EasyChair Preprint

№ 11093

FlexPipe: Fast, Flexible and Scalable Packet Processing for High-Performance SmartNICs

Klajd Zyla, Marco Liess, Thomas Wild and Andreas Herkersdorf

EasyChair preprints are intended for rapid dissemination of research results and are integrated with the rest of EasyChair.

October 13, 2023

FlexPipe: Fast, Flexible and Scalable Packet Processing for High-Performance SmartNICs

Klajd Zyla, Marco Liess, Thomas Wild, Andreas Herkersdorf
Chair of Integrated Systems
Technical University of Munich
Munich, Germany
{klajd.zyla, marco.liess, thomas.wild, herkersdorf
@tum.de}

***Abstract*—Data centers have been struggling to provide the necessary processing capacity to handle the surging rate of network traffic that is generated in an increasingly connected and service-oriented world. As a result, SmartNICs play an even more important role than before as they can offload various network applications and hence free CPU resources for application-layer processing, increase performance and reduce processing time. However, they often do not support flows with different offload requirements and cannot dynamically allocate offloads in runtime. In order to address these limitations, we propose FlexPipe, a fast, flexible and scalable packet-processing architecture for high-performance SmartNICs. Our design enables low-latency and runtime-reconfigurable packet forwarding at high traffic rates with minimal area overhead. Furthermore, it provides load-aware packet steering toward multiple offload units of the same type for low-bandwidth offloads. We implement a prototype of FlexPipe in Verilog and validate it via cycle-accurate register-transfer level simulations. Our evaluation results show that FlexPipe can process packets of arbitrary sizes with different offload requirements at line rate and on average 1.9x faster than a SmartNIC with a predefined sequence of offloads and 1.8x faster than PANIC, a flexible state-of-the-art SmartNIC.**

***Index Terms*—SmartNICs, Packet processing, Load balancing, SDN, 6G**

I. Introduction

As the world we live in becomes increasingly connected and more online services emerge, such as VR/AR, streaming, telepresence and diverse smartphone applications, network traffic reaches a higher rate [1]. While technology improvements have allowed data links and transceivers to keep up with the bandwidth requirements, data centers have not been able to meet the resulting demands concerning CPU performance and response time due to the increase of power consumption, thus creating a performance gap [2]. Another limiting factor is the frequency at which a CPU needs to access memory. The access time of the L3 cache of a modern CPU is $10-15\,\mathrm{ns}$ [3], but a SmartNIC that processes packets at a rate up to $100\,\mathrm{Gbit/s}$ can deliver a $64\,\mathrm{B}$ packet every $5.12\,\mathrm{ns}$. A common approach to tackle this problem is to offload network applications to SmartNICs, which frees processing capacity at the server, increases performance and reduces power consumption and response time. A vast amount of research has focused on optimizing these devices in different aspects or domains, such as I/O Virtualization [4], programmability [5]–[9], TCP/IP stack [10], receive-side scaling (RSS) [11], RDMA [12] and RPCs [13].

Low latency and high flexibility are two major requirements, not only for cloud and service providers, but also for radio access network (RAN) operators, as reflected in the expectations for future mobile communication generations, such as 6G [14]. Low latency is crucial for fulfilling the application requirements, while flexible packet-processing architectures are needed to support emerging protocols and network applications. SmartNICs typically contain specialized hardware to reduce the processing time to a minimum for various offloads, such as encryption, firewall, decompression, QoS and TCP [15]. These offloads are usually arranged sequentially in a pipeline in the order that they are intended to be activated. Some SmartNICs contain FPGAs and can hence be reconfigured to support different use cases. However, they do not necessarily accommodate flows that require different offloads in the same pipeline and usually provide no possibility to change the allocated offloads in runtime. For example, this feature is useful when a server receives traffic from clients that reside in the same network, which is neither encrypted, nor compressed, as well as traffic from clients that reside in a different network, which may be encrypted and/or compressed.

