On the Design of Fast and Scalable Network Applications Through Data Stream Processing

Alessandra Fais
PhD Student
Dipartimento di Ingegneria
Dell'Informazione
Università di Pisa
Pisa, Italy
Email: alessandra.fais@phd.unipi.it

Stefano Giordano
Supervisor
Dipartimento di Ingegneria
Dell'Informazione
Università di Pisa
Pisa, Italy
Email: stefano.giordano@unipi.it

Gregorio Procissi
Supervisor
Dipartimento di Ingegneria
Dell'Informazione
Università di Pisa
Pisa, Italy
Email: gregorio.procissi@unipi.it

***Abstract*—The current trend characterizing new generation networks is to accommodate a variety of services on the same shared infrastructure. In this context, two main aspects are vital for network operators. On one hand, there is the need for proper mechanisms to rapidly and easily (re-)configure the network, to adapt to its changing conditions over time. On the other hand, applications for continuous network monitoring become essential to detect security problems or performance degradation. This is fundamental to guarantee the Quality of Service (QoS) requirements for every running service. This paper aims at proposing a new framework for the implementation of fast and scalable applications for real-time continuous network monitoring and data analysis. High-level abstractions will be provided to the network programmer, and the Data Stream Processing computational model will be exploited to improve performance. The framework architecture is described, along with the implementation design for this kind of applications. The main challenges are presented, with proposed solutions to tackle them. Finally, the current status of the work is discussed, along with its future developments.**

I. Introduction

Traditional networks were typically conceived as pure physical communication media dedicated to a well defined service. This concept evolved during the last years, up to today's network infrastructures which accommodate a variety of services at the same time. The Software Defined Networking (SDN) evolution meets the need for mechanisms to (re-)configure the network in a rapid and easy way to adapt to new requests for service (de-)activation. Moreover, continuous monitoring for tracking security-related events or performance degradation becomes vital to guarantee the Quality of Service (QoS) requirements as network conditions evolve over time. Network operators therefore need to collect packets and any kind of valuable measure from the network, and to perform real-time analysis of these streaming data for preserving network security and performance. The latter aspect is still the most difficult to address [14], and the one we aim at improving with the proposed framework.

Nowadays, the need for new strategies devoted to continuous and real-time analysis of streaming data concerns several domains (e.g., networking, smart cities, smart mobility, smart logistics). The increased interest in solving this kind of problems boosted the research activity in the Data Stream Processing (DaSP) field. DaSP frameworks began to appear as tools to implement in an easier way efficient streaming computations. General-purpose streaming applications are modeled as data-flow graphs, where vertices correspond to core functionalities (and are also called *operators* or *transformations)*, and arcs model streams. Transformations are applied on the stream items (called *tuples*) as they flow through the graph, until the processing is concluded in the final stage (i.e. sink). The abstractions provided by modern DasP frameworks hide

- complexities related to stream handling (e.g, dealing with infinite sequences of items, irregular rates, decay of data significance as time goes on);
- aspects related to the management of parallelism (e.g., implementing communication patterns among operators, dealing with synchronization problems).

Therefore, the programmer needs to specify the logic of each operator, and relies on the framework for all the other aspects.

Different DaSP frameworks are characterized by the API exposed to the programmer (and the provided level of abstraction), and the runtime they are based on. These factors influence both expressiveness and achievable performance. Mainstream solutions are also called Big Data Stream Processing (BDSP) frameworks [20]. They target distributed systems (i.e., clusters of homogeneous machines) and typically rely on the Java Virtual Machine (JVM), whose platform independence provides portability at the cost of reduced achievable performance, due for example to processing overheads related to data (de-)serialization and garbage collection. Popular examples of BDSP frameworks are Apache Storm [6], Flink [2] and Spark Streaming [5]. Other solutions exists, with the same expressive power as mainstream ones in terms of offered abstractions, but differences in terms of runtime and targeted systems. One such example is the WindFlow [28] library. In fact, this C++17 library offers a low-latency programming environment for streaming applications by relying on the building blocks offered by the FastFlow [1] C++ library. Unlike state-of-the-art solutions, WindFlow targets single multicore machines, integrating support for GPU co-processors. Therefore, the

choice of the right DaSP framework for the particular use case must consider the target system and the strictness of the imposed constraints on throughput and latency performance parameters.

