ebpf-xdp-based configuration (2 rx_queue).

(d) Throughput and packet forwarding in different traffic load scenarios with the ebpf-xdp-based configuration (4 rx_queue).

(e) Throughput and packet forwarding in different traffic load scenarios with the ebpf-xdp-based configuration (8 rx_queue).

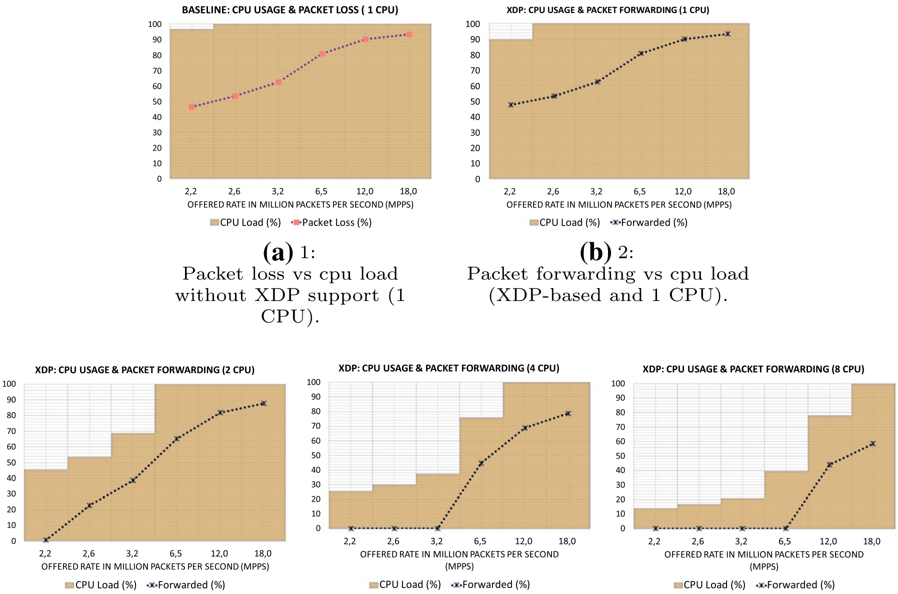
Fig. 4 Performance results in terms of accepted throughput, packet loss, and packet forwarding showing the comparison between the baseline implementation and the proposed eBPF-XDP-based approach. The optimal performance point is shown as Desired Throughput

Table 1 Performance evaluation experiments

Packet size (Bytes)	Offered load (Mpps)	Services	Prio services	Desired throughput (Mpps)
1500	≈ 2.2	16	8	≈ 1.1
1280	≈ 2.6	16	8	≈ 1.3
1024	≈ 3.2	16	8	≈ 1.6
512	≈ 6.5	16	8	≈ 3.2
256	≈ 12	16	8	≈ 6.0
132	≈ 18	16	8	≈ 9.0

traffic or packets exceeding the system capacities (overloaded CPUs) are forwarded to other systems to prevent them from being lost.

Figure 4 compares the performance between a baseline configuration and the proposed eBPF-XDP-based implementations. For all the experiments, the prioritised throughput is exactly the half of the total traffic injected, consequently the *"Desired Throughput"* is depicted as an horizontal (green) line at the 50%, meaning that once the *"Prio Throughput"* has reached this threshold, critical network traffic is guaranteed to be processed locally. The remaining throughput (non priority), can then either be accepted to be processed locally (depicted as *Non-prio throughput* in Fig. 4) or forwarded to other destinations depending on the current system capacity. It should be noted that in those scenarios in which the system cannot process all inbound prioritized traffic, this will also be forwarded. Figure 4a shows the normal behaviour of a traditional NIC attempting to process traffic from a muti-tenant virtualised mobile network. The complexity that nested encapsulation adds to packet processing tasks makes baseline NICs unable to dissect network traffic and therefore limiting the number of receive queues/CPUs to just one. This situation results in a high level of packets loss for transmissions greater than 1 Mpps (which is roughly the limit of the Linux kernel). A similar behaviour is seen in Fig. 4b. Meanwhile, although with one receiving queue configuration it is not possible to increase the accepted throughput values, XDP allows traffic that exceeds the system's capacity to be forwarded so as not to cause packet loss. The rest of the graphs show an improvement in throughput as the number of queues increases. As shown in Fig. 4e, the



(a) 1:
Packet loss vs cpu load without XDP support (1 CPU).

(b) 2:
Packet forwarding vs cpu load (XDP-based and 1 CPU).

(c) 3:
Packet forwarding vs cpu load (XDP-based and 2 CPUs).

