Toward a Practical Deterministic Datapath for 6G End Devices

Davide Rovelli, Michele Dalle Rive, and Patrick Eugster

Abstract

6G networks aim to achieve seamless integration of wireless and wired technologies through deterministic, end-to-end communication and processing across diverse technological domains (e.g., edge, cloud). While standards like 5G, Time Sensitive Networking (TSN), and Deterministic Networking (DetNet) have introduced significant advancements in predictable communication, much less attention has been given to ensuring deterministic packet processing on end devices. In many cases, especially in data centers, processing remains a primary source of unpredictable *jitter* since current *datapaths* - intended as packet processing software layers - focus heavily on optimizing common case latency and throughput, not considering the full tail latency spectrum. This paper provides the first comprehensive end-to-end latency characterization of high-performance packet processing technologies, highlighting their insufficient support for determinism. We propose a high-level design to address this issue and move towards a deterministic datapath for end devices employed in future 6G networks.

Introduction

6G calls for improved *deterministic, end-to-end* interactions among remote processes through better integration of many established and emerging architectural components (edge, cloud) as well as improved convergence across a wide range of technologies and standards [1].

End-to-End Perspective Requires Timely Processing. The "end-to-end" qualifier, which often refers to predictability and upper-bounded latency in network *communication* [2], was recently extended to include deterministic *processing* (i.e., computation) at the end devices [1]. However, recent mechanisms strongly focus only on the communication layer, commonly believed to be the weak point of interactions, dramatically lowering communication latency below the millisecond barrier while improving reliable delivery of time-critical traffic. These notably include recent 5G standard releases [3], Time Sensitive Networking (TSN), a set of IEEE 802.1 standards for deterministic Ethernet and Deterministic Networking (DetNet), an IETF framework to ensure bounded latency and reliability over IP networks. Following this trend, we can expect 6G to deliver even stronger guarantees on determinism in communication, putting pressure on the processing layer of end-to-end interactions, i.e., the *end device datapath*, which can no longer be neglected.

Processing Bottleneck in End Devices. The vast majority of end devices, including embedded, mobile, edge and datacenter appliances, run an Operating System (OS) to facilitate management of the underlying hardware. Widespread OSes deployed in production largely consist in a thin service layer sitting on top of a possibly customized Linux kernel (e.g., Android, Raspbian, Ubuntu Server) which includes networking functionality. Alas, the provided packet processing routines struggle to keep up with the pace imposed by the growing network capacity and end up becoming a bottleneck for high-throughput workloads which overload the end device with interrupts and packet processing tasks. This, often alongside computationally heavy application processing, causes contention in underlying compute resources and is particularly detrimental for time-sensitive tasks which can be preempted for an unbounded duration.

Improved Scalability With OS-Bypass. To address this inability to *scale* to heavy traffic load without compromising latency, OS-bypass frameworks, such as the Data Plane Development Kit (DPDK) [4], Remote Direct Memory Access (RDMA) [5] and more recently the extended Berkeley Packet Filter (eBPF)-based eXpress Data Path (XDP) [6], were developed to circumvent the OS stack by providing direct, fine-grained control of datapaths to application developers. These frameworks enable applications to bypass kernel involvement in certain operations, reducing overhead associated with the traditional network stack. For instance, DPDK facilitates high-performance packet processing in user space, while RDMA allows direct memory access between networked devices without Central Processing Unit (CPU) intervention, offering higher throughput and lower latency. Similarly, XDP extends eBPF to enable efficient packet processing at the lowest level of the network stack, allowing developers to execute custom actions on packets directly within the network driver. A large number of state-of-the-art systems combines such OS-bypass methods with custom schedulers, protocols, and hardware-offloading to optimize the tradeoffs between maximum end device processing throughput, common-case latency, and CPU utilization.

Davide Rovelli (corresponding author) is with the Faculty of Informatics, Università della Svizzera italiana (USI), 6900 Lugano, Switzerland, and also with SAP SE, 69190 Walldorf, Germany; Michele Dalle Rive is with the Faculty of Informatics, Università della Svizzera italiana (USI), 6900 Lugano, Switzerland, and also with the Computing Science Department, ETH Zürich, 8092 Zürich, Switzerland; Patrick Eugster is with the Faculty of Informatics, Università della Svizzera italiana (USI), 6900 Lugano, Switzerland.

Digital Object Identifier: 10.1109/MNET.2025.3551372
Date of Current Version: 13 May 2025
Date of Publication: 14 March 2025

While OS-bypass datapaths currently provide the best performance in packet processing at scale for end devices, it is unclear whether they can provide stronger guarantees to time-sensitive applications targeted by 6G networks.

The Gap to Determinism. While OS-bypass datapaths currently provide the best performance in packet processing at scale for end devices, it is unclear whether they can provide stronger guarantees to *time-sensitive* applications targeted by 6G networks. Recent related works focus on optimizing tail-latency for a low number of 9s of the 99.x^{th} percentile [7], which is not sufficient for the reliability requirements of future wireless and wired network standards [3]. This work fills the gap by providing extensive benchmarks of state-of-the-art OS-bypass methods and a systematic analysis of low *worst-case latency,* a key enabler of determinism. In this context, XDP is of particular relevance since, as a native subsystem of the Linux kernel, it is ubiquitous across the whole range of device types involved in 6G networks, from embedded devices to datacenter servers. We compare XDP against RDMA- the fastest OS-bypass datapath to our knowledge - and to classic OS-supported UDP sockets, to provide an overview of modern datapaths. While DPDK is also a valid candidate in this context, we exclude it from the analysis to focus on more emerging technologies. Our results highlight the brittleness of the methods evaluated, suggesting that current end device datapaths are not sufficient to cope with requirements of 6G networks.