The framework we are proposing, aims at providing high-level programming abstractions to implement fast and scalable applications for real-time continuous network monitoring. Underlying, it will exploit DaSP framework general-purpose abstractions to express possibly very complex computations on many data streams, while guaranteeing high *performance* and *scalability*. Scalability will be further improved by implementing the processing according to a two-layer structure. Therefore, early stage portions of the processing will be directly performed in the data plane programmable nodes, while only a more refined analysis of these pre-digested results will be carried out in the control plane. The framework expressiveness will be enhanced by the addition of new *ad hoc* abstractions oriented to network programming in the used DaSP frameworks. Figure 1 offers a high-level view of the proposed framework architecture. The general idea can be summarized in the following:

- In the data plane, queries that are continuously executed on packet streams at line rate can be implemented using DasP frameworks with similar characteristics to WindFlow. The reason is the higher achievable performance, needed for real-time packet level analysis. Suitable abstractions for implementing source entities supporting various packet capturing solutions will be designed and provided.
- In the control plane, queries that are continuously executed on streams of pre-digested statistics can be implemented with any DaSP framework. The reason is the lower data rate at which partial results will be received from the first processing stage. The achievable performance of BDSP frameworks should be enough at this stage of the computation. Suitable abstractions for implementing source operators collecting statistics from the data plane level will be provided.

A general introduction and the necessary background on the DaSP computational model have been provided in this section. The rest of the paper is structured as follows. Section II gives an overview on other related works in literature. Section III describes the main challenges we identified, and proposes a methodology to tackle them. Moreover, the most promising research directions are pointed out. Finally, Section IV presents the current status of the work, and Section V provides a summary of the proposed strategies and their expected implications.

II. Related Work

The idea of a unifying framework providing at the same time both expressiveness and performance is shared with the authors of Sonata [14, 15]. This query-driven network telemetry system offers a declarative interface, and executes queries on programmable protocol-independent switches and stream processors. With respect to Sonata, we are proposing a system that entirely relies on DaSP and exploits it at both data and control planes (rather than focusing on the data plane only). Other systems like NetQRE [30], that are based on DaSP alone and run on general-purpose CPUs, are expressive but achieve lower scalability and performance. As part of this group, Apache Metron [3] (with its Metron PCAP Backend [4]) and OpenSOC [23] security analytics frameworks support the execution of many types of queries, but they achieve limited performance since they exclusively rely on BDSP frameworks. On the contrary, monitoring frameworks using programmable switches alone can handle high traffic rates, but they typically support limited sets of queries (depending on the switch resources) [29], although more recent solutions [18, 19] achieve higher generality. Solutions [18, 19, 29] all exploit a two-level processing structure similar to the one proposed here, as does also [27]. In [22], it is claimed (as we do) the necessity for stream processing tools explicitly devoted to network analytics. The architecture proposed in this work has similarities with the one presented here. However, the main focus is on improving performance, rather than on highering the level of the exposed programming abstractions. Recently, Mellanox proposed a solution for streaming telemetry [21] capable of performing traffic analysis at line-rate inside switches. Data on events of interest are collected and then streamed out of the switch for further processing (e.g., implemented with proper stream pipelines as proposed in this work). It is also worth mentioning some new efforts to enhance the capabilities of BDSP frameworks [9].

III. Methodology and Challenges

This section presents a list of research challenges we need to face to design the framework introduced in Section I. For each identified challenge, a solution will be proposed.

Research Challenge 1. Design abstractions for network related aspects

Figure 1 shows that the streaming logic executing in the data plane must be able to receive and process packet streams coming from Network Interface Cards (NICs) at wire-speed. Therefore, we plan to use some of the available accelerated capture engines and to exploit parallelism not only at the processing level but also at the packet capturing level (assuming to work on multicore machines, possibly equipped with co-processor capabilities like GPUs, FPGAs and SmartNICs). To keep up with the high rate at which packets are received, at this stage we plan to implement the stream pipeline with DaSP frameworks like WindFlow, that are capable of exploiting in the best way the available parallelism on the machine.

Problem 1: Several solutions for software high-performance packet processing exist [7, 10, 13, 24]. They generally exploit multicore parallelism, kernel bypass techniques and offloading features in order to keep up with high packet rates. To obtain good performance, packet processing applications are often given direct access and control over the network hardware, at the cost of a not exactly trivial implementation. Having to deal

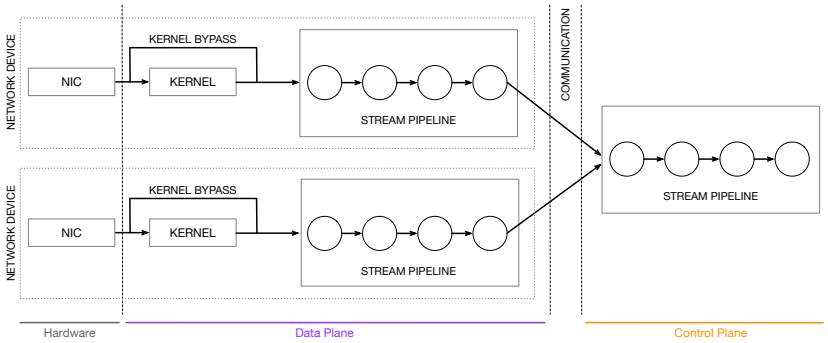

Fig. 1: High-level view of the architecture of the proposed framework.

with low-level aspects of the implementation makes it hard to write even a simple program.

