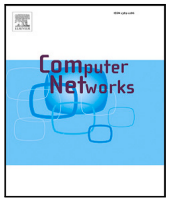
Computer Networks 216 (2022) 109227

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Latency control in service chaining using P4-based data plane programmability

Francesco Paolucci [a,*], Davide Scano [b], Piero Castoldi [b], Emiliano De Paoli [c]

[a] CNIT, Via G. Moruzzi, 56124 Pisa, Italy
[b] Scuola Superiore Sant'Anna, Via G. Moruzzi 1, 56124 Pisa, Italy
[c] MBDA Italia, Roma, Italy

ARTICLE INFO

MSC:
00-01
99-00

Keywords:
NFV
SDN
P4
Latency
Virtual network function
Switch
Data center

ABSTRACT

Service chaining is becoming one of the most considered service deployment frameworks in the context of Network Function Virtualization (NFV) in edge and data center environments, conveniently supported by automatic connectivity configurations offered by Software Defined Networking (SDN). Current research on the topic is focusing on how to guarantee Quality of Service (QoS) in terms of guaranteed end-to-end latency for time critical services. Indeed, latency issues may depend on intra-server virtualization inefficiencies, leading to Virtual Network Function (VNF) delivery delays, or by congestion events occurring at intermediate network elements connecting VNFs. Latency control requires stateful information such as flow delay measurements at a per-packet level, typically not available at traditional SDN switches or inside the VNF.

This paper proposes the adoption of SDN data plane programmability exploiting the P4 language and presents two P4 pipeline solutions, suitable for both intra-rack and inter-rack service chain deployments, to automatically check the path latency experienced by selected high priority flows, also resorting to the recent in-band telemetry applications. The programmable pipelines enforce proactive in-network functions, such as priority change or drop actions, in order to guarantee a bound SFC segment latency delivery, including both the network and the segment VNFs. The proposed solutions are implemented and evaluated in a network testbed employing programmable software switches showing their effectiveness in guaranteeing the configured end-to-end latency, and the limited effort in terms of additional processing at the P4 switch. The evaluation is carried out using the reference P4 software switch, i.e., BMv2. The aim is to validate the full P4 capabilities and the code feasibility in terms of scalability, load and resource impact and added intra-switch latency. The experimental results show the proposed approach scales with the number of forwarded flows and achieves per-segment latency control enforcement in both congested and non-congested scenarios with a very limited impact on the switch extra-latency, exploiting finer per-packet tuning of drop and priority change simply applicable through flow entry configuration. Applicability analysis on hardware switches guaranteeing line rate performance are provided.

1. Introduction

Network Service Function Chaining (SFC) represents an attractive solution to deploy network services and tenants with application-oriented differentiation in the context of the Network Function Virtualization (NFV) and Software Defined Networks (SDN) [1–5]. Such differentiation is realized by deploying and placing a number of Virtual Network Functions (VNFs) in the network and configuring traffic flows to selectively reach the desired VNF set in a chained (i.e., ordered) fashion to realize the desired Quality of Service (QoS). Some pre-defined VNFs enable service treatment (e.g., traffic profiler), traffic engineering (e.g., transport-layer balancer), application-specific processing (e.g., video transcoding), security (e.g., firewall function). All such functions are conveniently deployed on-demand as virtualized resources, such as virtual machines/containers/serverless function codes in the network cloud and edge clusters, with significant increase of service flexibility and cost reduction.

However, the chaining of virtualized services running as software releases on general purpose CPU, memory and storage resources offered by the cloud/edge are prone to performance limitations due to the absence of dedicated bare metal hardware platforms [3]. Virtual machines and containers are subject to a number of performance issues due to I/O

* Corresponding author.
E-mail address: francesco.paolucci@cnit.it (F. Paolucci).

https://doi.org/10.1016/j.comnet.2022.109227
Received 23 March 2022; Received in revised form 10 June 2022; Accepted 22 July 2022
Available online 30 July 2022

interrupts, operating system scheduling, CPU core scaling issues [6], virtual resource driver configuration and overloading. Such limitation may impact the VNF delivery delay, a key performance indicator for latency-critical service chains. The introduction of efficient I/O frameworks such as the Data Plane Development Kit (DPDK) may attenuate, but not delete, such issues, especially for time-critical services or when CPU and/or Network Interface Card (NIC) resources are limited [7,8].

On the network side, in the standard SDN framework, advanced packet treatment and processing are possible only sending out the packet to the SDN controller. Such procedures are prone to noticeable scalability issues given the single point of processing represented by the SDN controller. Moreover, the considered solution is not feasible for all the use cases in which latency constraints represent a key performance indicator. However, such indicators are not easy to extract from traditional SDN switches. Moreover, congestion events at the network level may break the latency bounds assured for the service chain deployment.

The SDN data plane programmability conceives a novel processing framework inside the network element directly, resorting to programmable pipelines [9]. The programmability include the definition at the switch level of custom parsers, flow tables, actions and stateful objects able to provide advanced functions that are not available with a standard SDN switch. The programmability resorts to a platform-agnostic high-level language called P4. The P4 (Programming Protocol-independent Packet Processors, p4.org) is a high-level programming language explicitly devoted to packet processors data plane [10]. The novelty and the potentials introduced by P4 has gained significant attention and interest by many chip manufacturers and the networking researchers [11–14]. Today, the P4.org ecosystem consortium includes more than 50 industrial corporations (e.g., Cisco, Dell, Google, HPE, Intel, Huawei, Juniper, Microsoft, ZTE) and many academic partners. The P4 programmability can be extended to programmable hardware switches (e.g., Intel Tofino and NVIDIA/Mellanox Spectrum-X), Linux servers employing DPDK or AF-XDP, and NIC. The main idea behind P4 is that traditional SDN-based control plane is able to populate an SDN network element with flow entries and actions related to standard pipelines, but it cannot explicitly program the data plane, while P4 may define novel pipelines with stateful capabilities, thus enabling innovative and decentralized in-network functions, ranging from telemetry up to cybersecurity applications [15].

