Data Stream Processing in Software Defined Networks: Perspectives and Challenges

Alessandra Fais*, Gregorio Procissi*, Stefano Giordano*, Francesco Oppedisano†
*Dipartimento di Ingegneria Dell'Informazione
Università di Pisa
Email: alessandra.fais@phd.unipi.it, {gregorio.procissi, stefano.giordano}@unipi.it
†NetResults S.r.l.
Email: oppedisano@netresults.it

***Abstract*—The new paradigm of network softwarization is pushing programmability and programming abstractions as key elements at different levels on both data and control planes of the network. However, the availability of programmable abstractions and devices *per se* is not sufficient to guarantee high-speed processing rates to network applications without the adoption of efficient programming models and accelerating methodologies. The paper discusses the possible sources of computation bottlenecks and proposes *Data Stream Processing (DaSP)* as a viable programming model for a unified scheme of accelerating data elaboration over both fast and slow data paths. Perspectives and implication of the adoption of DaSP are presented along with possible research directions.**

I. Introduction

The introduction of Software Defined Networking (SDN) is gradually modifying the approach to network processing as well as the design principles of network applications. This trend was first initiated by OpenFlow [1], in which the centralized control plane instructs network nodes forwarding by filling match/action tables. In the late years, the type of operations requested to the nodes has been significantly extended and new models have been proposed to delegate bigger and bigger portions of the processing tasks.

This provides the basis for the so-called concepts of network *softwarization* and *programmability*, in which network nodes become fully programmable elements for running a wide range of functions, such as switching, routing, shaping, load balancers, firewalls, and so on. This new paradigm assumes that the network control plane becomes the *network operating system (NOS)* on top of which network applications run. Obviously, the deployment of the whole framework requires, on one hand the availability of programmable hardware and, on the other hand, the design and implementation of proper *abstractions* and *languages* for data plane programming.

In addition, the delegation of processing functions to the data plane comes with stringent requirements in terms of performance and scalability. In general, this requires that traffic data is elaborated in place, as they arrive, by means of efficient (and possibly approximated) algorithms and new programming models. This, in turn, calls for breaking up functions into elementary primitives. As a result, network applications are built around the combination of micro-NFs that are tailored to the software/hardware platforms available on the data plane and are instantiated into the network nodes and/or in the controller.

This work aims at proposing a new approach for designing scalable and high-performance network applications in the SDN context. The approach is based on the adoption of the *Data Stream Processing* (DaSP) programming model to accelerate both data plane and control plane tasks. Hence, in the following the main challenges are discussed and new strategies are explored. We propose to use DaSP in combination with a two-layer structure of network services in the attempt of providing a unified and structured approach at different levels of the network programming hierarchy. As such, this exploratory work attempts to be a starting point for discussion and for future works in this area of research.

The rest of the paper is organized as follows. Section II provides the technical background and related works on network programmability and SDN. Section III describes the new multi-layer processing view of network functions. Section IV introduces the stream processing computational model and provides a detailed presentation of the methodology proposed to accelerate data elaboration at both the control plane and data plane levels. Finally, Section V sums up on the proposed techniques, and addresses the most promising research directions that emerge from the whole discussion.

II. Background

The natural evolution of Software Defined Networking and Network Function Virtualization is represented by the *softwarization* of networks, in which specialized hardware is replaced by general-purpose platforms and network functions are implemented as software applications that can be instantiated through the network control plane. In fact, in the new network model, network applications are nothing more than instantiations of SDN controllers that, on one hand, enforce the operations of programmable nodes on the data plane and, on the other hand, are in charge of the second layer of processing after receiving the early stage output from the data plane.

This novel processing paradigm raises a number of issues from different points of view. First, it calls for the availability of proper APIs and programming abstractions to allow the control plane to configure and instruct the data plane processing nodes. Second, the data plane itself must be able to perform early stage data elaboration at (or nearly at) network line rate. Third, the control plane itself must be able to cope with the possibly significant computation burden imposed by different use-cases – as an example, a system for Deep Packet Inspection (DPI) or, even more complex, Security Information and Event Management (SIEM).