Proposed solution: The general abstractions offered by DaSP frameworks do not help dealing with network programming details. Therefore, dedicated source operators which internally hide some of the low-level networking aspects will be designed and implemented. These new operators will enrich the abstraction set of DaSP frameworks selected to implement the stream pipeline at the data plane level (e.g. WindFlow). This will also encourage network programmers to adopt the proposed solution.

Problem 2: Packet capturing tools and parallel DaSP applications executing on the same multicore machine may compete for the same CPU resources. This may cause performance degradation if a proper configuration of the frameworks at both levels is not devised.

Proposed solution: We deem that a deep investigation is necessary to design a good strategy for optimizing the usage of the available cores on the machine. For example, a subset of the cores can be devoted to in-kernel packet processing, and the remaining to the DaSP application. Moreover, we plan to investigate the possibility to offload part of the workload in order to save CPU cycles. Again, solutions like eBPF/XDP [16] can do this at the capturing level (e.g., offload to a SmartNIC). Likewise, WindFlow offers support for GPUs [28]. Therefore, optimal trade-offs must be found to maximize performance combining parallelism and offloading features at both the capturing and the processing levels in the data plane.

Research Challenge 2. Selecting the most suitable DaSP frameworks

Problem: We must decide which are the right metrics to select the most suitable DaSP frameworks to implement the streaming pipelines shown in Figure 1.

Proposed solution: We started by observing that the stream pipelines at the data plane and the ones at the control plane have to handle very different input rates. The former must keep up with very high packet rates, the latter will work on significantly lower data rates (of the received statistics). Therefore, we propose a solution based on the performance results in [11], and on the discussion in [12]. We will consider JVM-based DaSP frameworks [2, 5, 6] to implement stream processing pipelines in the control plane. Since they target distributed architectures, they also offer the opportunity to distribute the stream pipeline on many instances of a distributed controller. Instead, solutions like Windflow are suitable to implement both kind of pipelines. In the particular case of WindFlow, the only constraint is that the computation must be performed on a single shared-memory system [28] (i.e., each control pipeline must execute on a single controller instance).

Research Challenge 3. Design the communication between data and control planes

Problem: An efficient communication schema between the data and control planes must be designed, together with strategies to minimize the communication delay and the amount of data exchanged.

Proposed solution: To minimize the quantity of data exchanged, data plane streaming pipelines will always produce some kind of aggregated statistics on the processed packets. Thanks to this, only a small amount of short tuples will be sent to the control plane. As for the communication latency, it is influenced by the distance. We observe that stream pipelines at the control plane can join several streams of statistics coming from different network devices. Therefore, our goal is to minimize the distance between all the communicating entities. To this end, we will devise new strategies to decide the optimal deployment of the streaming pipelines onto the physical nodes (network devices and controller instances) [12].

IV. Current Status of the Work

As an initial proof of concept, we started implementing an example of a data plane stream pipeline component illustrated in Figure 1. The pipeline will perform flow identification and analysis on a per-packet basis, and it has been implemented using the WindFlow DaSP framework.

A complete monitoring application could be made up by

- this data plane processing pipeline, which produces statistics such as sent and received bytes and segments, retransmissions, Round Trip Time (RTT), elapsed time, throughput and so on (in a way similar to what Tstat [26] does);
- a control plane stream pipeline (to be implemented), which collects these statistics, further aggregates them