(d) 4:
Packet forwarding vs cpu load (XDP-based and 4 CPUs).

(e) 5:
Packet forwarding vs cpu load (XDP-based and 8 CPUs).

Fig. 5 Relation between packet loss/forwarded and CPU load with different number of CPU implementations. The *irq_affinity* was 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

maximum performance point was achieved at 12 Mpps (approx. 24.5 Gbps) offered rate, where it reached the desired throughput (approx. 12.25 Gbps), meaning that all prioritised services will be handled by the host, and just the non-prioritised traffic will be forwarded.

Figure 5 provides details of the CPU usage and how it varied in relation to the different traffic load and the number of queues. Figure 5a shows a baseline configuration with just one CPU being used. Given that this implementation barely accepts a traffic load greater than 1Mpps, it can be observed that in all experiments, the CPU capacities were exceeded, which led to an increasing number of packet loss. The remaining graphs show the results based on the proposed eBPF-XDP-based approach and depict how allowing traffic steering between multiple queues alleviated CPU consumption levels. Even so, there were some cases where the number of queues was not enough to handle all the network traffic and the CPUs reached their maximum capacity. When this occurred, the percentage of forwarding traffic increases above 50% to also forward prioritised traffic before it was discarded. From these experiments it can be concluded that in the best case scenario (as shown in Fig. 5e) packet processing operations at 12Mpps with 8 CPUs working at their 80% of capacity were needed to handle all the prioritised traffic.

7 Conclusion

Next-generation networking systems such as the beyond 5G or pre-6G networks entail ultra-high packet processing capabilities for traffic engineering to meet the ever-growing performance requirements of various use cases. This paper proposes a next-generation networking platform framework for an envisioned pre-6G architecture featuring SmartNICs for advanced data plane control and hardware acceleration for significantly enhanced mobile edge computing. The proposed programmable data plane explores a hardware-offloading-based approach that combines cutting-edge technologies especially eBPF and XDP, and a new eBPF-XDP packet processing algorithm is devised accordingly. Furthermore, in the core of the control, a novel Data Plane Programmability Network Adapter Application is proposed to allow intent-based actions for monitoring and control purposes, with the full lifecycle defined. Experimental results have validated the design and implementation of the proposed solution, showing a superior packet processing capacity at 18 Mpps, system throughput up to 6.1 Mpps with no packet loss, and high flexibility of the framework to adapt to multiple network policy rules dynamically on demand.

Acknowledgements 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-SU-DS-2020/101020259 ” ARCADIAN-IoT: Autonomous Trust, Security and Privacy Management Framework for IoT.