In this paper we present FlexPipe, a fast, flexible and scalable design for high-performance SmartNICs. FlexPipe enables low-latency packet processing at line rate with minimal area overhead, while deploying a variety of offloads that can be dynamically mapped to multiple flows in runtime. As main contributions in this work, we

- define a packet-processing architecture focusing on efficient packet forwarding in a pipeline of offloads and load-aware packet steering toward offload units,
- implement a prototype of FlexPipe in synthesizable hardware description language (HDL) code, and
- evaluate our design via cycle-accurate register-transfer level (RTL) simulations.

Our evaluation results show that FlexPipe can process packets of arbitrary sizes with different offload requirements

We acknowledge the financial support from the Bavarian Ministry of Economic Affairs, Regional Development and Energy in the context of the project "6G Future Lab Bavaria".

at line rate without dropping any packets. Furthermore, our design can process packets corresponding to different flows at a traffic rate of $90\,\mathrm{Gbit/s}$ on average 1.9x faster than a SmartNIC with a predefined sequence of offloads and 1.8x faster than PANIC [16], a flexible state-of-the-art SmartNIC.

II. Related Work

In this section we group SmartNIC designs in three main classes and compare them with each other based on the following metrics: performance, latency, offload variety, flexibility and chip area. Then we describe PANIC [16], a state-of-the-art SmartNIC, and point out some of its drawbacks.

A. SmartNIC Designs

SmartNIC designs can be broadly categorized in three main classes: pipeline-of-offloads designs [4], [7], [10], [12], [13], manycore designs [9], [17] and match-action table (MAT)-based designs [5], [11].

Pipeline-of-offloads designs consist of multiple serially connected offloads, which are usually hardware accelerators that perform specific packet-processing functions, such as checksum verification, authentication, decryption and RSS. The order of execution is known in advance and it dictates the sequence in which the modules are arranged. Such designs deploy a variety of offloads that are optimized for high performance and low latency. However, they usually have fixed forwarding logic, which means that all incoming packets are forwarded to all offloads in the same sequence. Empty pipeline stages can be used to avoid packet manipulation from units whose function is not required. This leads to additional delay for packets that require only a subset of the available offloads.

Manycore designs are composed of a large amount of CPU cores that can process packets in parallel. By sharing the compute load of incoming traffic between multiple cores, such designs can achieve a high throughput. Nevertheless, whether they can process packets at line rate or not, depends on the number of instructions per packet [17]. Like pipeline-of-offloads designs, manycore designs can deploy a variety of offloads by executing different functions in different cores. By contrast, they are significantly more flexible because they can be programmed to execute any application. However, since CPU cores are not optimized for packet processing, the per-packet processing time is significantly longer. Furthermore, as the amount of cores required to achieve a high throughput increases, the design consumes a larger chip area.

MAT-based designs contain a pipeline of reconfigurable match-action tables, which implement various packet-processing operations. MATs can be programmed to extract different header fields and they can be reconfigured in runtime to match on arbitrary values and perform various actions, such as modify the value of a header field. Hence they are more flexible than pipeline-of-offloads designs, but not as flexible as manycore designs. Each pipeline stage must complete the processing of a packet in one clock cycle [18], thus making such designs capable of receiving packets at a high rate and reducing the processing time to a minimum. However, this constrains the variety of offloads they can provide to simple operations that can be executed in one clock cycle. Consequently, MAT-based designs do not inherently support complex functions, such as authentication, encryption and decompression.

Table I provides a qualitative comparison of the aforementioned SmartNIC designs based on five metrics: performance, latency, offload variety, flexibility and chip area.

TABLE I
Qualitative comparison of existing SmartNIC designs: green check mark means the metric is fulfilled; orange check mark means the metric is partially fulfilled; X mark means the metric is not fulfilled.

SmartNIC design	Performance	Latency	Offload variety	Flexibility	Chip area
Pipeline	✓	✓	✓	✗	✓
Manycore	✓	✗	✓	✓	✗
MAT	✓	✓	✗	✓	✓

Pipeline-of-offloads designs lack the capability to forward packets to arbitrary offloads. This is a very useful feature in use cases where a SmartNIC processes packets associated with flows that need different sets of offloads. In such scenarios a SmartNIC vendor would resort to deploying the largest set of required offloads in the same pipeline, which would lead to additional processing time for packets that need only a subset of the available offloads. Some designs contain several pipelines in parallel, which packets belonging to different types of traffic are forwarded to. However, they use additional chip area in order to replicate offloads needed by multiple flows in different pipelines and provide no possibility to change the allocated subsets of offloads in runtime.