Deterministic Datapath. Specifically, requirements for deterministic 6G datapaths are:

Reliability: Processing at end devices must guarantee upper-bounded latency for at least for 99.9999% of packets to match the reliability target of the latest 5G advanced release 18 standard (first introduced in release 16 [3]).

Low-Latency: Packet processing routines need to keep up with the speed of the communication layer boosted by previous standards (5G, TSN, and DetNet). This requires worst-case µs-scale reaction times for time-sensitive tasks under arbitrary processing load [1], [3].

Solutions to the unpredictability of the networking software include offloading time-sensitive tasks to hardware, e.g., Field Programmable Gate Arrays (FPGAs). Such approaches are very effective but require high development efforts and costs which are often prohibitive and lack flexibility for rapidly evolving applications. An alternative is to use specialized real-time OSs, such as FreeRTOS, which are however limited to simple devices (e.g., sensors, microcontrollers). However, large servers expected to be involved in 6G networks need to be able to accommodate a large number of applications with different requirements (not only time critical), making specific real-time OSes unsuitable for the purpose. A more generic approach to improve determinism of time-sensitive packet processing tasks is to fine-tune a general-purpose OS for low jitter performance. We show that, in practice, this approach currently has several limitations which might hamper deterministic end-to-end interaction and widespread adoption of 6G.

In the following, we start by giving an overview of state-of-the-art datapath technologies, then present our experimental analysis on deterministic latency, and conclude with proposing a design for deterministic and practical datapaths for 6G end devices and discussing open research questions therein.

Technologies Overview

This section outlines the main features and differences of the following candidate datapath technologies for deterministic 6G interactions, which we evaluate subsequently: UDP sockets, XDP, and RDMA. Listed from most generic to most specialized, these three technologies provide a complete spectrum of the choices currently available to network programmers.

UDP Socket

Sockets are a classic method of inter-process, message-passing communication provided by the OS. User datagram protocol (UDP) sockets are ubiquitous across OSes as they are the endpoint of one of the core communication protocols of the Internet protocol suite. We use UDP sockets as baseline program in our comparative analysis since they require the involvement of the OS network stack which is known to be a bottleneck for high-throughput workloads. Figure 1(a) overviews the communication steps occurring during UDP

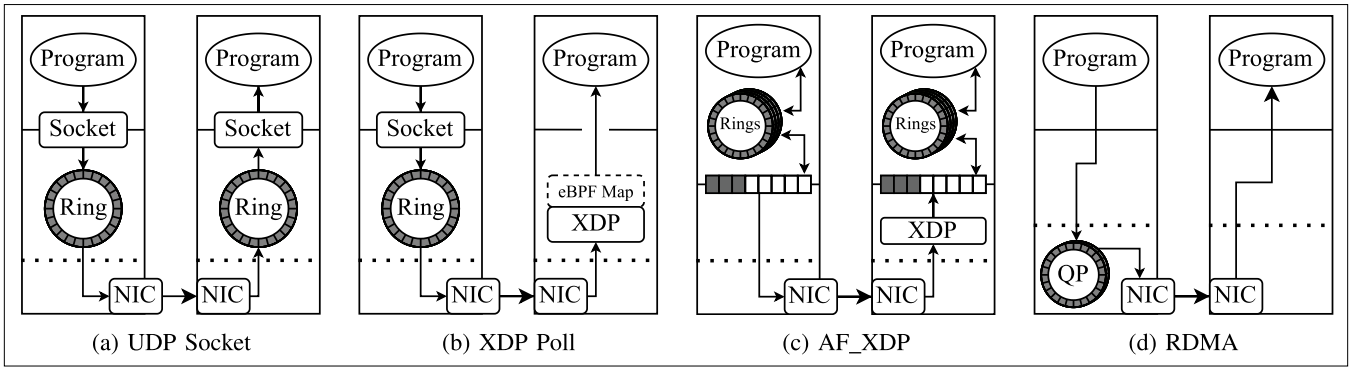

FIGURE 1. Diagrams depicting the high-level packet flow of four different datapaths evaluated in our setup. For each datapath, the left and right boxes represent sending and receiving end devices respectively. There are two lines inside each device: a solid line which separates user-space (top) from kernel-space (middle), and a dotted line which separates kernel-space from the NIC operations (bottom). a) UDP Socket. b) XDP Poll. c) AF_XDP. d) RDMA.

socket operations at the end device. Packets are copied from the userspace program to a kernel ring buffer (a fixed size, overwritable data structure managed by the OS) through calls to socket API. The OS then copies the packet to the network interface controller (NIC) memory, which transmits it over the network. These multiple copies and context switches are present in both the transmission and reception paths, increasing the risk of high and unstable latency.