This paper proposes novel P4 pipelines at the switch level to control the latency of service chained flows in a stand-alone and decentralized configuration. The proposed programmability allows to monitor the combined network/VNF delay variations accumulated by latency critical flow packets traversing one or a set of chained VNFs. The pipelines are programmed to enforce traffic treatment (e.g., change of priority, drop) on a per-packet basis, guaranteeing a maximum accumulated latency bound on the SFC network paths (i.e., segments) connecting the chained VNFs. In particular, two configurations are explored, implemented and evaluated in a P4 network testbed: (1) a stand-alone solution conceiving a single switch attached to a number of VNFs, reproducing intra-data center involving the top-of-rack switch, (2) a number of decentralized switches serving different VNFs each and employing extended in-band network telemetry (INT) solutions, reproducing inter-rack, inter-cluster and inter-data center chaining scenarios.

The evaluation is carried out using the BMv2 software switch [16]. Although not designed for high performance (i.e., maximum achieved line rate is around some Gbps), the BMv2 is the only platform that fully supports all the P4 processing capabilities with the highest degree of freedom. The evaluation highlights the impact of the proposed pipelines with respect to the reference pipeline only providing layer-2 forwarding, providing evidence that the solution is lightweight, scalable and feasible for hardware platforms, given the limited amount of employed stateful register resources.

The paper is organized as follows. In Section 2 we discuss the main issues related to the latency control in SFC and review the current state of the art of the literature, addressing the solutions proposed so far especially targeting data plane programmability. In Section 3 we propose our P4-based latency control solutions at the programmable switches, describing the rationale, the algorithms and the P4 design for the two baseline use cases (i.e., stand-alone switch, multi-switch INT segment). In Section 4 we report the experimental evaluations with a detailed and extensive set of results, highlighting the scalability of the solutions and the impact of each P4 tuning parameter. In addition, we discuss in detail the applicability of the solutions in the current programmable hardware platforms supporting P4. Finally, we draw the conclusions in Section 5.

2. P4 and service chaining latency: current issues

Data plane programmability in SDN enables the deployment of in-network functions at the hardware level of the switches. This is particularly attractive for SFC, in which a number of VNFs in a cascade configuration need to be connected using network routes guaranteeing QoS requirements.

The scenario of Fig. 1 illustrates the SFC flows deployment on different data center scenarios, assuming the SDN framework. The red service chain is made of three VNFs (i.e., V1a, V1b and V1c) instantiated in the same data center. In particular, V1b and V1c are instantiated in the same rack, thus the top-of-rack switch is configured to send and receive different flows belonging to the same service chain, (i.e., flow entries to forward V1a to V1b and V1b to V1c). Instead, the green service deploys three VNFs in three different racks, in addition V2b and V2c are deployed in a different cluster (or another data center) with respect to V2a. In this case, the VNFs are connected by a network path spanning a high number of SDN switches (e.g., 4 switches connecting V2a to V2b). Typically, QoS constraints are enforced at the VNF. The QoS requirements related to network performance, for example service latency constraints, require active cooperation between switches. In particular, with reference to SFC, two main issues prevent a full control of the packet latency along the chain:

1. the unpredictable packet delay experienced at each VNF step, due to software virtualization issues;
2. the network congestion events at each switch of the service chain.

Accumulated latencies originated by VNFs and switches may lead to bursts of delayed, high jitter packets. Such combined latency control is not straightforward, since it may be monitored locally, not in a distributed and proactive fashion. For example, INT may be exploited to monitor the network delay and pro-actively instructing in-network dynamic priority [17], however per-packet monitoring at the VNF level may not scale. Additional issues rely on the VNF packet manipulation, so that a single end-to-end flow may be terminated at some VNFs or splitted in segmented sub flows with different network matches, leading to significant end-to-end monitoring issues.

2.1. Related work on service chaining and data plane programmability

The introduction of data plane programmability has pushed the research ecosystem to propose novel, extended and high-performance service chain architectures, workflows and technologies resorting to NFV offload, stateful traffic steering and advanced functions, that have been proposed in different literature works to enable efficient SFC solutions.

First, the role of innovative I/O acceleration platforms and tools able to speed up virtual functions chaining inside a single host, such as the Virtual Ethernet Bridge (VEB), the Ethernet Port Aggregator (VEPA), the Single Root IO Virtualization (SR-IOV) has lead to a relevant research branch demonstrating that service function chaining achieves significant scalability (i.e., hundreds of virtual functions)

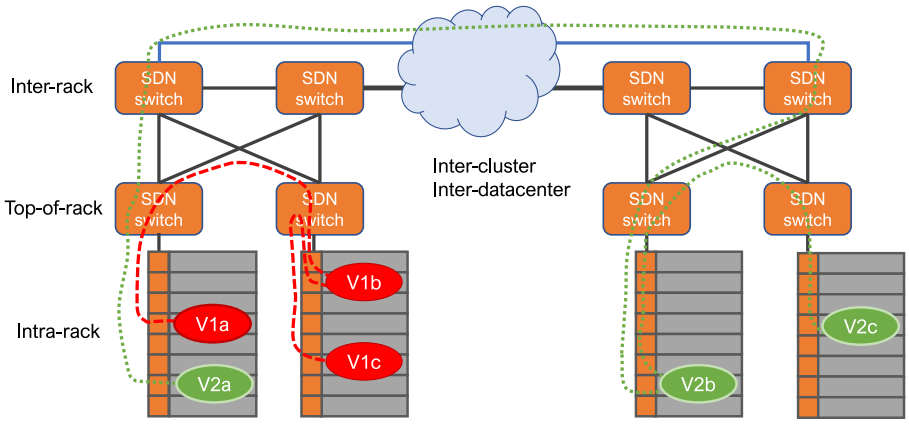


Fig. 1. Example of service chained virtual network functions in inter/intra data center scenario. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

and reduced intra-VNF latency burden [18]. For example, the works in [19,20] target service chaining performance improvement, in terms of throughput and latency, resorting to the SR-IOV technique, allowing buffer separation for each VNF at the NIC level (i.e., virtual NIC). The analysis considers undersubscribed and oversubscribed systems.

Data plane programmability at the SDN level introduces a further acceleration degree to SFC, allowing the offloading of selected network functions inside programmable devices. The work in [21] proposes the adoption of a chaining-box, developed as a Berkeley Packet Filter (BPF), a dedicated sequence of stages performing all the needed service function actions and achieving up to 40% latency decrease with respect to standard solutions. The work in [22] exploits strict source routing to deploy SFC-based traffic steering including SFC labeling and tableless switches forwarding based on label computation. The work in [23] proposes a framework to deploy service function parallelism, conceived to execute selected VNFs in parallel, thus achieving significant latency reduction. The work utilizes a DPDK-based container to adapt and replicate packet flows to be sent and received to/from parallel VNF instances.