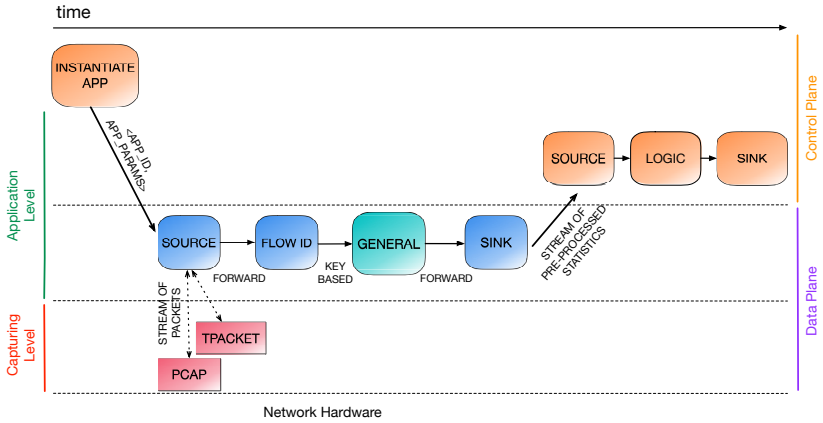

Fig. 2: High-level view of the operators implementing the proposed proof-of-concept application. The control plane creates an instance of the application when required. After the graph at the data plane level is created, it starts its execution on streams of captured packets. Results at the sink will be sent to the control plane, which can further analyze them, also generating visual representations or alerts (this can be implemented in the orange operator labeled as *Logic*). As for the data plane pipeline structure, FORWARD tuple distribution preserves load balancing, while the KEY-BASED one logically partitions the stream based on one or more tuple fields (e.g. *flow ID*) used as *key*. The green operator labeled as *General* will be extended to evaluate a variety of statistics. Notice that each operator can be fully replicated and it can be potentially characterized by a complex parallel internal structure.

(maybe joining together data coming from different network devices, to have a more global view), and for example generates alerts based on some threshold (for performance and security purposes).

Figure 2 shows a high-level view of the operators implementing the described proof of concept application. The current implementation covers the data plane pipeline, with support for two capturing level solutions.

Data plane stream pipeline: SOURCE. Two kinds of WindFlow source operators (represented by the blue SOURCE entity in Figure 2) have been implemented.

- The first source type supports the capturing of packets from the NIC by exploiting the `pcap` [25] library.
- The second type of source exploits the TPACKET (version 3) socket [17] to capture packets from the NIC.

In both cases, the blue SOURCE entity receives packets from the underlying level as a single input stream. Therefore, these two kinds of source entities are never parallelized (i.e., data plane pipeline using pcap-based or TPACKET-based source types always uses a SOURCE with parallelism degree 1). The SOURCE will produce an output stream of *tuples* (in the format supported by WindFlow), one for each collected packet.

We are working on adding support for other fast packet capturing engines able to distribute the processing directly at the capturing level [8, 16]. This means gaining the ability to access in parallel to different streams of packets. Therefore, the idea is to have a parallel SOURCE, and each replica will receive traffic coming from a different packet stream. In this way, the input rate the pipeline can handle will be increased. This is the first step of the investigation proposed to solve Problem 2 of Research Challenge 1 in Section III.

Data plane stream pipeline: FLOW IDENTIFIER. This operator (represented by the blue node labeled FLOW ID in Figure 2) receives all the packets as WindFlow tuples from the SOURCE. For each tuple, it identifies the flow to which the packet belongs. A *flow* is uniquely defined by a pair of source and destination IPv4 addresses, a pair of source and destination ports and the transport level protocol (TCP in this case). A unique flow identifier is generated, and each processed tuple is enriched with this ID field before it is forwarded to the successive operator. When FLOW ID is instantiated with parallelism degree higher than 1, tuples from the SOURCE are distributed in a load balanced fashion to all its replicas. Tuple distribution is handled directly by the WindFlow framework.

Data plane stream pipeline: GENERAL. This operator maintains a state and works on *keys*. The concept of key generally refers to a subset of the fields defining the stream tuples, which can be exploited to logically identify sub-streams [11]. Tuples generated by the FLOW ID operator are distributed by hashing on the flow ID field (used as key). Hence, if this *stateful operator* is instantiated with parallelism degree higher than 1, each one of its replicas will work exclusively on a subset of the identified flows, and will access only its own partition of the state (e.g., computed statistics for each flow). The structure of this operator is kept very general since it can be extended to keep track of a large set of statistics, and it can be in turn internally parallelized to keep up with the amount of processing it needs to perform on a single tuple. At the moment, simple packet and byte counters for each identified flow have been implemented.

Data plane stream pipeline: SINK. This operator is always the last one in the graph. It receives the computed per-flow statistics from the operator in the previous stage of the pipeline. The gathered information are aggregated and tuples containing these pre-processed statistics are sent to the control plane stream pipeline for further analysis.

The processing structure described in Figure 2 gives an idea of the execution flow in all the components of the architecture

presented in Figure 1. The controller starts new instances of data plane streaming pipelines on some network device. Results are received from one or more network devices and further analyzed in the control plane pipeline (the orange operators in the top-right part of Figure 2).