Express Data Path (XDP)

eBPF's XDP [6] is an OS-bypass framework that enables high-performance programmable packet processing in the Linux kernel. XDP is an established subsystem of the Linux Kernel (introduced in v3.18, 2014), making it widely available in the entire range of Linux-powered of devices, including Android mobiles [8]. XDP sits at the lowest level of the network stack, allowing developers to execute custom actions on packets directly within the network driver. To illustrate a scenario where XDP could enhance determinism, consider a programmer using it to offload time-sensitive components of a large application (e.g., monitoring critical events) while allowing the rest of the application traffic to pass through the default network stack, where packet processing occurs in the standard best-effort manner. We evaluate two different XDP-based datapaths which we outline below.

XDP Poll: This datapath exploits eBPF maps, a mechanism that allows eBPF programs to store data in a shared memory region. We use a map as single-producer single-consumer ring buffer between kernel space and user space. Figure 1(b) shows a high-level diagram of XDP Poll in which the XDP program detects specific packets and inserts them in the map. The user-space application actively checks the shared memory waiting for newly arrived packets and handling as they arrive. This method does not offer any advantage at transmission. In our experimental program, we use a normal UDP socket to send packets.

AF XDP: `AF_XDP` [9] is a type of socket protocol introduced in Linux Kernel v4.18 that allows XDP programs to redirect frames to a user-space memory buffer. The method is visualized in Figure 1(c). The user-space application handles packets management, sending and reception through four rings associated with a shared memory region, offering fine-grained control. This datapath is composed of a XDP program that redirects packets to using an eBPF map, and a user-space application that processes the packet.

Remote Direct Memory Access (RDMA)

The second OS-bypass datapath evaluated, RDMA [5], is a popular communication framework that can perform direct memory access on remote hosts, without involving the host's OS or CPU. To accomplish this, RDMA exploits powerful NICs which can establish access to a shared memory location of the host at setup time. Unlike UDP sockets and XDP, RDMA requires specific hardware features which are only available on certain server-only NICs (currently not available in mobile and embedded devices), limiting its adoption on cloud servers and large workstations at the edge. Nonetheless, we choose to include RDMA as a reference of the (probably) best possible performance in achievable with modern devices. Figure 1(d) shows the strong bypass capabilities of RDMA, notably the use of only one "domain" crossing (from user space to NIC) compared to other methods. All Tx and Rx operations are posted on Queue Pairs (QPs), which are composed of a send queue and a receive queue. Our implementation uses *verbs* API (two-sided) on RDMA over Converged Ethernet (RoCE) and Unreliable Datagram (UD) transport mode, for fair comparison with UDP.

Experiments and Latency Analysis

This section outlines our experimental analysis of OS-bypass datapaths in order to investigate their suitability as deterministic packet processing layers for 6G networks devices.

Motivation

OS-bypass datapaths are considered to be the default solution for achieving optimal packet processing performance, with their main design goal being to withstand high network traffic with low packet processing latency. While end-to-end interactions targeted by 6G [1] certainly require such processing scalability, they have the additional requirement on deterministic delivery, which translates into *upper-bounded* packet processing latency. Unfortunately, state-of-the-art prototypes [7], [10] and recent analyses [11], [12] provide only partial evaluation of tail latency, limited to low number of 9s of the 99.x^{th} percentile, which is not sufficient for the reliability requirements of future 6G wireless and wired network standards. We deploy a benchmark suite to address this information gap, evaluating *worst-case* latency and jitter of the different packet processing frameworks. We refrain from evaluating the performance of the network, which is not intended to represent a possible 6G network, but is only used as mean to test the end device datapaths.

To motivate previously defined requirements, our evaluation addresses the following research questions.

- **RQ1:** How stable is the latency of end device datapaths at varying throughput (the section "RQ1: Throughput Response")? (cf. *reliability*)
- **RQ2:** How do end devices datapaths respond to different CPU loads and performance tuning settings (the section "RQ2: Tuning Effects")? (cf. *reliability, low-latency*)

Methodology

Each experimental latency distribution is obtained from at least 1 million ping-pong rounds in which a client node starts sending ping packets at a fixed periodic rate to a server, which sends them back as pong packets upon reception. We ensure that the network is free from any traffic other than the ping-pong periodic traffic in order to solely evaluate the packet processing response. We run the experiments using the previously-introduced OS-bypass datapaths (XDP Poll, AF_XDP, RDMA) and use classic UDP socket as baseline. In the experiments including CPU stress, the CPU load is generated by the `stress-ng` tool. We run all tests using small and large packet sizes of 128B

and 1KiB to investigate possible size-related optimizations of the different bypass systems. Since the obtained measurements show no major effect between the two, we only report results for 1KiB packets. We attribute this to the fact that both 128B and 1KiB are within the Maximum Transmission Unit (MTU) (1500B) of our system. Evaluation of larger packet sizes is outside of scope since (1) packets within the MTU are expected to deliver the best latency performance [11] and (2) datacenters often disable packet fragmentation as it is too detrimental for high-speed networking.