B. PANIC

PANIC [16] is a high-performance programmable NIC for multi-tenant networks, which provides a solution to the aforementioned issue. It contains flexible forwarding logic realized as a crossbar, which can steer incoming packets toward arbitrary offloads in an arbitrary sequence, also known as offload chaining. A MAT-based ingress module parses incoming packets and determines which offloads they should be forwarded to and in which order based on the flows they are associated with. This information is written in so-called packet descriptors and forwarded along with the packet. PANIC employs a centralized scheduler that contains per-offload hardware-based priority queues called PIFOs [6] in order to reduce the queuing time of high-priority packets when one or more offloads are highly loaded. If an offload cannot process packets at line rate, PANIC deploys multiple units in parallel in order to increase the throughput. The scheduler decides which offload unit the packets should be forwarded to according to a credit-based scheme. The lower the amount of packets buffered in the ingress queue of an offload unit, the more credits this unit has. PANIC applies push-based chaining and load-aware steering. Push-based chaining means that once an offload has finished processing a packet, it can directly

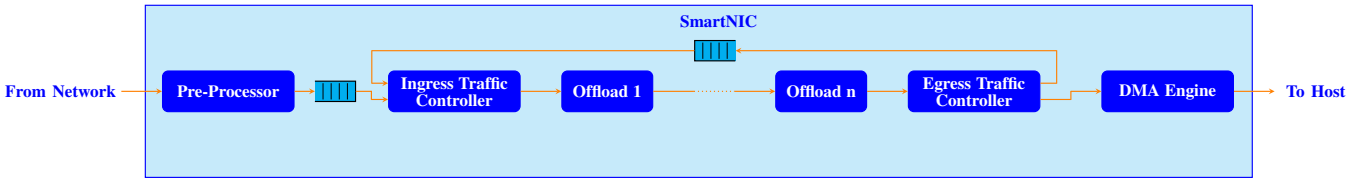

Fig. 1. Block diagram of the architecture of FlexPipe

forward the packet to the next required offload instead of sending it back to the scheduler. This method reduces the per-packet latency and the bandwidth demands on the interconnect. However, if the ingress queue of the offload unit that receives the packet is full, the unit does send the packet back to the scheduler, also known as detour routing. Load-aware steering means that if any offload unit has enough credits, the scheduler forwards incoming packets to it. Otherwise, it buffers them until this is the case.

Although PANIC combines centralized scheduling and push-based chaining in order to allow fast packet forwarding in low-load conditions and load- and priority-aware forwarding in high-load conditions, it has some drawbacks. As offload chains get longer and the detour rate increases, the scheduler can turn into a bottleneck and increase the per-packet latency. Deploying a load balancer at the entry of each offload instead would eliminate the need for a central unit. Another limitation is that the PIFO blocks run at a lower clock rate than the rest of the scheduler, thus reducing the peak throughput that the design can achieve. Moreover, the offload unit which packets are forwarded to is predetermined in the scheduler for offloads that cannot achieve line rate and remains the same until that unit runs out of buffer space. This leads to longer queuing time for all packets as they wait to be fetched by the offload unit. In PANIC the packet descriptors are forwarded ahead of the corresponding packets in the data path. This reduces the available on-chip bandwidth and creates backpressure when packets arrive at the ingress module. Furthermore, although the crossbar provides a high-bandwidth interconnect, it does not scale well in terms of latency and chip area. The scalability issue becomes worse when low-bandwidth offloads need a large number of units in parallel to achieve line rate and each of them is individually attached to the crossbar.

III. Design

In order to address the shortcomings of state-of-the-art SmartNIC designs, we propose FlexPipe, a new packet-processing architecture that provides low-latency packet forwarding in a pipeline of offloads and load-aware packet steering toward offload units of the same type. Figure 1 depicts the architecture of FlexPipe.