References

1. Saad, W., Bennis, M., Chen, M.: A vision of 6g wireless systems: applications, trends, technologies, and open research problems. IEEE Netw. **34**(3), 134–142 (2019)
2. Zhang, Z., Xiao, Y., Ma, Z., Xiao, M., Ding, Z., Lei, X., Karagiannidis, G.K., Fan, P.: 6G wireless networks: vision, requirements, architecture, and key technologies. IEEE Veh. Technol. Mag. **14**(3), 28–41 (2019). https://doi.org/10.1109/MVT.2019.2921208
3. 5G-PPP: Empowering vertical industries through 5g networks—current status and future trends. https://5g-ppp.eu/wp-content/uploads/2020/09/5GPPP-VerticalsWhitePaper-2020-Final.pdf (2020)
4. Fifth Generation Fixed Network (F5G). https://www.etsi.org/technologies/fifth-generation-fixed-network-f5g. Accessed 26 Jan 2021
5. DPDK Data Plane Development Kit. http://dpdk.org/ (2021)
6. netmap - the fast packet I/O framework. http://info.iet.unipi.it/~luigi/netmap/. Accessed 3 Feb 2021
7. PF_RING: High-speed packet capture, filtering and analysis. https://www.ntop.org/products/packet-capture/pf_ring/. Accessed 2 July 2021
8. Almesberger, W., et al.: Linux network traffic control–implementation overview (1999)
9. Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., et al.: The design and implementation of open vswitch. In: 12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15), pp. 117–130 (2015)
10. Purdy, G.N.: Linux iptables Pocket Reference: Firewalls, NAT & Accounting. O'Reilly Media Inc, Sebastopol (2004)
11. Salva-Garcia, P., Calero, J.M.A., Wang, Q., Arevalillo-Herraez, M., Bernabe, J.B.: Scalable virtual network video-optimizer for adaptive real-time video transmission in 5G networks. IEEE Trans. Netw. Serv. Manage. (2020). https://doi.org/10.1109/TNSM.2020.2978975
12. Escolar, A.M., Alcaraz-Calero, J.M., Salva-Garcia, P., Bernabe, J.B., Wang, Q.: Adaptive network slicing in multi-tenant 5G IoT networks. IEEE Access **9**, 14048–14069 (2021). https://doi.org/10.1109/ACCESS.2021.3051940
13. Kurtz, F., Bektas, C., Dorsch, N., Wietfeld, C.: Network slicing for critical communications in shared 5G infrastructures-an empirical evaluation. In: 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), pp. 393–399 (2018). IEEE
14. Martins, R.F.T., da Silva Villaça, R., Verdi, F.L.: Bitmatrix: a multipurpose sketch for monitoring of multi-tenant networks. J. Netw. Syst. Manag. **28**(4), 1745–1774 (2020)
15. Intel Ethernet Converged Network Adapter X520. https://www.intel.la/content/dam/www/public/us/en/documents/product-briefs/ethernet-x520-server-adapters-brief.pdf. Accessed 19 Feb 2020
16. Intel Ethernet Converged Network Adapter X540. https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/ethernet-x540-t2-brief.pdf. Accessed 19 Feb 2020
17. Intel Ethernet Converged Network Adapter XL710. https://www.intel.la/content/dam/www/public/us/en/documents/product-briefs/ethernet-xl710-brief.pdf. Accessed 19 Feb 2020
18. Intel: Intel 82599 10 Gigabit Ethernet Controller: Datasheet. https://www.intel.la/content/www/xl/es/embedded/products/networking/82599-10-gbe-controller-datasheet.html (2016). Accessed 14 June 2018
19. Intel FPGA Programmable Acceleration Card N3000 for Networking. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/po/intel-fpga-programmable-acceleration-card-n3000-for-networking.pdf. Accessed 20 Feb 2020
20. Thomas, D., Moorby, P.: The Verilog® Hardware Description Language. Springer, New York (2008)
21. Navabi, Z.: VHDL: Analysis and Modeling of Digital Systems. McGraw-Hill Inc, New York (1997)
22. Bittware: 100G NIC application. https://www.bittware.com/fpga/smartnic/ (2017). Accessed 10 Mar 2018

23. Martinek, T., Kosek, M.: Netcope: Platform for rapid development of network applications. In: Design and Diagnostics of Electronic Circuits and Systems, 2008. DDECS 2008. 11th IEEE Workshop On, pp. 1–6 (2008). IEEE
24. Zilberman, N., Audzevich, Y., Covington, G.A., Moore, A.W.: Netfpga sume: toward 100 gbps as research commodity. IEEE Micro **34**(5), 32–41 (2014)
25. Ricart-Sanchez, R., Malagon, P., Salva-Garcia, P., Perez, E.C., Wang, Q., Calero, J.M.A.: Towards an FPGA-accelerated programmable data path for edge-to-core communications in 5G networks. J. Netw. Comput. Appl. **124**, 80–93 (2018)
26. Ricart-Sanchez, R., Malagon, P., Alcaraz Calero, J.M., Wang, Q.: Netfpga-based firewall solution for 5g multi-tenant architectures. In: 2019 IEEE International Conference on Edge Computing (EDGE), pp. 132–136 (2019). IEEE
27. Ricart-Sanchez, R., Malagon, P., Matencio-Escolar, A., Alcaraz Calero, J.M., Wang, Q.: Toward hardware-accelerated GOS-aware 5G network slicing based on data plane programmability. Trans. Emerg. Telecommun. Technol. **31**(1), 3726 (2020)
28. Ricart-Sanchez, R., Malagon, P., Alcaraz-Calero, J.M., Wang, Q.: P4-netfpga-based network slicing solution for 5G mec architectures. In: 2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), pp. 1–2 (2019). IEEE
29. Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., et al.: P4: programming protocol-independent packet processors. ACM SIGCOMM Comput. Commun. Rev. **44**(3), 87–95 (2014)
30. Yan, Y., Beldachi, A.F., Nejabati, R., Simeonidou, D.: P4-enabled smart NIC: enabling sliceable and service-driven optical data centres. J. Lightwave Technol. **38**(9), 2688–2694 (2020)
31. Budiu, M.: Programming networks with p4. https://blogs.vmware.com/research/2017/04/07/programming-networks-p4/ (2017)
32. Bertrone, M., Miano, S., Risso, F., Tumolo, M.: Accelerating linux security with ebpf iptables. In: Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos, pp. 108–110 (2018)
33. Scholz, D., Raumer, D., Emmerich, P., Kurtz, A., Lesiak, K., Carle, G.: Performance implications of packet filtering with linux ebpf. In: 2018 30th International Teletraffic Congress (ITC 30), vol. 1, pp. 209–217 (2018). IEEE
34. Enberg, P., Rao, A., Tarkoma, S.: Partition-aware packet steering using xdp and ebpf for improving application-level parallelism. In: Proceedings of the 1st ACM CoNEXT Workshop on Emerging in-Network Computing Paradigms, pp. 27–33 (2019)
35. Miano, S., Doriguzzi-Corin, R., Risso, F., Siracusa, D., Sommese, R.: Introducing smartnics in server-based data plane processing: the DDOS mitigation use case. IEEE Access **7**, 107161–107170 (2019)
36. Ricart-Sanchez, R., Salva-Garcia, P., Chirivella-Perez, E., Alcaraz Calero, J.M., Wang, Q.: Empirical design, prototyping and evaluation of a new Hardware-Based network slicing approach for 6G backbone networks. In: 2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit): Network Softwarisation (NET) (2021 EuCNC & 6G Summit - NET) (2021)
37. eBPF - Introduction, tutorial & community resources . https://ebpf.io Accessed 16 Apr 2021
38. Cilium, BPF and XDP Reference Guide. https://docs.cilium.io/en/stable/bpf/. Accessed 4 May 2021
39. Vieira, M.A.M., Castanho, M.S., Pacífico, R.D.G., Santos, E.R.S., Câmara, E.P.M., Vieira, L.F.M.: Fast packet processing with EBPF and XDP: concepts, code, challenges, and applications. ACM Comput. Surv. (2020). https://doi.org/10.1145/3371038
40. Kicinski, J., Viljoen, N.: ebpf hardware offload to smartnics: cls bpf and XDP. Proceedings of netdev **1** (2016)
41. Docomo, N.: 5G Evolution and 6G, Whitepaper. (2020)
42. Chirivella-Perez, E., Calero, J.M.A., Wang, Q., Gutiérrez-Aguado, J.: Orchestration architecture for automatic deployment of 5G services from bare metal in mobile edge computing infrastructure. Wirel. Commun. Mobile Comput. **2018** (2018)
43. Chirivella-Perez, E., Gutiérrez-Aguado, J., Claver, J.M., Calero, J.M.A.: Hybrid and extensible architecture for cloud infrastructure deployment. In: 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, pp. 611–617 (2015). IEEE
44. Brewer, E.A.: Kubernetes and the path to cloud native. In: Proceedings of the Sixth ACM Symposium on Cloud Computing, pp. 167–167 (2015)