Programmable Abstractions. The very first approach to network programmability in SDNs came from the switching domain, with the simple match-and-action abstraction brought by OpenFlow [1]. While still quite limited, the OpenFlow abstraction was a winning solution in forcing manufacturers and vendors to start considering a balancing view of fully programmable networks and the necessary pragmatism for real-world deployment. In the last few years, the P4 [2] language has widely extended the concept of node programmability and established itself as the successor of OpenFlow as the *de facto* standard language for programmable switches, such as [3]. Other sophisticated abstractions that originate in the academic community include OpenState [4], Open Packet Processor (OPP) [5] and FlowBlaze [6]. While the above approaches are mainly targeted at programmable hardware nodes, a parallel set of programming abstractions are also available for traditional software approaches. Enif–lang [7] extends the seminal programming platforms of [8], [9] to provide a simple Haskell-based tool for programming generic network functions on the Linux OS. Approaches based on imperative language are Pyretic [10] (from the Frenetic [11] family of network programming languages) and Streamline [12]. Vector Programming Processing (VPP) [13] is also a recent proposal that originates from the Cisco world for switch/router functionality. Finally, a very promising approach is given by eBPF (extended Berkeley Packet Filter) [14]. eBPF is natively supported in the Linux OS kernel and is gaining more and more popularity for its high flexibility and high performance over low cost platforms.

High-Speed Traffic Handling. As introduced above, the technological maturity reached by off-the-shelf hardware platforms makes general-purpose servers reasonable candidates for implementing real network nodes. As a result, a considerable number of approaches have been proposed in the literature for effective handling of packets at sustained rates. At the capture level, the main limitations of general-purpose operating systems have been thoroughly addressed and a complete review of possible solutions is contained in [15], [16], [17].

PF_RING [18] was historically the first accelerated tool for packet capturing over 1 Gbps links. Later on, PF_RING ZC (Zero Copy) [19] and Netmap [20] significantly pushed forward the performance of capture engines up to multi-gigabit line rate by memory mapping the ring descriptors of Network Interface Cards (NICs) at the user space. Intel's DPDK [21] and PFQ [9] added room for packet processing beside high-speed capture rates. Notably, it is worth mentioning that the Linux kernel itself has significantly improved its capture performance with the adoption of the TPACKET (version 3) socket [22], and that also the `pcap` library has been recently extended [23] to support packet fan-out and enable multi-core processing.

Network Applications Design. The availability of efficient capture engines for fast packet handling is definitely a necessary condition to provide programmability to data plane elements. Unfortunately, it is still not sufficient if not coupled with an efficient design of the processing functions. In fact, the high rate sustained by the accelerated engines may even be harmful against inefficient functions that may get overwhelmed by a huge amount of traffic that cannot be "consumed" within a reasonable time.

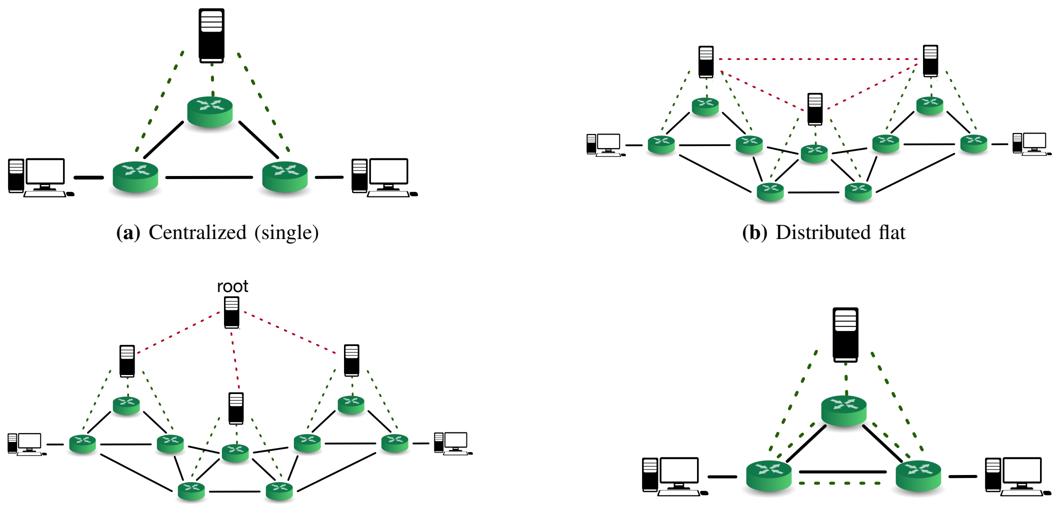
PFQ [9], DPDK [21] and eBPF over XDP [14] integrate the concept of packet processing beside mere fast capturing. We deem that this idea should be further extended by applying different programming paradigms that come from the Computer Science theory. In particular, we focus our attention to *Data Stream Processing* (DaSP) as it specifically addresses real-time processing of data in the form of *streams*. So far, DaSP frameworks have been typically applied for developing streaming applications in traditional distributed environments (i.e. homogeneous clusters) and they achieve platform independence by relying on the Java Virtual Machine (JVM). As such, they seem quite well tailored for accelerating the slow path of network applications (i.e., the control plane portion). However, the use of innovative software libraries such as WindFlow [24] may extend the domain of application of DaSP to the data plane.