The *Pre-Processor*, which operates as a sequence of MATs, parses incoming packets and generates metadata based on the extracted information. The packet metadata are transmitted in parallel along with the packet data, thus keeping the on-chip bandwidth intact. They convey useful information about the packet to the following pipeline modules, such as the packet size, flow type, priority class, offload chain, next required offload and timestamp. The flow type is derived from a specific field or set of fields in the header, such as the source/destination IP address / port, and is mapped to an offload chain. This mapping can be reconfigured in runtime by a software-defined networking (SDN) controller.

The *Ingress Traffic Controller* and the *Egress Traffic Controller* allow packets to be recirculated into the pipeline of offloads. This is a useful feature in scenarios where some packets cannot be forwarded to all offloads in the correct order on the first pass. However, such packets contend with traffic that is just coming from the network, thus reducing the available on-chip bandwidth. The Egress Traffic Controller forwards packets that have been processed by all the required offloads to the *DMA Engine*, which sends them to the host.

The *Offload*s are directly connected to each other and arranged in the sequence that they are most likely to be executed. Figure 2 represents the architecture of an Offload. The *Traffic Splitter* and the *Traffic Arbiter* of each Offload implement the flexible forwarding logic of our SmartNIC.

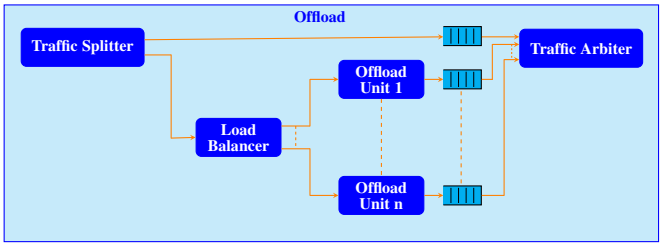

Fig. 2. Block diagram of the architecture of an Offload

The Traffic Splitter checks the field ”next required offload” in the metadata of every incoming packet in order to determine whether it needs to be processed by the corresponding Offload next. If this is the case, then it forwards the packet to the *Load Balancer*. Otherwise, it bypasses the processing unit of the Offload and forwards the packet immediately to the Traffic Arbiter.

If a single *Offload Unit* does not have enough bandwidth to process packets at line rate, then we deploy multiple units in parallel, which share the load of the incoming traffic. The following formula can be used to determine the number of required Offload Units:

$$num_{units} = \left\lceil \frac{w_{pipeline}}{w_{unit}} \cdot \frac{f_{clk,pipeline}}{f_{clk,unit}} + delay_{unit} \right\rceil, \quad (1)$$

where w is the data width, f_{clk} is the clock rate, and $delay$ is the number of not pipelined clock cycles

Since each Offload may process packets of different sizes, simply forwarding each packet to the next Offload Unit in round-robin manner does not distribute the traffic evenly and leads to longer queuing time and higher on-chip latency in Offloads that process the payload of the packet. Hence we employ a load-aware packet steering method. Our Load Balancer monitors the current load of each Offload Unit and forwards incoming packets to the least loaded unit. In order to keep track of the load of each Offload Unit, the Load Balancer increases the load counter of a unit proportionally to the packet size when it starts storing a packet in the input queue of the unit. Then it decrements the counter when the Offload Unit receives a fragment of the packet from its queue.

The Traffic Arbiter contains a highest-demand-first scheduler, which decides whether packets coming directly from the Traffic Splitter or those coming from the processing unit are forwarded to the output based on the amount of traffic that is coming from each source. It can immediately switch from a waiting state to a forwarding state and between different inputs. Incoming packets are temporarily stored in FIFO queues with a size of $16\,\mathrm{kB}$ and the Traffic Arbiter fetches packets from the queue with the highest fill level. A nested round-robin scheduler iterates through all Offload Units in a looping manner and grants units that have finished processing of the whole packet the right to forward packets to the highest-demand-first scheduler. This combination of scheduling algorithms leads to an efficient utilization of the available queues and fair resource sharing among flows with different offload requirements.