V. Conclusion

This work proposes a new framework which offers high-level abstractions to implement fast and scalable applications for real-time continuous network monitoring and data analysis. We propose to design this kind of applications following a two-layer structure. The processing will be implemented adopting the Data Stream Processing (DaSP) computational model, at both control and data planes. The main challenges and proposed solutions have been thoroughly described. The main implications of this work are to achieve through DaSP an optimal exploitation of the available hardware resources, preserving at the same time a high level of abstraction in the exposed API. On one hand, improving performance and optimizing resource usage will allow to accommodate on the same infrastructure a higher number of processing applications (observing their QoS constraints). On the other hand, providing new suitable abstractions will encourage network programmers to adopt the proposed solution.

Acknowledgment

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] M. Aldinucci et al. "Fastflow: High-Level and Efficient Streaming on Multicore". In: *Programming multi-core and many-core computing systems*. 2017.

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

[3] *Apache Metron*. http://metron.apache.org/. (Accessed on July 26, 2020).

[4] *Apache Metron PCAP Backend*. https://metron.apache.org/current-book/metron-platform/metron-pcap-backend/index.html. (Accessed on July 26, 2020).

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

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

[7] N. Bonelli, S. Giordano, and G. Procissi. "Network Traffic Processing With PFQ". In: *IEEE Journal on Selected Areas in Communications* 34.6 (2016).

[8] N. Bonelli et al. "Packet Fan–Out Extension for the pcap Library". In: *IEEE Transactions on Network and Service Management* (2018).

[9] Junguk Cho et al. "Typhoon: An SDN Enhanced Real-Time Big Data Streaming Framework". In: *ACM CoNEXT*. 2017.

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

[11] A. Fais. "Benchmarking Data Stream Processing Frameworks on Multicores". Università di Pisa, Oct. 2019.

[12] A. Fais et al. "Data Stream Processing in Software Defined Networks: Perspectives and Challenges". In: *Proceedings of the IEEE CAMAD 2020*. 2020.

[13] F. Fusco and L. Deri. "High speed network traffic analysis with commodity multi–core systems". In: *Proc. of IMC '10*. Melbourne, Australia: ACM, 2010.

[14] Arpit Gupta et al. "Network Monitoring as a Streaming Analytics Problem". In: *15th ACM Workshop on Hot Topics in Networks (HotNets '16)*. 2016.

[15] Arpit Gupta et al. "Sonata: Query-Driven Streaming Network Telemetry". In: *Proceedings of the 2018 ACM SIGCOMM Conference*. 2018.

[16] T. Høiland-Jørgensen et al. "The eXpress Data Path: fast programmable packet processing in the Operating System Kernel". In: (2018). CoNEXT '18.

[17] Linux Kernel Contributors. *PACKET_MMAP*. https://www.kernel.org/doc/Documentation/networking/packet_mmap.txt. (Accessed on 2018-06-15).

[18] Zaoxing Liu et al. "Nitrosketch: Robust and General Sketch-Based Monitoring in Software Switches". In: *Proceedings of the 2019 ACM SIGCOMM Conference*.

[19] Zaoxing Liu et al. "One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon". In: *Proceedings of the 2016 ACM SIGCOMM Conference*.

[20] Ahmed Al-Mansoori et al. "A Survey on Big Data Stream Processing in SDN Supported Cloud Environment". In: *Proceedings of the Australasian Computer Science Week Multiconference*. ACSW '18. 2018.

[21] Mellanox. *What Just Happened (WJH) Telemetry*. https://www.mellanox.com/related-docs/solutions/SB_Mellanox_WJH.pdf. (Accessed on October 10, 2020).

[22] Oliver Michel et al. "Packet-Level Analytics in Software without Compromises". In: *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*. 2018.

[23] *OpenSOC*. https://github.com/OpenSOC/opensoc. (Accessed on July 26, 2020).

[24] Luigi Rizzo. "Netmap: a novel framework for fast packet I/O". In: *Proc. of USENIX ATC 2012*.

[25] *TCPDUMP & LIBPCAP: Programming with pcap*. https://www.tcpdump.org/pcap.html. (Accessed on July 16, 2020).

[26] *Tstat: TCP STatistic and Analysis Tool*. URL: http://tstat.polito.it/.

[27] J. Vestin et al. "Programmable Event Detection for In-Band Network Telemetry". In: *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*.

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

[29] Minlan Yu et al. "Software Defined Traffic Measurements with OpenSketch". In: *10th USENIX NSDI 2013*.

[30] Y. Yuan et al. "Quantitative Network Monitoring with NetQRE". In: *Proceedings of SIGCOMM '17*. 2017.