Our setup consists in two off-the-shelf servers connected via 100Gbps Ethernet through an Edgecore Wedge 100BF-32X TOR switch. Servers are equipped with Intel Xeon Gold 5315Y 3.20 GHz CPUs, 188G of RAM, Mellanox ConnectX-7 100Gbps NICs and run Ubuntu 22.04 with Linux kernel v6.6.19. We focus our analysis mostly on servers since they constitute the convergence point of several end-to-end systems. Furthermore, since 6G networks are expected to extend across multiple domains including the Cloud [1], it is likely that powerful servers will provide the upper bound in processing performance among devices connected to a 6G network, since they have more powerful hardware (i.e., CPU frequency, networking speed) compared to mobile and embedded devices. To demonstrate servers' faster processing capabilities, we perform UDP socket latency tests on a Linux-based (Raspbian Buster) embedded device: a Raspberry Pi 4B with 2GB RAM, connected to a 5GHz Wi-Fi network. We do not provide extensive evaluation of slower embedded and mobile devices since, as we show shortly, servers already display significant limitations in deterministic packet processing. Experiments refer to the server setup unless specified otherwise.

Source code is available at https://github.com/swystems/ det-bypass

Metrics

We insert four 64B timestamps into every ping-pong packet, just before every send and receive event. Note that timestamps from different nodes cannot be directly compared since the clocks of the hosts are not synchronized. Packets include a unique monotonically increasing identifier and a random nonce used for error-checking and synchronization. We use timestamps to obtain the following metrics:

- **Latency:** End-to-end latency consisting of network delay and processing delay of each packet. Inspired by clock synchronization methods, we assume that latency is similar for both ways and obtain it as half the round-trip time of a packet. Timestamps on the receiver side are only used to calculate the processing time on the receiver side.
- **Jitter:** Difference between maximum latency and minimum latency of a specific method. It captures how stable the latency is.

In our setup with no additional traffic, and benchmark traffic throughput well below network capacity, a perfectly deterministic packet processing layer should result in only the jitter corresponding to the variability in switch forwarding latency (expected to be in the ns range due to hardware precision).

RQ1. Throughput Response

Figure 2 shows the latency distribution of the analyzed methods. We choose 8 Mbps as lower bound, i.e., 1 packet sent every 1 ms, interval which is largely sufficient to process a single packet for any datapath, minimizing the probability of queuing. We then increase throughput by 10× (80Mbps and 800 Mbps) until 1600 Mbps, which we find to be the experimental (single-core) limit of our system beyond which we start observing inconsistent latency values. As expected, higher throughput values lead to higher latency variance for all approaches as it can be seen by the increasing height of the boxes. Overall, RDMA shows the best median latency around 3.3µs, evidencing the benefits of NIC hardware support. XDP Poll has a more variable latency with a jump from 9µs to 128µs at 800Mbps while AF_XDP achieves much better scalability with stable median latency around 15µs. The interesting phenomenon of median latency reduction for higher throughput values is covered as part of the section "RQ2: Tuning Effects" OS-bypass datapaths manage to keep the 99.99th percentile latency below 100µs for the two lower throughput values but suffer from a 2× increase at higher rates.

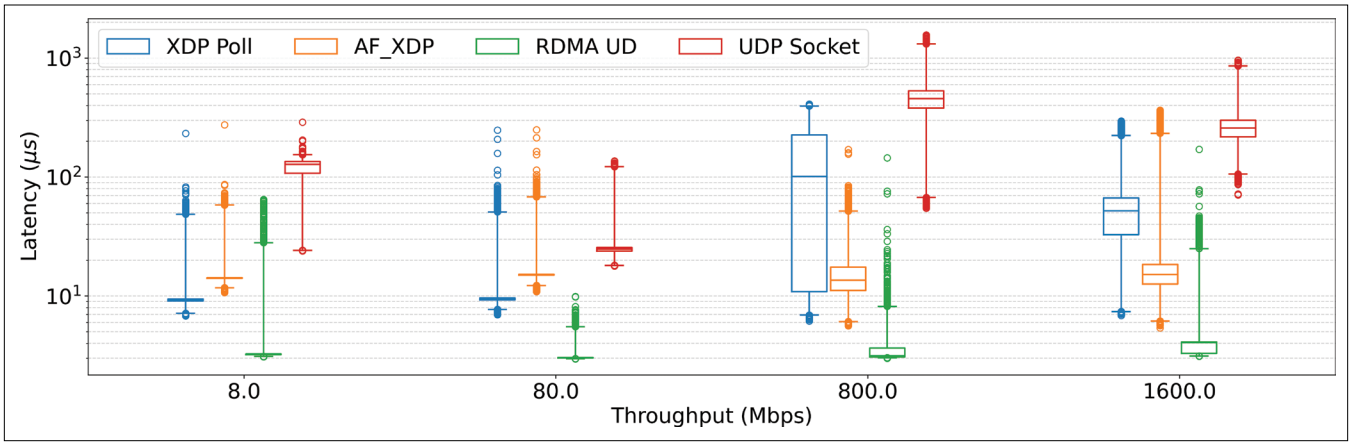

FIGURE 2. Single trip latency of 1KiB packets at increasing throughput values. Boxes edges range from 25% to 75% of the distribution, whiskers from 0.01% to 99.99% percentile and circles show the remaining outliers. The y-axis is in log-scale. Boxes are ordered left to right for each method listed in the legend.