In our packet-processing architecture we shift the arbitration from the on-chip interconnect to the Offloads, thus distributing the packet forwarding logic and making it simpler, more efficient and scalable in terms of chip area. In PANIC [16] the number of wires and queues increases quadratically in relation to the total number of Offload Units, while in our design it increases linearly in relation to the number of Offloads. In contrast to PANIC, FlexPipe forwards all packets to all Offloads, which increases the possibility for conflicts and hence the queuing time. However, we compensate this effect by optimizing the load-balancing method. FlexPipe reduces the queuing time at the ingress of the Offload Units by sending packets to the least loaded unit and avoids additional trips to a central unit by decentralizing the Load Balancer. Since we deploy enough Offload Units to achieve line rate in our design, packets never wait for a unit to become available. Hence, unlike PANIC, FlexPipe does not need priority queues.

IV. Evaluation

We implement a prototype of FlexPipe in Verilog and validate it via cycle-accurate RTL simulations in Vivado. We compare it with a SmartNIC that employs a predefined sequence of offloads and PANIC [16] in terms of throughput and latency. FlexPipe, excluding slow offload units, runs at $250\,\mathrm{MHz}$ and has a data width of $512\,\mathrm{bit}$, thus achieving a bandwidth of $128\,\mathrm{Gbit/s}$. All modules communicate with each other via the AXI4-Stream protocol [19].

In order to evaluate our design, we develop a traffic generator and a traffic sink, which interface with FlexPipe via the AXI4-Stream protocol. The traffic generator injects packets of different sizes, allocates each of them a flow ID and configures the mapping of flow IDs to offload chains in the Pre-Processor. When a packet should be sent, a random number generator determines the size of the packet and its flow ID. The traffic sink receives processed packets from FlexPipe and measures its throughput and the per-packet latency. The throughput is the rate at which FlexPipe transmits data to the traffic sink in $\mathrm{Gbit/s}$, while the per-packet latency is the period of time in µs that passes from the moment the Pre-Processor parses the packet until the moment it arrives at the traffic sink.

We integrate six different offloads into FlexPipe and arrange them in the following order: checksum verification (CRC) [20], firewall [21], authentication (SHA-3) [22], en-/decryption (AES) [23], JPEG decoder [24] and RSS [11]. Table II provides the input width, clock rate and number of units required to achieve line rate for each of them. Although the addition of offload units increases the total bandwidth of an offload, it does not reduce its processing time because the load balancer forwards each packet as a whole to a single unit, thus making packet queuing necessary. Hence the lower the per-unit bandwidth of an offload, the longer its processing time.

TABLE II
Input width, clock rate and number of units required to process packets at line rate (equation 1) for each offload that we integrate into FlexPipe

	CRC	Firewall	SHA-3	AES	JPEG decoder	RSS
Input width (bit)	64	8	64	128	32	256
Clock rate (MHz)	250	150	150	250	100	200
Number of units	8	128	39	4	40	3

We use an open-source prototype of PANIC [25] as a baseline for our evaluation. Since the crossbar implementation in PANIC does not support more than eight initiators/targets, we integrate all offload units of the same type into a single module, as in FlexPipe. Another limitation is that the implementation of the priority queues does not support more than two separate queues. This means that no more than two offloads can be deployed in the design. We circumvent this issue by defining a specific offload as the first one in every offload chain, deactivating detour routing and applying push-based chaining for every packet. Moreover, we employ the same load-balancing method, as described in Subsection II-B, but avoid additional trips to the central scheduler by determining the offload unit which a packet should be sent

to at the ingress of the offload. Hence we compare FlexPipe with a faster version of PANIC. We also simulate a version of our design with fixed forwarding logic, which we call StaticPipe and use as a representative of a SmartNIC with a predetermined sequence of offloads. StaticPipe forwards all packets to the processing unit of every offload. All three designs have the same bandwidth (128 Gbit/s) and deploy the same number of offload units for each offload.