The use of DaSP in designing network applications as a whole is the main contribution of the paper: as such, the main concepts, the major existing incarnations and their application in the SDN scenario will be thoroughly elaborated upon in section IV.

III. Multi–Layer Processing

Scalability is an important issue to be addressed in SDN, due to the logically centralized architecture characterizing this kind of networks. On one hand, moving the control over all network elements in the controller entity simplifies the management of large networks, enabling at the same time a faster innovation. On the other hand, the control plane may become a scalability bottleneck due to factors such as the communication delay between controller and switches and the capacity of the controller. The first one is a *latency* problem, and depends on the distance between network devices and controller. This affects the flow setup time, increasing the delay for flow rule addition, deletion or update in switch flow tables. Moreover, the collection of information regarding configuration, statistics and capabilities from network devices is impacted as well. The increased latency could potentially lead to congestion problems in both the control plane and the data plane. The *capacity* aspect is related to both the number of data plane devices and the quantity of events and requests handled for each device in the control plane.

Therefore, the architecture of the controller is one of the most important factors which has to be taken into account to study scalability issues in this context. Several works are available in literature, which aim at comparing the scalability



(a) Centralized (single) **(b)** Distributed flat **(c)** Distributed hierarchical **(d)** Hybrid

Figure 1: Topology-related controller architectures. Network devices are represented in green, while controller instances are depicted in black. Solid black lines represent two-way data path communications among network devices. Dashed green lines represent two-way control path communications between controller and network devices. Dashed red lines represent controller-to-controller communications. Notice that, in hybrid designs, network devices also participate in network control.

performance of different SDN controller architectures. In [25], Yang et al. evaluated scalability in the SDN control plane by simulating the behavior of five different control plane architectures, organized according to the proposal of Karakus et al. in [26]. Here, control plane architectures are classified based on two main approaches, which can be adopted to improve the controller scalability: *topology-related* approaches and *mechanism-related* ones. The first class of solutions includes single (centralized) controller designs and distributed ones, which can in turn follow a flat, hierarchical or hybrid structure. Figure 1 illustrates these four topology-based approaches. In the second group it is possible to find approaches that exploit parallelism in multi-core systems, and others which rely on control plane routing scheme optimizations (e.g. new strategies to reduce the number of events to be processed) [25], [26].

In the context described so far, network applications run on top of the controller, hence implementing network functionalities in software. Given the previous considerations on scalability in SDN controllers, the approach proposed in this paper tackles the problem of designing highly scalable network applications by leveraging a *two-layer processing model*. The studies performed so far on the relationship between control plane architectures and scalability act as a starting point for our discussion.

The experiments carried out by Yang et al. in [25] show that hierarchical distributed controllers are the ones capable of achieving the best results in terms of scalability, if compared to the other approaches listed above. In this design, controllers are organized by following a general tree structure, as in Figure 1c: the leaves (in the bottom layer) are controller instances in charge of directly managing only a limited portion of the network, while the root controller provides a complete view of the network by aggregating the local controllers' views of each sub-network.