Takeaways. Our tests show that both XDP and RDMA introduce non-negligible jitter, resulting in over 10× increase in latency in the worst case, slightly above 100µs. Overall, the best approach under normal system conditions is RDMA and the worst is UDP in terms of median and maximum latency, while all approaches perform similarly in terms of relative jitter.

RQ2; Tuning Effects

This experiment analyzes the effects of performance tuning for low jitter on the evaluated methods. In order to showcase the impact of performance tuning even for light processing loads, we choose a fixed low throughput of 80Mbps for all our tests. Inspired by seminal work on deterministic coordination [13] and system tuning guides [14], we apply the following system optimizations. We

- pin the sender and receiver processes to cores assigned to the `isolcpus` kernel boot parameter;
- route all IRQs away from the dedicated cores using the `proc` interfaces to further limit (but not disable) OS interrupts targeting our isolated cores;
- disable interrupt coalescence settings in the NIC to avoid unwanted IRQ batching at send and receive;
- set the CPU governor of dedicated cores to the highest power state (Intel C-state C0) to avoid costly wake-ups from sleep states.

Figure 3 and Table 1 present latency metrics. We report the latency distribution of the default system with no load from RQ1 experiments (labelled "Default") and compare it against the system under 80% CPU load (labelled "Default & Stress") and the system with performance tuning under the same 80% CPU stress (labelled "Tuned & Stress"). Application of stress dramatically deteriorates the default system's worst-case latency performance by at least 100× for server experiments: upper whiskers (99.99th percentile) and outliers reach almost ~4ms latency for both XDP methods and RDMA, with UDP sockets reaching ~7ms peaks. The tuned system, on the other hand, manages to achieve much better worst-case latency, even improving on the default configuration: isolation settings prevent context switch interference introduced by arbitrary CPU load. The same effect can be seen on the embedded device (Raspberry Pi) wireless setup, which suffers from a 5× (72ms) jitter increase upon high CPU load, mitigated by tuning which brings it down to 1.8× (23ms). As expected, the absolute latency and jitter values are much higher than in the wired server setups. We omit XDP tests for the Raspberry Pi setup due to the lack of native XDP support from the network driver.

A key outcome of performance tuning is that the 99.99th and 100th percentile latencies of *all* approaches in server experiments are much closer to each other than in the default setup. An interesting observation is that mean and median latencies of the untuned configuration with high CPU stress seem to improve on the default system configuration with no stress. We attribute this to CPU power states: a busy CPU is less likely to be assigned a lower power state, hence lower frequency, rendering the CPU more responsive in the common case. The tuned configuration achieves the same effect by manually setting the CPU governor to performance mode.

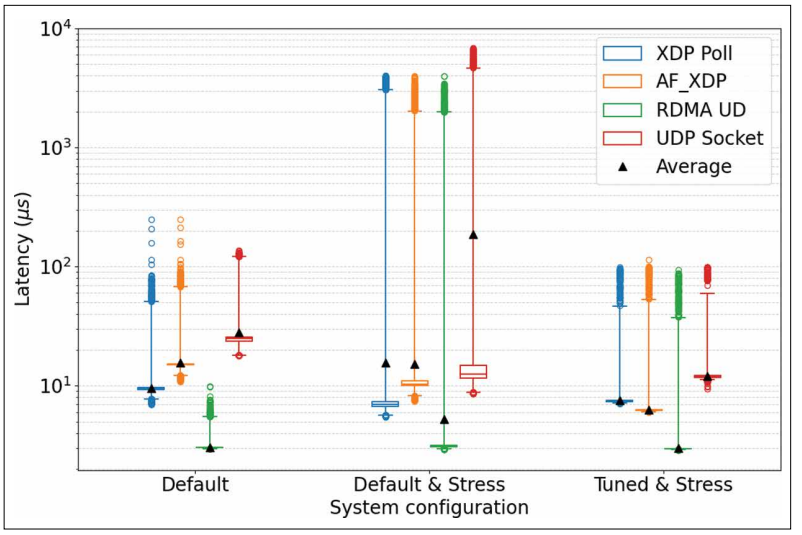

FIGURE 3. Latency distribution at 80 Mbps with different system configurations. The left plot reports results of the default system from Figure 2, to which we apply 80% CPU stress (central plot) and then tune the system for latency-critical performance (right plot). The y -axis is in log-scale.

Combining an OS-bypass datapath with performance tuning provides the best of both worlds: low average latency at high traffic load and limited jitter at high processing loads.

Takeaways. Results suggest that performance tuning of the OS is the predominant factor in jitter mitigation, independently from the datapath used. This is required in Linux-based servers and embedded devices under heavy CPU load to prevent processing latency spikes overbearing network latency. Combining an OS-bypass datapath with performance tuning provides the best of both worlds: low average latency at high traffic load and limited jitter at high processing loads.

The Usability Barrier Hindering Determinism

OS-bypass methods have a degree of specialization which sits in between established OS-supported network I/O methods, such as UDP sockets, and custom hardware solutions, such as dedicated FPGAs or ASICs. Nonetheless, we find that they still require substantial development effort, further aggravated by the equally challenging procedure of tuning the OS for low latency and jitter. In this section we gauge the engineering effort involved in the attempt of bringing determinism to the packet processing layer, discussing how limited usability is detrimental for determinism.