We inject 11000 packets of different sizes into each design, each of which is associated with a flow type. We start the measurement after the traffic sink has received 500 packets and stop it after it has received 10500 packets, thus ensuring that the pipeline is full throughout the measurement phase. The mean throughput and per-packet latency values stabilize after at least 2000 packets. When a packet should be dispatched, the traffic generator randomly selects a packet size out of the following set: 64 B, 128 B, 256 B, 512 B, 1024 B and 1500 B. It also randomly assigns one out of four available flow IDs to the packet, which are mapped to different offload chains:

- Flow 1: CRC → firewall → SHA-3 → AES → RSS
- Flow 2: CRC → SHA-3 → JPEG → RSS
- Flow 3: CRC → AES → JPEG
- Flow 4: CRC → firewall

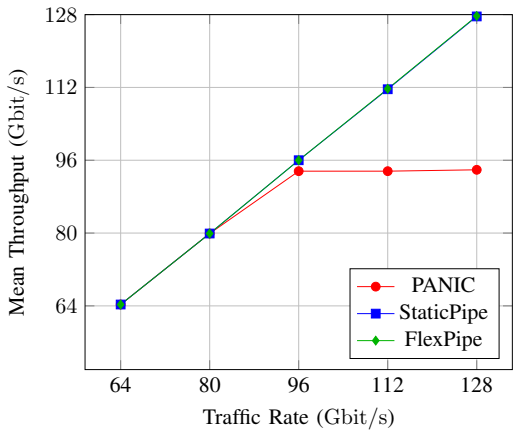

Fig. 3. Mean throughput that is achieved by PANIC, StaticPipe and FlexPipe when receiving traffic corresponding to four flow types at different rates

Figure 3 shows the mean throughput that is achieved by PANIC, StaticPipe and FlexPipe when receiving packets corresponding to the four aforementioned flow types at different traffic rates. All three designs achieve the required throughput when receiving traffic at a rate of 64 Gbit/s and 80 Gbit/s. However, PANIC cannot handle traffic rates of 96 Gbit/s or higher as it achieves a peak throughput of about 94 Gbit/s. This is due to the fact that the priority queue runs at a lower clock rate than the rest of the design and the packet descriptors are forwarded in the data path, as mentioned in Subsection II-B. On the other hand, both StaticPipe and FlexPipe achieve the required throughput when receiving traffic up to a rate of 128 Gbit/s, which is equal to the total on-chip bandwidth.

Figure 4 shows the mean latency for packets of different sizes associated with the four aforementioned flow types that

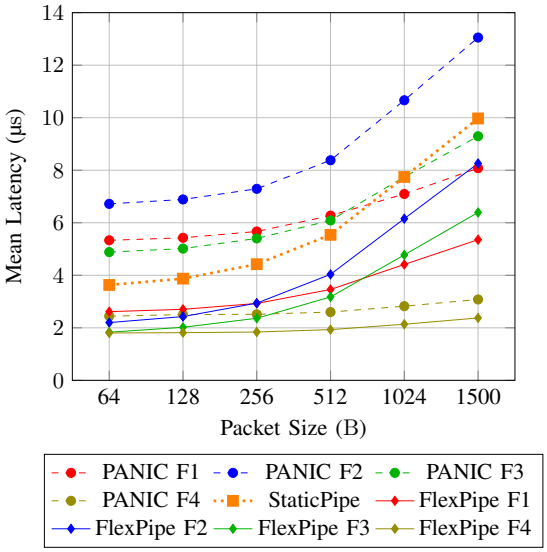

Fig. 4. Mean latency for packets associated with four flow types that is measured in PANIC, StaticPipe and FlexPipe when receiving traffic at a rate of 90 Gbit/s

is measured in PANIC, StaticPipe and FlexPipe when receiving traffic at a rate of 90 Gbit/s. We generate traffic at a rate lower than 94 Gbit/s in order to allow a fair comparison with PANIC regarding latency. Since StaticPipe forwards all packets to all offloads, the flow type makes no difference. FlexPipe processes packets on average 1.6x faster for flow 1, 1.4x faster for flow 2, 1.7x faster for flow 3 and 2.9x faster for flow 4 compared with StaticPipe. As the processing latency of the bypassed offloads increases, the latency reduction becomes more significant. Furthermore, we can notice that FlexPipe achieves a lower mean latency than PANIC on a per-flow basis for packets of any size due to employing a more optimal load balancer. FlexPipe processes packets on average 1.6x faster for flow 1, 2.5x faster for flow 2, 1.9x faster for flow 3 and 1.3x faster for flow 4 compared with PANIC. The average difference between the maximum latency value and the minimum latency value is 6.605 µs for PANIC, 1.422 µs for StaticPipe and 1.404 µs for FlexPipe.