This philosophy can be readily lent to the data plane by proposing a multi-layer approach as advocated in the softwarized network context discussed in Section II. The computations are designed in a way that a first coarse-grained processing could be performed directly at the data plane level, while a second stage of more fine-grained computations will be carried out in the control plane on the partial results received from the data plane. The controller always maintains control over all network nodes, but its workload is reduced thanks to the delegation of part of the processing to the underlying devices. The following points highlight the key factors behind this new class of *multi-layer network services*.

- *Controller workload reduction*: one way of improving the performance scalability of a single controller module is to reduce its workload. A solution could be to redesign its architecture as a distributed system, where the workload is partitioned and distributed to load-balanced entities. Another approach resides in the parallelization of each controller instance. Of course, the two solutions could be also exploited in conjunction. In this way, it is possible to obtain low latencies in the controller-switches communications and increased control plane capacity. Guaranteeing high-throughput and low-latency communications between data plane and control plane is of paramount importance for supporting our two-layer model for NFs.
- *Programmable nature of network devices*: general-purpose network nodes are becoming quite popular in modern networks. The possibility to (re-)configure these devices in a programmatic way offers the ability to

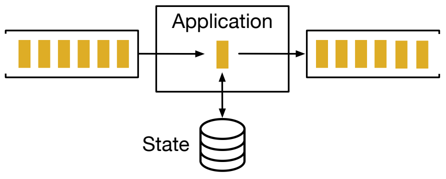


Figure 2: Stream processing application. Stream elements are represented in yellow. The application receives a stream in input and performs some computation on each received item, producing a stream of results in output. Operators can be *stateless* or *stateful*, in case they maintain a state containing information on the data processed so far.

adapt the network operations to each specific network deployment context. This is another key enabler. In fact, a low-level custom processing stage in the data plane can be implemented only if the adopted network devices offer the right degree of flexibility and programmability.

In the following (Section IV), we shall provide more details on how a scalable multi-layer network function could be implemented, considering both the first stage of processing at the data plane level and the second stage of computation at the control plane level.

IV. Stream Processing Acceleration

The Data Stream Processing (DaSP) computational model is gaining interest from the research community as a solution oriented to the always increasing need for continuous processing and real-time analysis of data streams. Different DaSP frameworks have been designed to offer suitable programming abstractions for implementing streaming applications. In general, such applications must be capable of performing real-time computations *on-the-fly*, as new data appear on the input streams. Figure 2 shows the general model for a streaming application. For its intrinsic characteristics, DaSP appears to be a good candidate for accelerating network functions (NFs) at both levels in the two-layer model. In the following, we will show the principal points which characterize the stream-based computational model and how they relate to our specific use-case.

A. *Working on streams*

In the DaSP context, the word *stream* identifies unbounded sequences of records of attributes in the form of key-value pairs, called *tuples*. Stream items continuously arrive at very high speed, and possibly with irregular and time-varying rates, and their significance decays over time. Implementing real-time computations over this kind of data imposes to find the correct strategies to deal with the streams' peculiarities, and this complexity needs to be managed in the streaming program.

In the proposed multi-layer NF processing, streams can be identified at both levels. In detail:

- *First stage (data plane level)*: streams of packets are received at high-speed (line rate in the order of Gbps) from one or many NICs.
- *Second stage (control plane level)*: streams of results pre-processed in the data plane are received to be further analyzed.

At the data plane level, a generic first-stage streaming application processes packets – special kind of tuples – and produces one or more streams of partial results in output. These records are sent towards some control plane streaming application, which applies further processing to the received items, producing one or more streams of final results in output. At both levels, operators in the NFs optionally maintain some stream history about past data received in input. This can be done by implementing a *state* using probabilistic data structures (e.g. bloom filters, counting bloom filters, sketches) or saving the last received items in a buffer (*window*).

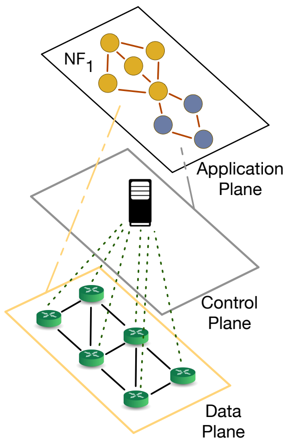
B. *Performance constraints*

In order to guarantee a real-time kind of processing, tight constraints must be placed in terms of *bandwidth* (output rate) and *latency* (response time). For this reason, new stream processing approaches have been proposed, as opposed to the more traditional batch computing model. The latter cannot be adopted to implement efficient real-time computations on streams, due to the unfeasibility to store the whole stream of data before starting the computation. Indeed, this is the typical approach adopted to implement offline computations, after a first phase where large amount of data are collected over a period of time and stored (in databases or distributed file systems).