A number of recent works are focused on the introduction of P4 for VNFs offloading, handling and for specific in-network and protocol solutions. Concerning P4-based VNF offloading, the work in [24] proposes a set of primitives processing SFC expressions directly deployed in P4 programs, drastically improving the performance with respect to software-based solutions. The work in [25] presents an overall offloading implementation of multiple service chain functions directly implemented inside a P4 switch to perform network acceleration. The works in [26] and in [27] illustrate two design methodologies for implementing multiple network functions and their programmable service chains inside a single P4 switch, resorting to FPGA and bare metal switch platforms, respectively, and showing constant latency performance. The work in [28] improves SFC performance by implementing P4 segment routing at the switches and SR-IOV virtualization at the hosts. A similar work in [29] considers the use of P4 to decompose the execution of pre-programmed VNFs in a network processor using a multi-level chaining scheme in order to optimize the resource allocation. A similar approach was considered by the work in [30], including the recirculation of functions over a programmable data plane resource and a service function chain manager performing placement and ordering of functions to minimize the chain completion time.

Concerning in-network functions for latency-critical services, the work in [17] implements extended INT and per-packet priority tuning at P4 switches in order to allow fast packet transit only for delayed traffic in the INT domain. The work in [31] proposes, in the framework of beyond 5G services, an extension of INT at the user equipment to measure and forecast traffic latency towards the target cloud resource, while automating latency-aware P4 switch steering to the closest edge node. Similarly, the work in [32] utilizes INT monitoring for dynamic serverless application migration between edge nodes.

Concerning time-critical services, a number of works target the problem of routing and placement of the VNFs belonging to a service chain, including delay constraints [33–35]. However, these affect the provisioning of the SFC rather than the control enforcement during the SFC execution, due to dynamic network conditions. To the best of our knowledge, there are not literature works addressing P4 inside programmable switches connecting VNFs for SFC end-to-end or segment-based latency control at per-packet level.

2.2. Service chaining scenario and objective

The application of the P4 pipelines is mapped and evaluated in a general service chaining scenario.

The flows subject to strict decentralized latency control are hereafter referred to as Latency Constrained flows (LC). Such flows are directed to a number of processing VNFs defined by the service chain graph. Some of these VNFs may terminate/modify the network protocol stack. Thus, LC flows are mapped onto a sequence of sub flows F1, F2, F3 with different L2, L3 and L4 fields, crossing network switches, even repeatedly. Standard SDN implementations are not able to intercept LC traffic properly, compute the latency associated to the flow sequence and apply latency-aware policies. Moreover, solutions based on INT are not feasible, since VNFs are often served by legacy NIC that do not support INT processing. A possible solution is to tag LC traffic flows with static high priority flag. However, this strategy has the following drawbacks:

- all LC traffic is treated high priority in the same way, regardless of the packet latency condition;
- additional concurrent high priority traffic is treated in the same way, possibly competing in the switch queues and with undefined behavior from the point of view of the SFC end-to-end latency;
- latency bounded conditions are not met and high priority guarantees the minimum latency only at the switch level, and not at the service chain level including the transit to/from VNF.

The objectives of this work are the following:

1. Extend the service chain SDN framework to data plane programmability using P4 switches for segment latency control.
2. Implement, evaluate and demonstrate a proof-of concept of programmable P4 switch handling fine-grained per-packet treatment without the intervention of the controller.
3. Implement in-network latency-bounded functions at the switches able to compute multi-hop accumulated packet latency due to both network and VNF and dynamic packet attribute enforcement policies, such as priority change and early drop profiles in the case of specific latency thresholds.

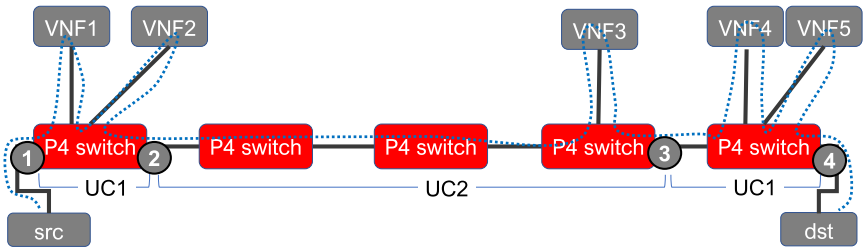


Fig. 2. Combination of UC1 and UC2 for SFC latency segment control.

3. P4 latency control solutions

The description of the architectural and implemented solutions based on the P4-based programmable data plane for service chains is articulated considering two main baseline Use Cases for a given SFC segment:

- Use Case 1 (UC1), considering a stand-alone scenario of a single P4 switch connecting the considered VNF chains (e.g., single Top of the Rack scenario)
- Use Case 2 (UC2), including a cooperation of different upstream P4 switches running INT (e.g., inter-rack scenario).

While UC1 considers only a single switch segment, UC2 covers the segment identified by a number of subsequent INT-enabled switches. This scenario includes intra-data center connectivity. In addition, it potentially includes inter-data center connectivity, provided that the metro network connecting the data centers is INT-enabled and data center controllers may issue dedicated flow entry configuration requests to the metro network controller. The use cases are hereafter described with their specific implemented solutions. Complex VNF placements may be seen as the combination of the two aforementioned use cases. For example, the latency control of VNFs connected to different switches can be decomposed in the latency control of two UC1-based or UC2-based SFC segments (e.g., UC1+UC1 between VNFs connected using two top-of-the rack switches, UC1+UC2 between VNFs connected in intra-rack and subsequent inter-cluster configuration, UC2+UC2 for double inter-cluster configuration). The example of Fig. 2 shows the application of SFC segment latency control to a SFC composed of five VNFs. In this case, three latency control segments are envisioned: segment 1–2 (including VNF1 and VNF2) and segment 3–4 (including VNF4 and VNF5) are controlled using the stand-alone scenario (UC1), while segment 2–3 is under the INT-based scenario (UC2), there being more then one switch connected without intermediate VNFs.