45. Nikaein, N., Marina, M.K., Manickam, S., Dawson, A., Knopp, R., Bonnet, C.: Openairinterface: a flexible platform for 5G research. ACM SIGCOMM Comput. Commun. Rev. **44**(5), 33–38 (2014)
46. ETTUS: USRP B210 SDR Kit - Dual Channel Transceiver (70 MHz–6GHz). https://www.ettus.com/all-products/ub210-kit/ Accessed 18 Feb 2021
47. Giordani, M., Polese, M., Mezzavilla, M., Rangan, S., Zorzi, M.: Toward 6G networks: use cases and technologies. IEEE Commun. Mag. **58**(3), 55–61 (2020)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Pablo Salva-Garcia is a postdoctoral researcher 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.

Enrique Chirivella-Perez is a researcher at the University of the West of Scotland, where he obtained his PhD. Enrique is Co-investigator in several Horizon 2020 EU projects, such as 6G-BRAINS, 5G-INDUCE and ARCADIAN-IoT. His main interests include E2E Network Slicing, Monitoring, network control and management, and Infrastructure zero-touch deployment.

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, ARCADIAN-IoT 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.

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, ARCADIAN-IoT and 6G BRAINS projects. His professional interests include network cognition, management, security and control, service deployment, automation and orchestration, and 5G mobile networks.

Authors and Affiliations

Pablo Salva-Garcia[1] · **Ruben Ricart-Sanchez**[1] · **Enrique Chirivella-Perez**[1] · **Qi Wang**[1] · **Jose M. Alcaraz-Calero**[1]

✉ Pablo Salva-Garcia
pablo.salva-garcia@uws.ac.uk

Ruben Ricart-Sanchez
ruben.ricart-sanchez@uws.ac.uk

Enrique Chirivella-Perez
Enrique.Chirivella-Perez@uws.ac.uk

Qi Wang
Qi.Wang@uws.ac.uk

Jose M. Alcaraz-Calero
jose.alcaraz-calero@uws.ac.uk

[1] School of Computing, Engineering and Physical Sciences, University of the West of Scotland, High Street, Paisley, Glasgow PA12BE, Scotland, UK