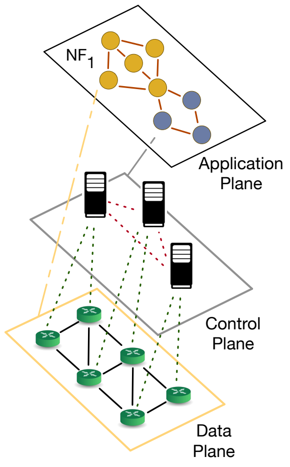
Both high-throughput and low-latency are inherent requirements of multi-layer NFs. At the data plane level, the real-time performance requirements are more stringent with respect to the ones imposed at the control plane. This is directly related to the wire speed at which packets are received from the network cards. In the controller, among others, we are interested in real-time analysis and event detection kind of processing, in order to gain the ability to make prompt decisions regarding network monitoring or security aspects – to name two very hot domains from the research point of view. This slight difference in the real-time performance requirements of the two cooperating parts of each NF can be used in our favor. In fact, this allows us to exploit different kinds of DaSP frameworks, capable of achieving different results in terms of performance depending on their respective architecture and the underlying execution environment.

C. *Computational model*

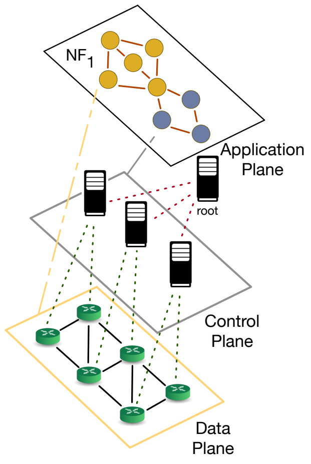
In general, real-time DaSP frameworks speed up the execution by exploiting parallelism. The underlying adopted computational model is typically a Data-Flow graph. Hence, a streaming application is structured as a graph of simple functionalities composed together to provide the complete computation. These *operators* or *transformations* – as the core functionalities are typically called in the DaSP context – correspond to the graph's nodes. They are connected together by arcs which model streams, defining the execution flow through data dependencies in the graph.



(a) Multi-layer NF with centralized control plane.



(b) Multi-layer NF with distributed flat control plane.



(c) Multi-layer NF with distributed hierarchical control plane.

Figure 3: Complete design schema for multi-layer DaSP-based network functions. A generic network function (called NF_1 in the three sub-figures) has been implemented using DaSP frameworks. The deriving Data-Flow graph will be executed by choosing the most convenient deployment on the nodes in the data plane and the controller instances. In this example, yellow operators are mapped on some network device in the data plane, while the remaining bluish operators are executed in the control plane. The controller can be centralized (3a), or distributed (3b, 3c).

Structuring the multi-layer network services as combinations of micro-functionalities, according to the Data-Flow model, appears to be a good strategy to achieve the desired flexibility and performance. Once the application has been decomposed as a Data-Flow graph of core micro-NFs, the second step would be to identify which operators are convenient to run at the data plane level and which ones can be run at the control plane level. Moreover, innovative strategies to find an efficient displacement of the NF graph's operators on the network devices and controller instances are required. This is a problem of mapping a graph (the Data-Flow graph of micro-NFs) to another graph's sub-graph (the network graph of data plane devices and controller instances). Since the sub-graph isomorphism problem is known to be NP-complete, some good heuristics to obtain sub-optimal approximate solutions for this mapping are required. Factors such as link costs, distance between a network device and a controller instance, and so on can be taken into account in order to minimize the latency and maximize the bandwidth of the system (i.e. the graph of streaming operators implementing the NF). Possible ways to address this problem could include the use of techniques borrowed from constraint programming and operations research. This part will be addressed in future works.