3.1. UC1: Stand-alone P4 switch

3.1.1. Problem description and assumptions

In UC1, service chain is composed by different VNFs connected to the same switch. To assure the latency control of a specific portion of SFC (hereafter denoted as *SFC segment*), the switch must be aware of each flow mapping forming the service chain. Moreover, to apply latency-aware policies, a specific mechanism is required at the switch to compute the latency that the flows are subject to in the forward and backward paths to/from the different VNFs. Without loosing generality, referring to Fig. 3, the key idea is to consider the ingress timestamp of the SFC flow packet received at interface 1, and the ingress timestamp of the same SFC flow packet received at interface 2, after all the SFC attached VNFs have been crossed (i.e., the interface connected to the last VNF). Intermediate VNF processes are considered within these two reference timestamps. Fig. 3 shows the example of three VNFs, however the concept is generalized to any number of attached VNFs. The difference between the two edge timestamps represents the total

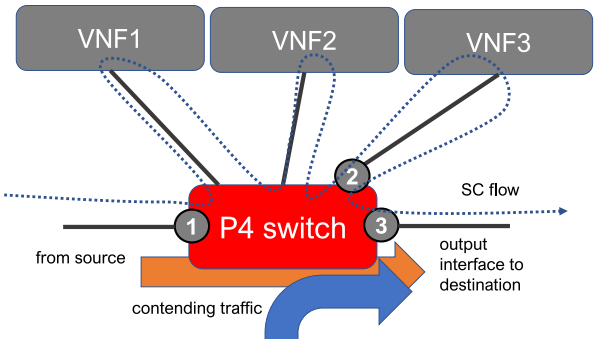


Fig. 3. Use Case 1: stand-alone switch scenario.

latency of the packet in the multi-VNF re-direction and transit. In order to compute this latency properly, it is necessary to match the flows correctly, temporarily store the packet timestamps in the switch and correlate the timestamps to the same packet without any sequence ambiguity. To solve the sequence order uncertainty, it may be not worth to rely on standard matches based on addressing, since the various flows may change MAC, IP and L4 ports after the VNF processing. Possible sequencing may be exploited using the identification field of the IP protocol, typically providing an incremental id value. In this paper, without loosing of generality, we assume that packets are processed sequentially with no packet loss at the VNF layer and without any reordering process.

Another assumption is that the interfaces towards the VNF are not subject to congestion issues. Since VNFs are related to latency constrained traffic, logical interfaces are assumed configured in order to assure the throughput of LC traffic at the minimum latency.

Thus, variable latency contribution sources are the VNFs and the final forwarding to the next segment using the output interface towards other switches or the SFC destination e.g., the latency experienced between interface 2 and interface 3 in the example of Fig. 3. At this regard, the co-existence between LC traffic and other high priority traffic may induce congestion at the node, thus increasing LC latency in the SFC segment. For this reason, LC traffic needs to be treated in a per-packet fashion and must be subject to strict latency-constraint conditions, related to both upstream delay (i.e., VNF delays) and switch-internal delay (e.g., congestion at the queue). Therefore, the latency-controlled policy is applied to the SFC segment between interface 1 and interface 3, thus accounting both the VNFs latency and the intra-switch latency.

3.1.2. Latency control and policies

Given the aforementioned assumptions, a proactive per-packet latency control policy at the switch is proposed.

Referring to Fig. 3, LC traffic is timestamp-stored at interface 1 (i.e., the interface connected to the upstream network segment, used by LC traffic to reach the switch for the first time) and at interface 2, (i.e., at the ingress interface connected to the last processed VNF, in

this case VNF3). The related ingress timestamps t_1 and t_2 are referred to the same switch and thus are not prone to synchronization issues. Thus, the difference $t_2 - t_1$ is the latency experienced by the packet in the multi-VNF processing steps. A second contribution must take into account the switch queue time from interface 2 to interface 3.

The following workflow is proposed to store LC traffic timestamps and compute per-packet segment latency $t_2 - t_1$. A mobile window register array stores the ingress timestamps of the LC packets in the order they arrive. A specific offset register at each array identifies the last processed packet. This way, if the dimension of the array is large enough to avoid cyclic rewrite operations, the timestamps at a given offset are referred to the same LC packet and latency can be computed as the difference between the two timestamps identified by the offset of the last array. Note that, if such assumptions are not valid, further header processing (e.g., the use of IP id field) may lead to the same result.

The total segment latency experienced by the packet is $(t_2 - t_1) + T_q$, where T_q is the time spent to forward the packet from interface 2 to interface 3, where resource contention may take place with the overall outgoing traffic and thus associated to the time spent in the switch queue.

In order to implement latency-controlled policies, the rationale assumes that latency-sensitive services (in particular, hard time-critical services, such as in avionics, automotive, spacecraft, IoT, military applications) need to be executed within a maximum time threshold, often referred to as worst-case execution time (WCET) [36,37]. In these contexts, a data must go through a long processing chain and transformations (functional chains) to reach the end of this chain and be definitively consumed by the effector. In many cases, if the processing time is too long and exceeds the threshold, the information has to be considered out-of-date and not usable, as it is known a-priori that the performance of the system (if the old data is used) is below the acceptable QoS limit. Moreover, the circulation of the old data on the network would also cause a further performance issue due to the useless occupation of network resources with the risk of congestion. For these reasons, the possible actions that a P4 switch can enforce on a time-critical SFC packet are either to speed up its forwarding (i.e., changing the packet priority) if the accumulated latency is below but close to the hard threshold, or to drop the packet if it already exceeds it.

The proposed latency-constrained policies introduce two parameters, called *Priority Latency* (PL) and *Drop Latency* (DL). The Priority Latency (PL) is the maximum allowed segment latency for which no priority change at the switch level is enforced. The Drop Latency (DL) is the maximum allowed segment latency for which the packet is not dropped. The relationship between the two parameters is given by $PL \leq DL$. The two DL and PL parameters are conceived to be configured by the SDN controller in the switches during the SFC flow rules instantiation and may be tuned if latency constraints are time dependent. The proposed policies are the following:

1. If $(t_2 - t_1) + \overline{T_q} > PL$ then the packet is changed priority to speed up its transmission and avoid congestion at the switch;
2. If $(t_2 - t_1) + \overline{T_q} > DL$ then the packet is dropped since the information carried out is considered out of date.