Development Effort

Table 2 shows the Lines of Code (LoC) used for the C implementation for each datapath with standard optimization. As expected, the well-established UDP socket abstraction results in the smallest number of LoC. XDP Poll follows with a total of 600 LoC which include additional user-space logic to implement the polling functionality and the eBPF map operations in kernel space.

Method	Median latency			99th percentile lat.			99.99th percentile lat.			Maximum latency			Jitter		
	D	D&S	T&S	D	D&S	T&S	D	D&S	T&S	D	D&S	T&S	D	D&S	T&S
XDP Poll	9.44	7.02	7.46	11.22	12.84	9.07	50.8	3051	46.72	248.3	4003	99.46	241.3	3998	92.38
AF_XDP	15.09	10.36	6.24	46.34	14.56	7.81	68.12	2024	52.91	249.6	3998	114.6	238.7	3990	108.6
RDMA UD	3.02	3.1	2.96	3.59	3.41	3.16	5.52	1997	37.4	9.9	3974	93.57	6.94	3971	90.69
UDP Socket	24.82	12.46	11.92	85.21	2174	14.09	122.2	4669	60.69	136.8	6827	99.37	118.9	6818	89.96
UDP Socket (RPi setup*)	1821	2912	1823	11026	66592	19805	13725	74329	24225	14723	75325	24844	12902	72413	23021

TABLE 1. Summary of latency metrics at 80 Mbps. System configurations are indicated with D = Default, D&S = Default and Stress, T&S = Tuned and Stress. Values are in μs. *Embedded test setup using Raspberry Pi 4 on a 5GHz wireless network.

Method	Userspace LoC	Kernelspace LoC
UDP socket	176	0
XDP poll	397	103
AF_XDP	689	45
RDMA	625	0

TABLE 2. Lines of Code (LoC) per implemented method. All programs were written in C and use standard optimization.

The two more flexible approaches, AF_XDP and RDMA come with similar development overhead due to concurrent low-level buffer handling logic. This showcases the typical cost of specialization, which makes the adoption and optimization of OS-bypass technologies a difficult task. We argue that the limited maturity of OS-bypass technologies is also a major contributing factor to the added development effort and advocate for the need of suitable abstraction layers without trading off performance, as attempted by some projects, e.g., Demikernel [10].

Like other state-of-the-art low-latency frameworks [13], we applied low-latency performance tuning of the OS following engineering best practices [14], involving manual configuration of scheduling, power management, Linux kernel and NIC settings (the section "RQ2: Tuning Effects"). The procedure was very time-consuming and error-prone, requiring several weeks of trial-and-error iterations needed to find the right "recipe". High resource consumption is another consequence of the tuning procedure which, in our analysis, requires one full CPU core idling with performance power settings, causing non-negligible disruption to other processes, especially under heavy load.

Usability Challenges

The set of software tools to enable high-performance packet processing currently available in commodity OSs (e.g., Linux) make it possible to achieve a certain degree of reliable, low-latency packet processing. However, while it remains uncertain whether their *programmability* will meet the stricter determinism demands anticipated for future 6G networks and wired standards like TSN and DetNet, our analysis highlights a notable gap in *usability*. This encompasses both the high complexity involved in the implementation of highly specialized OS-bypass techniques and, most importantly, the absence of straightforward mechanisms for configuring "deterministic" settings. We claim that poor usability is a major barrier to determinism for two reasons. First and foremost, it hinders the correct implementation of deterministic programs, leading to situations in which a single misconfigured option of a single device would impact the end-to-end response time guarantees of a whole system. Secondly, it affects reproducibility, obstructing widespread adoption of a deterministic datapath across a wide range of device types. We argue that substantial advances are required to improve usability in the packet processing layer to better support *reliability* and *low-latency* targets of current networked systems and, even more critically, of 6G networks.

Design Guidelines for a Deterministic and Practical 6G Datapath

In this section we elaborate on the takeaways from our analysis to propose a generic design of a packet processing layer which is both deterministic, meaning it provides *reliable* packet delivery with *low latency*, and is *practical*. Figure 4 shows the proposed design which we discuss in the sections below.

Combining OS Tuning With OS Bypass

State-of-the-art datapaths (i.e., OS-bypass technologies) provide unmatched scalability, allowing end devices to withstand hundreds of Gbps of throughput from rapidly-evolving networks. However, they are not sufficient to guarantee reliable low-latency interactions under high processing load, which is, unfortunately, a growing requirement of edge-cloud systems in 6G networks, especially in end devices. Let's take, for instance, AF_XDP's 99.99th percentile one-way latency between two powerful servers under processing load of 2 ms (see Table 1). Considering that the 5G new radio (NR) standard release 17 targets one-way Radio Access Network (RAN) downlink or uplink user-plane latencies of 0.5 ms with reliability up to 99.9999th percentile [3] (unchanged in 5G-advanced release 18), we can easily see how processing can result in having larger "non-determinism" in end-to-end communication. Our analysis indicates that performance fine-tuning of the end device OS is essential to cap latency to around 100µs, falling into the 5G target range.