D. Execution environment

DaSP frameworks are tools designed to ease the development of efficient streaming applications. They provide suitable abstractions which mask the complexities related to the management of streams.

Mainstream Stream Processing Engines (SPEs) – Apache Storm [27], Flink [28], Spark Streaming [29], to name a few – target distributed systems of homogeneous machines (*clusters*). They typically rely on the Java Virtual Machine (JVM) processing environment and increase the application throughput as needed by scaling out the computation over the cluster's computing nodes. Therefore, many Data-Flow graphs can be executed in parallel, and operators in each graph can be parallelized as well, by exploiting stateless replication and stream partitioning [30]. The adoption of the JVM facilitates obtaining platform independence, at the price of processing overheads induced by factors such as garbage collection and data serialization and de-serialization. Thus, in these state-of-the-art solutions, the design of efficient data accesses, and as a consequence the overall performance, is limited by the presence of overheads related to the run-time and the support to distributed systems. For this reasons, mainstream DaSP frameworks are not capable of efficiently exploiting the full hardware capabilities offered by single scale-up servers equipped with multi-core CPUs and co-processors (e.g. GPUs and FPGAs), as discussed in [31], [32].

Alternative DaSP frameworks exist, such as the Wind-Flow [24] library, which offers high-level abstractions to implement efficient stream processing targeting single multi-core machines, with integrated support for GPUs. This parallel library is implemented using C++17 constructs and relies on the Data-Flow graph model to implement streaming programs, as extensively discussed in [30].

The stream processing frameworks introduced above are good candidates for implementing multi-layer modular NFs, exploiting the Data-Flow graph computational model. They differ in terms of performance (bandwidth, latency) they are able to achieve, and they target different types of systems (distributed or single node scenarios). Nevertheless, this allows us to exploit a combination of various DaSP frameworks to implement multi-layer network applications, leveraging the strengths of each SPE depending on which stage the streaming functionalities we are implementing will be located.

More in detail, this translates in the following considerations:

- *First processing stage*: at the data plane level, general-purpose network devices can be commodity PCs equipped with multi-core CPUs and possibly co-processors capabilities. In this context, a DaSP library such as WindFlow appears to be a good candidate to implement streaming computations optimized to efficiently exploit the hardware resources of such scale-up machines.
- *Second processing stage*: at the control plane level, different architectures have to be considered, as depicted in Figure 1. In case of single centralized controller, again a DaSP framework with characteristics similar to WindFlow can be leveraged. On the contrary, if the control plane has a distributed architecture, the proper choice would be to adopt one of the mainstream SPEs targeting distributed systems.

The design of scalable multi-layer NFs leveraging DaSP techniques, implemented following the methodology described in this section, is summarized in Figure 3.

V. Conclusions

The paper addresses possible performance issues of network applications and proposes Data Stream Processing (DaSP) as a promising and unifying approach to bring coding efficiency at both data plane and control plane levels in the SDN scenario. A new hierarchical scheme of applications is described, as well as the specific areas that can conveniently take advantage of processing acceleration by means of DaSP frameworks. The implications of the proposed approach are discussed together with its perspectives and the research directions that can be worth exploring.

Acknowledgments

This work was partially supported by the Italian Ministry of Education and Research (MIUR) in the framework of the CrossLab project (Departments of Excellence) and by the project PRA 2018–2019 Research Project "CONCEPT – COmmunication and Networking for vehicular CybEr-Physical sysTems" funded by the University of Pisa, Italy.

References

[1] N. McKeown *et al.*, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, mar 2008.

[2] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, jul 2014.

[3] "Tofino 2: Second-generation of World's fastest P4-programmable Ethernet switch ASICs," https://www.barefootnetworks.com/products/brief-tofino-2/, (Accessed on June 26, 2020).

[4] G. Bianchi *et al.*, "Openstate: Programming platform-independent stateful openflow applications inside the switch," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 44–51, apr 2014.

[5] M. Bonola *et al.*, "Implementing advanced network functions for datacenters with stateful programmable data planes," in *LANMAN 2017*, June 2017, pp. 1–6.

[6] S. Pontarelli *et al.*, "Flowblaze: Stateful packet processing in hardware," in *NSDI 19).* Boston, MA: USENIX Association, Feb. 2019, pp. 531–548.