Note that $\overline{T_q}$ is the estimation (or a forecast) of T_q. In fact, priority may be enforced at the switch before the packet enters the scheduler and the queue. Thus, a P4 implementation has to implement $\overline{T_q}$ computation.

3.1.3. P4 implementation

The considered P4 implementation is based on the V1 architectural model assumed by the reference P4 software switch, the Behavioral Model version 2 (BMv2) [16]. The model includes a programmable ingress pipeline, a non-programmable queuing, replication and scheduling module and a programmable egress pipeline. The ingress stage is utilized to perform forwarding, while the egress pipeline is utilized to perform operation after the forwarding output port assignment. The proposed P4 implementation includes both the pipelines.

The internal structure of the ingress pipeline is depicted in Fig. 4. The pipeline is executed after the P4 parsing section, not shown in the figure. The pipeline includes the cascaded execution of three flow tables:

1. *Table0*, responsible of packet forwarding;
2. *Table Store*, responsible of recording LC packet timestamp at its first occurrence ($t1$);
3. *Table Enforce*, responsible of recording LC packet timestamp at its last occurrence ($t2$) and applying per-packet latency-constrained policies (priority, drop).

After parsing, the packets enter the Table0 table, responsible of forwarding, i.e., selecting the output port based on L2 MAC addresses (switch behavior). Table Store is responsible of intercepting SFC traffic received at interface 1 and activate the related action of storing the related timestamp t_1 in a specific array *t1* of programmable registers, using a register *offset1*, pointing to the last processed packet index. At the end of the action offset1 is incremented. The P4 code allows to define the size of the register array, so that a reasonable size may be allocated as a function of the average LC throughput, in order to avoid overloading.

Table Enforce matches LC packets received at interface 2. The related action stores the t_2 timestamp in a second *t2* register array using the *offset2* index. In addition, it reads the t_1 register at the offset2 index to retrieve the related t_1 timestamp. Then, t_2 is subtracted from t_1 and compared with PL and DL, stored in appropriate registers. If latency exceeds PL, the packet metadata priority is increased, and if exceeds DL the packet is dropped (using a P4 pre-defined drop action). Priority is handled in BMv2 with seven levels selectively applied to each egress interface packet queue, once forwarding is performed [16]. In this implementation, priority is set to the maximum value when exceeding PL, in order to speed up packet transmission as fast as possible. To account for the $\overline{T_q}$ estimation, a feedback mechanism between the ingress and the egress pipelines is implemented, shown in Fig. 4. The egress pipeline includes a specific flow table called Congestion Check that matches standard traffic (i.e., best effort transit traffic, not subject to latency control) and evaluates the time spent by each matched packet in the queue. Such data are retrieved using the pre-defined queueing metadata, available once the packet has crossed the queue itself (i.e., in the egress pipeline). The standard queueing metadata available in P4 switches are reported in the figure and include the queue entrance timestamp, the queue size at both entrance and exit time instants, and the time spent in the queue (i.e., *deq_timedelta*). The latter is written in a specific register (*T_queue*), available at the ingress pipeline level. The time spent in the queue by a packet is computed automatically by the P4 switch in a very precise fashion. In fact, pre-defined packet metadata are used to timestamp the packet before entering and after exiting the queue, and the delay is the result of the subtraction of the two metadata timestamps (e.g., in the BMv2 switch this delay has 1 μs resolution). In this way, the switch estimates the queue time that an incoming packet will likely experience in the case latency policies are not applied to the packet. This queue time is utilized as $\overline{T_q}$ in Table Enforce inside the ingress pipeline. This means that the estimation considers the queueing time of the last best effort packet exited from the queue at the time the considered packet is just entering the queue, thus it offers the most updated snapshot of the queue state. The computed $\overline{T_q}$ may result to delayed latency estimation in case of micro bursts (i.e., a congestion event in a very limited time between the queue processing of the last packet and the entrance of the considered packet). In this case the P4 code may be refined and the estimation may consider additional queueing metadata, the queue depth at the packet enqueue (Q_e) and dequeue time (Q_d), and the estimation updated as $\overline{T_q} = deq\_timedelta\ Q_d/Q_e$. The proposed implementation is applicable to a different number of LC flows. The pipeline structure and the flow

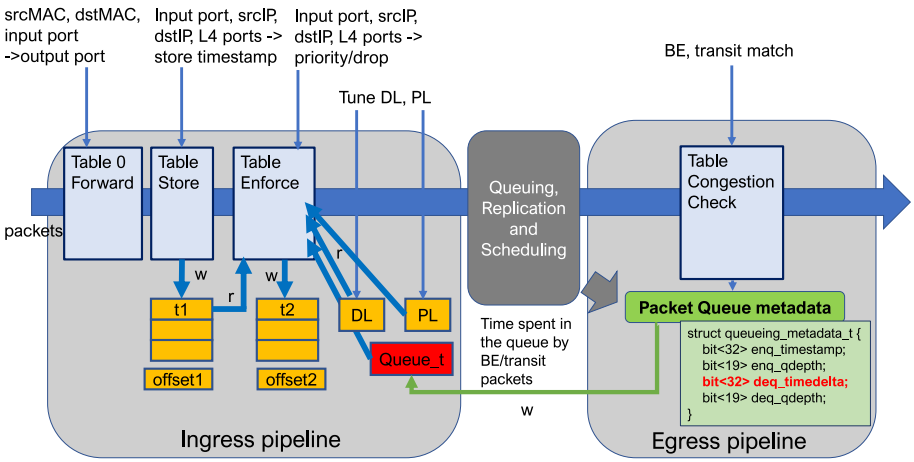


Fig. 4. UC1: P4 pipelines implementation.

tables are the same. For each LC flow, a couple of array registers needs to be instantiated to store timestamps. The match between the flows and the registers is performed using direct registers linked to Table Store, allowing flow match along the detection of the registers used by that flow. The P4 program may define the maximum number of supported LC flows.