Future deterministic packet processing layers should therefore combine high-performance packet processing with performance tuning of the OS as shown in Figure 4, possibly leveraging techniques from established real-time OSes. Moreover, our analysis substantiates the need for advances in both packet processing technologies and performance tuning to better support determinism. This is required to keep up with the evolution of 6G networks in terms of *reliability* and *low-latency* and avoid processing overhead to become the bottleneck of the communication.

A Practical, Homogeneous Packet Processing Layer

In the section "The Usability Barrier Hindering Determinism" we discussed how the limited usability of current packet processing layers and commodity OSs hampers determinism of end-to-end interactions in current and future networked systems. Additionally, as highlighted by recent proposals of integration of different mobile networks innovations into a consistent architecture, "6G superpower must be simplicity" [15]. Establishing suitable abstraction layers without trading off performance is therefore a primary requirement of end devices in 6G networks. Demikernel [10], a seminal work in this area, introduces a promising datapath design which aggregates multiple OS-bypass systems to enhance usability and portability. However, neither Demikernel, nor other work, to our knowledge, addresses end-to-end, reliable processing.

We propose an architecture draft (Figure 4) where a deterministic datapath offers a homogeneous, full-featured interface to the user to significantly improve usability of configuring and programming end devices. The interface benefits from (1) including existing high-performance packet processing frameworks and should seamlessly (2) integrate endpoints for configuration performance tuning. We include both features into the deterministic datapath (green box in Figure 4), which time-sensitive applications use as a reliable packet processing layer to send and receive packets over a 6G network. This component should also be extendable with future technologies and mechanisms arising from end device software research. Support of a commodity OS (e.g., Linux-based) would also foster widespread adoption. This would allow applications to seamlessly choose between normal, best-effort packet processing, or deterministic packet processing, providing the tools to optimize a given workload through a single interface.

Conclusion and Outlook

Following the trend from 5G, 6G networks are expected to deliver strong deterministic guarantees for end-to-end communication between time-sensitive applications. While the proposed integrations with wired standards such as TSN and DetNet show promising advances in the communication layer, the packet processing layer at the end devices has been left out of the picture. We show that current high-performance packet processing technologies, namely OS-bypass datapaths, struggle to keep up with the speed and reliability of the network. Going forward we advocate for improvements on how quickly and reliably network packets are processed at the end devices. In particular

- additional work is required to further increase reliability and speed of datapaths in 6G end devices subject to heavy processing load and multitasking. Namely, datapaths should aim at upper-bounded, µs-scale processing latency as targeted by recent 5G standards targets [3].
- deterministic response times should be guaranteed across the entire spectrum of 6G devices (datacenter servers, mobile, embedded, etc).
- more work is needed to abstract highly complex implementation and configuration of deterministic packet processing pipelines involving OS-bypass technologies and performance tuning. A practical datapath is required to facilitate convergence in 6G network devices, enabling error-free deterministic processing.
- the community needs to address the challenging task of providing the right

A practical datapath is required to facilitate convergence in 6G network devices, enabling error-free deterministic processing.

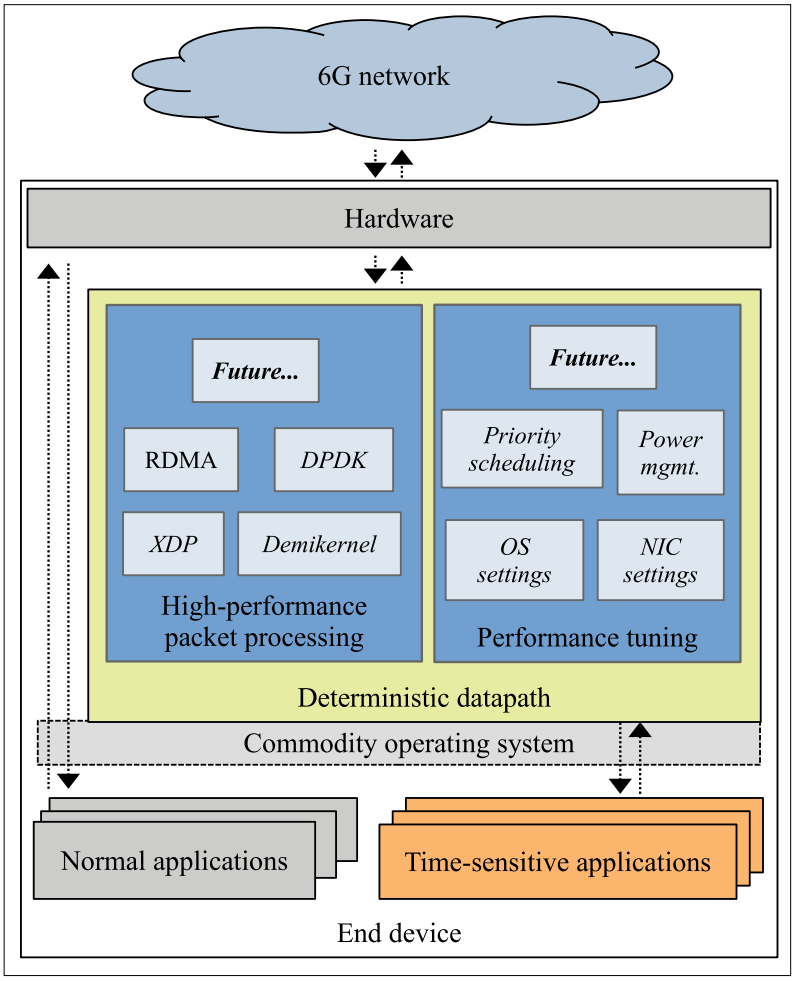

FIGURE 4. Architecture draft for a practical, homogeneous datapath to support deterministic processing in 6G end device communication.