[7] N. Bonelli, S. Giordano, and G. Procissi, "Enif-lang: A specialized language for programming network functions on commodity hardware," *Journal of Sensor and Actuator Networks*, vol. 7, p. 34, 08 2018.

[8] N. Bonelli, S. Giordano, G. Procissi, and L. Abeni, "A purely functional approach to packet processing," in *ANCS '14*. ACM, 2014, pp. 219–230.

[9] N. Bonelli, S. Giordano, and G. Procissi, "Network traffic processing with PFQ," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 6, pp. 1819–1833, June 2016.

[10] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing software-defined networks," in *NSDI 2013*, 2013, pp. 1–14.

[11] "The Frenetic Project," http://frenetic-lang.org/, (Accessed on 2018-06-15).

[12] W. de Bruijn *et al.*, "Application-Tailored I/O with Streamline," *ACM Trans. Comput. Syst.*, vol. 29, no. 2, pp. 6:1–6:33, may 2011.

[13] "VPP," https://wiki.fd.io/view/VPP, (Accessed on 2018-06-15).

[14] "extended Berkeley Packet Filter," https://www.iovisor.org/technology/ebpf, (Accessed on June 24, 2020).

[15] L. Braun *et al.*, "Comparing and improving current packet capturing solutions based on commodity hardware," in *IMC '10*. ACM, 2010, pp. 206–217.

[16] V. Moreno *et al.*, "Commodity packet capture engines: Tutorial, cookbook and applicability," *Communications Surveys Tutorials, IEEE*, vol. 17, no. 3, pp. 1364–1390, thirdquarter 2015.

[17] S. Gallenmüller *et al.*, "Comparison of frameworks for high-performance packet io," in *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 29–38.

[18] F. Fusco and L. Deri, "High speed network traffic analysis with commodity multi–core systems," in *Proc. of IMC '10*. ACM, 2010, pp. 218–224.

[19] L. Deri, "PF_RING ZC (Zero Copy)," http://www.ntop.org/products/packet-capture/pf_ring/pf_ring-zc-zero-copy/, (Accessed on 2018-06-15).

[20] L. Rizzo, "Netmap: a novel framework for fast packet i/o," in *Proc. of USENIX ATC'2012*. USENIX Association, 2012, pp. 1–12.

[21] "DPDK," http://dpdk.org, (Accessed on June 15, 2020).

[22] Linux Kernel Contributors, "Packet_mmap," https://www.kernel.org/doc/Documentation/networking/packet_mmap.txt, (Accessed on 2018-06-15).

[23] N. Bonelli, F. D. Vigna, S. Giordano, and G. Procissi, "Packet fan–out extension for the pcap library," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2018.

[24] "WindFlow: A C++17 Data Stream Processing Parallel Library for Multicores and GPUs," https://paragroup.github.io/WindFlow/, (Accessed on May 24, 2020).

[25] H. Yang, J. Ivey, and G. F. Riley, "Scalability comparison of sdn control plane architectures based on simulations," in *IPCCC 2017*, 2017, pp. 1–8.

[26] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in software-defined networking (sdn)," *Computer Networks*, vol. 112, pp. 279 – 293, 2017.

[27] "Apache Storm," https://storm.apache.org/, (Accessed on June 22, 2020).

[28] "Apache Flink," https://flink.apache.org/, (Accessed on June 22, 2020).

[29] "Apache Spark Streaming," https://spark.apache.org/streaming/, (Accessed on June 22, 2020).

[30] A. Fais, "Benchmarking Data Stream Processing Frameworks on Multicores," Master's thesis, Università di Pisa, Oct. 2019.

[31] S. Zeuch, B. D. Monte, J. Karimov, C. Lutz, M. Renz, J. Traub, S. Breß, T. Rabl, and V. Markl, "Analyzing Efficient Stream Processing on Modern Hardware," *Proceedings of the VLDB Endowment*, vol. 12, no. 5, pp. 516–530, 2019.

[32] S. Zhang, B. He, D. Dahlmeier, A. C. Zhou, and T. Heinze, "Revisiting the Design of Data Stream Processing Systems on Multi-Core Processors," *IEEE ICDE 2017*, 2017.