3.2. UC2: Multi-switch latency control using INT

The considered UC2 extends the applicability of UC1 by imposing extended network segment latency policies involving a number of different distributed switches. While UC1 confines the latency check inside a stand-alone switch, UC2 introduces a switch network domain and latency check is carried out resorting to INT. A INT-enabled domain conceives a number of switches able to insert extra headers to selected traffic carrying out switch metadata. As an example, the hop latency of each switch is inserted in packet extra headers and carried out along the network domain. The nodes are distinguished as source, transit and sink nodes. A source node starts the INT extra-header, transit nodes add their own extra headers, while the sink node extracts all extra headers data and removes the extra headers to guarantee end-to- end traffic transparency. The application of INT to latency-constrained traffic is considered by extending the problem description and the scenario described in UC1, as shown in Fig. 5. With respect to UC1, in UC2 the connection between the source and the switch is replaced with a network domain of switches. In the case of programmable switches (or, at least the switches at the domain edge), latency control may be extended to all the network path and not confined in the VNF-to-VNF processing as described in UC1. The key idea is that each switch may insert the information related to its hop latency in each packet. It is worthwhile to note that INT is implemented in a per-packet fashion. That is, each packet is able to carry out the information related to the actual latency experienced by the packet itself inside each different traversed switch, and then along the network domain.

The scenario may be simplified as reported in Fig. 5 to explain the extension of the latency control to a network domain. An upstream source node is in charge of adding INT header to selected UDP traffic. The source node inserts the experienced hop latency l_{h1} experienced in the source node. The INT packet is processed and updated by transit switches (optionally computing l_{h2} and l_{h3}, if the domain is fully INT-enabled) and is received by the P4 switch attached to the VNF hosts acting as a INT-enabled sink switch. This way, the sink switch removes the extra header and collects the hop latency of the source switch to be considered in the latency-based policies proposed in UC1. Excluding transmission and propagation time contributions (stable delay contributions that may be estimated easily and considered in the latency computation budget), the sink switch may monitor the packet latency up to the ingress interface of the source node, thus emulating a functional migration of interface 1 up to the source edge INT switch interface 0 (see Fig. 5). The INT-enabled latency policies applied at the sink switch are similar to UC1. However, in this case, the $t1$ timestamp is decremented to account for the total accumulated latency along the upstream network extracted by INT headers and denoted as l_h.

The DL and PL definitions are the same, however applied to the entire INT network segment (i.e., from interface 0 to interface 3). The proposed policies in UC2 are the following:

1. If $(t_2 - t_1 + l_h) + \overline{T_q} > PL$ then the packet is changed priority to speed up its transmission and avoid congestion at the switch;
2. If $(t_2 - t_1 + l_h) + \overline{T_q} > DL$ then the packet is dropped since the information carried out is considered out of date.

3.2.1. P4 implementation

The UC2 implementation includes the P4 codes of the source switch and the sink switch. Source INT switch P4 code is standard as provided by the P4.org specifications. The sink switch code is an extension of the UC1 P4 code. The main differences are related to the ingress pipeline structure and the implemented actions. The ingress pipeline structure is depicted in Fig. 6. First, the parser section has been enriched with the INT extra header structure definitions. These extended parsers are required to process incoming INT-tagged packets properly and to extract hop latency values related to upstream switches. Details of the INT parsers are reported in the P4.org specifications document [38]. Additional flow tables (INT source, transit, sink) are introduced to include the full INT processing at each switch stage and for each switch role. In particular, dedicated flow entries are required to specify the role of the involved switch in the INT chain (i.e., source, transit, sink) and to specify the incoming interfaces, the traffic matches and the metadata to be retrieved and encapsulated in the extra headers. In this implementation the INT header is created and conveyed over the UDP header. The carried metadata information are the switch hop latency (used) and the egress timestamp (not used).

The specific functions implemented in the sink node are the deletion of the INT header and the insertion of specific INT fields inside the packet metadata. The most important metadata retrieved by INT is the source node hop latency. The forwarding table (Table 0) is left unchanged with respect to UC1. The Store Table matches the LC traffic and stores in the registers array the ingress timestamp subtracted with the source node hop latency $(t_1 - l_h)$. The rest of the implementation is the same of UC1, including the egress pipeline. In this way the same latency policies are implemented by considering the packet latency accumulated in the two switches and in the VNF transit.

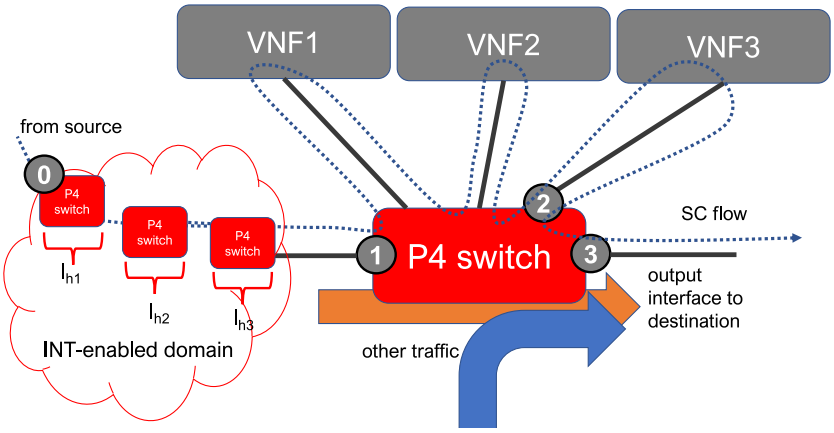


Fig. 5. Use Case 2: distributed INT switch scenario.

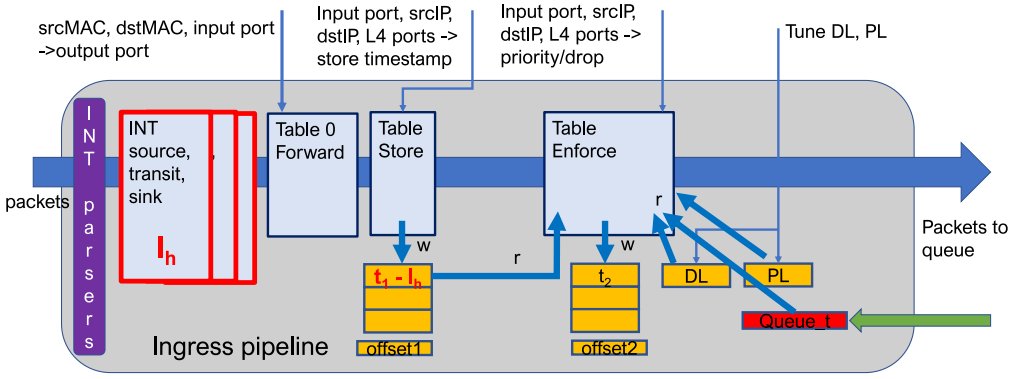


Fig. 6. UC2: P4 implementation of the sink switch: ingress pipeline.