Figure 5 shows the mean latency for packets corresponding to flows that require packet recirculation in FlexPipe (different sequence for the above-mentioned offload chains) in relation to the proportion of recirculated packets for both PANIC and FlexPipe when receiving traffic at different rates. We do not consider StaticPipe as it does not support packet recirculation. Since recirculated packets require additional bandwidth, the amount of packets that FlexPipe can recirculate without congesting the pipeline depends on the traffic rate. As shown in the figure, when receiving traffic at a rate equal to 70 % of the total bandwidth (90 Gbit/s), FlexPipe cannot recirculate more than around 43 % of the incoming packets due to the limited on-chip bandwidth. However, if the traffic comes from the data link at such a high rate, then the amount of packets that FlexPipe is able to recirculate can be raised by increasing the

data width or the clock rate. This, in turn, requires additional costs in terms of chip area or power consumption. Moreover, we can observe that if the available bandwidth is not fully utilized, FlexPipe processes packets faster than PANIC even when it recirculates a large proportion of the traffic.

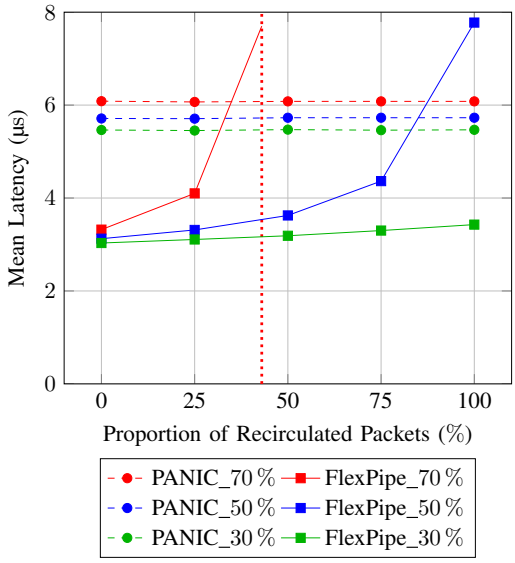

Fig. 5. Mean per-packet latency that is measured in PANIC and FlexPipe when receiving traffic at different rates (expressed as percentage in relation to the total bandwidth) corresponding to flows that require packet recirculation

V. Conclusion and Outlook

In order to tackle the challenges faced by state-of-the-art SmartNIC designs, we propose FlexPipe, a fast, flexible and scalable packet-processing architecture that can handle high traffic rates. FlexPipe provides efficient and runtime-reconfigurable packet forwarding in a pipeline of offloads, as well as load-aware packet steering toward multiple offload units of the same type. We implement a prototype of FlexPipe with six different offloads in Verilog and validate it via cycle-accurate RTL simulations. Our evaluation results demonstrate that our design can process packets of arbitrary sizes with different offload requirements at a traffic rate of $90\,\mathrm{Gbit/s}$ on average 1.9x faster than a SmartNIC with a predefined sequence of offloads and 1.8x faster than PANIC [16], a flexible state-of-the-art SmartNIC. We plan to extend FlexPipe by introducing priority queues, in order to reduce the waiting time for high-priority packets in scenarios where offloads do not have enough bandwidth to handle temporary traffic bursts.

References

[1] J. Ang *et al.*, *Decadal Plan for Semiconductors*. Semiconductor Research Corporation, 2021.

[2] Missing Link Electronics, "Network Function Accelerators, FACs, NICs and SmartNICs," https://www.missinglinkelectronics.com/www/www/index.php?option=com_content&view=category&layout=blog&id=141&Itemid=310.