interfaces to programmers, allowing them to tailor deterministic properties to the specific needs of an application, striking the balance between programmability and practicality.
- deterministic datapaths need to support a very diverse software and hardware ecosystem. Future datapaths should be easily extendable to support new technologies (frameworks, technologies, OS configurations, hardware platforms, etc.) and automatically adapt to the underlying system.

We conclude by proposing a high-level architecture draft of a deterministic datapath which addresses some of the above-mentioned challenges.

Acknowledgment

This work was supported in part by SNSF under Grant 192121 and Grant 197353, in part by the Hasler Foundation, and in part by the Meta Research through Research in Distributed Systems Award.

References

[1] G. P. Sharma et al., "Toward deterministic communications in 6G networks: State of the art, open challenges and the way forward," *IEEE Access*, vol. 11, pp. 106898–106923, 2023.
[2] M. Polese et al., "Toward end-to-end, full-stack 6G Terahertz networks," *IEEE Commun. Mag.*, vol. 58, no. 11, pp. 48–54, Nov. 2020.
[3] T.-K. Le, U. Salim, and F. Kaltenberger, "An overview of physical layer design for ultra-reliable low-latency communications in 3GPP releases 15, 16, and 17," *IEEE Access*, vol. 9, pp. 433–444, 2021.
[4] (2024). *DPDK, Data Plane Processing Kit*. Accessed: May 15, 2024. [Online]. Available: https://www.dpdk.org/
[5] (2007). *RFC 5040—A Remote Direct Memory Access Protocol Specification*. Accessed: May 15, 2024. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5040
[6] (2024). *Cilium EBPF and XDP Reference*. Accessed: Oct. 29, 2024. [Online]. Available: https://docs.cilium.io/en/latest/bpf/
[7] S. McClure et al., "Efficient scheduling policies for microsecond-scale tasks," in *Proc. 19th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Apr. 2022, pp. 1–18.
[8] (2024). *Android–Extend the Kernel With EBPF*. Accessed: Oct. 29, 2024. [Online]. Available: https://source.android.com/docs/core/architecture/kernel/bpf//
[9] (2018). *AF_XDP—The Linux Kernel Documentation*. Accessed: May 15, 2024. [Online]. Available: https://www.kernel.org/doc/html/v5.12/networking/af_xdp.html
[10] I. Zhang et al., "The demikernel datapath OS architecture for microsecond-scale datacenter systems," in *Proc. ACM SIGOPS 28th Symp. Operating Syst. Princ.*, New York, NY, USA, Oct. 2021, pp. 195–211.
[11] D. Géhberger et al., "Performance evaluation of low latency communication alternatives in a containerized cloud environment," in *Proc. IEEE 11th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2018, pp. 9–16.
[12] K. C. du Perron, D. L. Pacheco, and F. Huet, "Understanding delays in AF_XDP-based applications," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2024, pp. 5497–5502.
[13] P. Jahnke et al., "Live in the express lane," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, Jul. 2021, pp. 581–597.
[14] (2020). *Erik Rigtorp's Low Latency Tuning Guide*. Accessed: May 15, 2024. [Online]. Available: https://rigtorp.se/low-latency-guide/
[15] G. Pongrácz et al., "Towards extreme network KPIs with programmability in 6G," in *Proc. 24th Int. Symp. Theory, Algorithmic Found., Protocol Design Mobile Netw. Mobile Comput.*, New York, NY, USA, Oct. 2023, pp. 340–345.

Biographies

Davide Rovelli (roveld@usi.ch) received the B.Eng. and M.Eng. degrees in electronic and software engineering from the University of Glasgow. He is currently pursuing the Ph.D. degree with the Software Systems (SWYSTEMS) Laboratory, Università della Svizzera italiana (USI). His research explores fast and reliable distributed systems with particular focus on synchronous systems leveraging latest datacenter technologies, including operating systems, in-network computing, and hardware/software co-design.

Michele Dalle Rive received the B.Sc. degree in informatics from Universita della Svizzera italiana (USI). He is currently pursuing the M.Sc. degree in computer science with ETH Zürich. During his time as a Research Assistant at USI, he worked on system programming for synchronous systems, with focus on operating system bypass, low-level optimizations, and safe programming languages.

Patrick Eugster received the M.S. and Ph.D. degrees from EPFL in 1998 and 2001, respectively. He is currently a Full Professor of computer science with Università della Svizzera italiana (USI), where he leads the Software Systems (SWYSTEMS) Group. His research interests include distributed systems, especially their intersection with security and programming languages. He is a member of the DARPA Computer Science Study Panel in 2011. He was a recipient of the NSF CAREER Award in 2007 and the ERC Consolidator Award in 2012.