4. Experimental evaluation

To evaluate the effectiveness of the proposed data plane programmability functionalities at the P4 switches, a comprehensive set of experimental evaluations have been carried out and hereafter reported.

The set of reported results aims at assessing the following key performance evaluation: (1) the low impact of the P4 codes in terms of data plane resource load and increased intra-switch latency with respect to standard P4 codes (e.g., layer-2 forwarding only) (2) the scalability of the P4 codes in terms of the number of considered flow entries related to transit traffic (3) the effectiveness of latency-control policies applied at the data plane for the UC1 and UC2 SFC segments in both static and dynamic conditions, evaluating the benefits of the policies in the case of VNF delay and switch congestion events.

The P4 codes of UC1 and UC2 have been written in P4 version 16 [10], deployed using the p4c compiler and enforcing the resulting json file in the running instance of the Behavioral Model version 2 software switch [16]. A unique P4 code, suitable for both source, transit and sink nodes in UC2 has been considered. Moreover, a number of auxiliary P4 codes have been written to deploy the experimental setup. The VNF considered in the evaluation has been designed to be a generalized VNF with the following capabilities: forwarding using the same input/output port, configurable variable VNF processing time, configurable packet treatment and manipulation at layer 2, layer 3 and layer 4. In particular, the VNF processing emulation has been implemented resorting to a further P4 code (referred to as P4 Reflector VNF), responsible of forwarding and modifying the MAC addresses, the IP addresses and the UDP ports of selected incoming traffic flows. Finally, a simple forwarding P4 code has been employed as baseline code to evaluate the impact of the logic implemented for the latency-controlled mechanism.

Table 1
Spirent traffic generator parameters.

Traffic parameters	Configured values
Interfaces	10GBE
Packet length (IP)	256 byte, 1500 byte
Transport protocol	UDP
Number of flows	up to 1000
IP address ranges (LC)	10.10.0.0/16
L4 port ranges (LC)	[43 000, 47 000]
Throughput	up to 1400 Mbit/s
Traffic duration	120 s

The experimental setups used for the two use cases are similar. The traffic has been generated and analyzed using the Spirent N4U traffic generator.The synthetic traffic parameters considered in the tests have been summarized in Table 1, while the actual values are detailed in the test description. The P4 switch under test have been deployed in a Linux Ubuntu PC server (Intel Xeon CPU E5-2620 6-core 2.10 GHz, 16 GB RAM). The P4 Reflector VNF runs in a different Linux Ubuntu PC server and the two servers are connected using a 10Gigabit Ethernet copper cable interface. The P4 code implementing the considered use cases (P4 file) are first compiled using the p4c compiler and specifying the BMv2 simple switch instance as target node.

4.1. UC1 setup and results

The setup shown in Fig. 7 is utilized to perform the UC1 evaluation. The generator is attached to two 10 Gb Ethernet optical interfaces and generates three different flows: the Latency Constrained flow (LC) is the flow subject to latency control, the Best Effort flow (BE) is the flow co-routed with LC but not subject to latency control. A transit traffic with

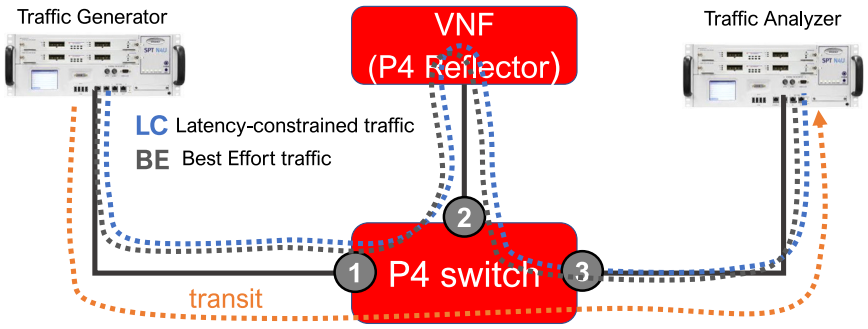


Fig. 7. Testbed used to evaluate Use Case 1.

```
#MAX_ALLOWED_ACCUMULATED_LATENCY_TO_PERMIT_FORWARDING (DL)
register_write drop_latency 0 15000

#FLOW RULE CONGESTION EMULATION
#set_queue_rate 6000 3

#QOS TRAFFIC FORWARDING
table_add table0 set_egress_port 1 00:04:aa:00:00:00 00:04:aa:00:00:01 => 2
table_add table0 set_egress_port 2 00:04:aa:00:00:01 00:04:aa:00:00:03 => 3
table_add table0 set_egress_port 1 00:04:00:00:00:00 00:04:00:00:00:01 => 2
table_add table0 set_egress_port 2 00:04:00:00:00:01 00:04:00:00:00:03 => 3

#STANDARD TRAFFIC FORWARDING
table_add table0 set_egress_port 1 00:04:00:00:00:10 00:04:00:00:00:03 => 3

#QOS LATENCY CONTROL TREATMENT
table_add table_store store_timestamp1 1 10.10.10.1 10.10.10.2 43000 45000 =>
table_add table_enforce enforce 2 10.10.10.2 10.10.10.3 45000 47000 =>

#EGRESS CONGESTION CHECK
table_add table_congestion_check do_update_congestion_reg 1 00:04:00:00:00:10 00:04:00:00:00:03 =>
```

Fig. 8. P4 switch flow entries for UC1 tests.

higher transmission rate have been added to emulate the medium and low priority traffic at the switch. Both LC and BE traffics are directed to the local VNF. Forwarding and latency control configurations are enforced using specific flow entries at the P4 switch using the BMv2 command line interface (CLI).