[3] D. Molka, D. Hackenberg, and R. Schöne, "Main memory and cache performance of Intel Sandy Bridge and AMD Bulldozer," in *Proceedings of the Workshop on Memory Systems Performance and Correctness*, ser. MSPC '14. New York, NY, USA: Association for Computing Machinery, 2014.

[4] H. Rauchfuss, T. Wild, and A. Herkersdorf, "A network interface card architecture for I/O virtualization in embedded systems," in *Proceedings of the 2nd Conference on I/O Virtualization*, ser. WIOV'10. USA: USENIX Association, 2010, p. 2.

[5] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, p. 87–95, jul 2014.

[6] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, S.-T. Chuang, A. Agrawal, H. Balakrishnan, T. Edsall, S. Katti, and N. McKeown, "Programmable packet scheduling at line rate," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 44–57.

[7] M. T. Arashloo, A. Lavrov, M. Ghobadi, J. Rexford, D. Walker, and D. Wentzlaff, "Enabling programmable transport protocols in High-Speed NICs," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 93–109.

[8] M. S. Brunella, G. Belocchi, M. Bonola, S. Pontarelli, G. Siracusano, G. Bianchi, A. Cammarano, A. Palumbo, L. Petrucci, and R. Bifulco, "hXDP: Efficient software packet processing on FPGA NICs," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 973–990.

[9] T. Hoefler, S. Di Girolamo, K. Taranov, R. E. Grant, and R. Brightwell, "sPIN: High-performance streaming processing in the network," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, 2017.

[10] D. Sidler, Z. István, and G. Alonso, "Low-latency TCP/IP stack for data center applications," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, 2016, pp. 1–4.

[11] A. Oeldemann, F. Biersack, T. Wild, and A. Herkersdorf, "Inter-server RSS: Extending receive side scaling for inter-server workload distribution," in *2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2020, pp. 46–53.

[12] D. Sidler, Z. Wang, M. Chiosa, A. Kulkarni, and G. Alonso, "StRoM: Smart remote memory," in *Proceedings of the Fifteenth European Conference on Computer Systems*, ser. EuroSys '20. New York, NY, USA: Association for Computing Machinery, 2020.

[13] S. Ibanez, A. Mallery, S. Arslan, T. Jepsen, M. Shahbaz, C. Kim, and N. McKeown, "The nanoPU: A nanosecond network stack for datacenters," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, Jul. 2021, pp. 239–256.

[14] R. Bassoli *et al.*, *Analysis of 6G architectural enablers' applicability and initial technological solutions*. Hexa-X, 2022.

[15] Nvidia, "NVIDIA BlueField-3 DPU," https://resources.nvidia.com/en-us-accelerated-networking-resource-library/datasheet-nvidia-bluefield?lx=LbHvpR&topic=networking-cloud.

[16] J. Lin, K. Patel, B. E. Stephens, A. Sivaraman, and A. Akella, "PANIC: A High-Performance programmable NIC for multi-tenant networks," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 243–259.

[17] S. Di Girolamo, A. Kurth, A. Calotoiu, T. Benz, T. Schneider, J. Beránek, L. Benini, and T. Hoefler, "A RISC-V in-network accelerator for flexible high-performance low-power packet processing," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 958–971.

[18] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, and S. Licking, "Packet transactions: High-level programming for line-rate switches," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1528.

[19] Arm Developer, "AMBA 4 AXI4-Stream Protocol Specification," https://developer.arm.com/documentation/ihi0051/a.

[20] "Ultimate CRC," https://opencores.org/projects/ultimate_crc.

[21] "RMII Firewall FPGA," https://github.com/jakubcabal/rmii-firewall-fpga.

[22] "SHA3 (KECCAK)," https://opencores.org/projects/sha3.

[23] "AES," https://opencores.org/projects/tiny_aes.

[24] "High throughput JPEG decoder," https://github.com/ultraembedded/core_jpeg.

[25] "PANIC," https://bitbucket.org/uw-madison-networking-research/panic_osdi20_artifact/src/master/.