The flow entries utilized to enable the P4 behavior for the three considered flows are reported in Fig. 8. The first entries are related to the configuration of the PL and DL parameters inside specific registers. The congestion emulation entry is utilized to produce traffic congestion at the P4 switch due to the limited transmission rate of output interface 3. Then, all the flow rules related to forwarding (table Table0) are hereafter reported. Each traffic is identified by the input interface, the source MAC and the destination MAC. For each match the related set_egress_port action selects the output interface indicated after the arrow as action parameter. The LC traffic needs to be matched in the Store and Enforce tables. The two traffic matches are different since the flows are modified by the P4 reflector VNF instance (incoming traffic has 10.10.10.1 → 10.10.10.2 IP addresses, UDP ports set to 43 000 and 45 000, while outgoing traffic has 10.10.10.2 → 10.10.10.3 IP addresses, UDP ports set to 45 000 and 47 000). The two tables require to specify the incoming interface of the matched traffic (interface 1 for store, interface 2 for enforce, see interface numbering in Fig. 7), thus keeping the switch configuration extremely flexible. Finally, the egress congestion check table match identifies the transit traffic through its MAC addresses and the incoming interface (interface 1).

4.1.1. Switch throughput, end-to-end latency and scalability

The end-to-end latency performance of the switch related to transit traffic is reported in Fig. 9. The plot shows the packet latency measured by the Spirent analyzer able to sustain the throughput indicated by the input load in the X-axis with a packet loss rate below 0.1%. Therefore, for each plot, the point with the highest X-axis value represents the maximum sustainable throughput of the P4 switch. The plot shows that, using of 1500 byte long packets the switch is able to sustain up to 1.4 Gbps throughput with a stable latency profile between 80 and 120 µs. Moreover, the impact of the latency control (FL) with respect to standard forwarding function only (F) is limited, in the order of 5–10 µs of additional latency. The throughput decreases with a reduced packet length (i.e., 256 bytes), since the software switch has a processing rate limitation based on the packet arrival rate. In this case the throughput is reduced to around 300 Mbps and the additional load induced by the latency control code is limited with respect to baseline forwarding operations. This confirms that the latency control operation has a limited impact in terms of processing burden inside the switch.

The plot of Fig. 10 evaluates the end-to-end latency of BE and LC traffic traversing the switch, the VNF and the switch again. The plot shows that the additional processing burden induced by the full latency control processing results in an incremental latency contribution between 20 µs and 50 µs, depending on the input traffic throughput. This means that P4 latency control requires an additional latency contribution at the switch level of tens of microseconds. The price to pay for latency control is kept limited with respect to latency values to be controlled, typically in the order of hundreds of microseconds. In the case of congestion, the latency size is expected to increase at tens of milliseconds. This confirms that latency control has a limited impact on the switch processing and additional latency contribution.

Finally, scalability in terms of number of flow entries have been evaluated. The flow entries are related to the forwarding Table 0 entries, impacting on the number and type of flows that need to be forwarded. The results reported in Fig. 11 show that no significant impact is achieved when increasing number of flow entries are actively installed in the switch, from the point of view of switch latency. This is confirmed by many studies in the literature. The impact relies only in a slight restriction of the working range of the switch. In fact, the switch achieves a 1.3 Gbps throughput with one entry, while it sustains 1.2 Gbps when 1000 entries are installed. Such limitation is due to the internal software switch architecture. The switch instance is divided in four main threads. When the most processing threads, running on

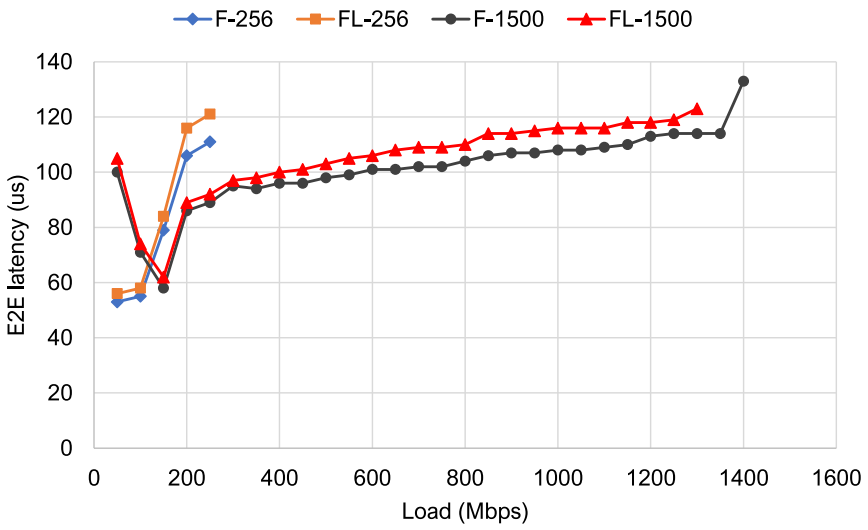


Fig. 9. End-to-end latency and sustainable throughput with (FL) and without (F) latency control functions (packet size 256 and 1500 bytes).

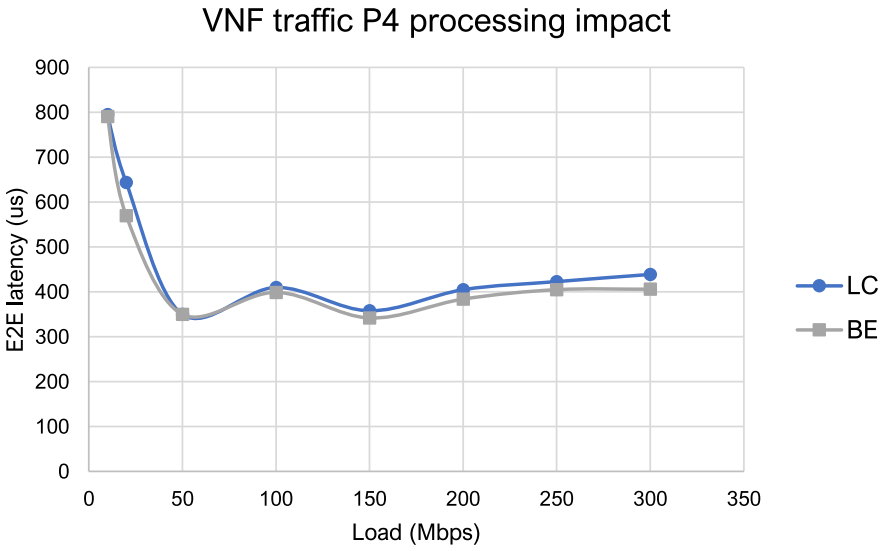


Fig. 10. Service chaining traffic end-to-end latency: impact of latency control workflows.

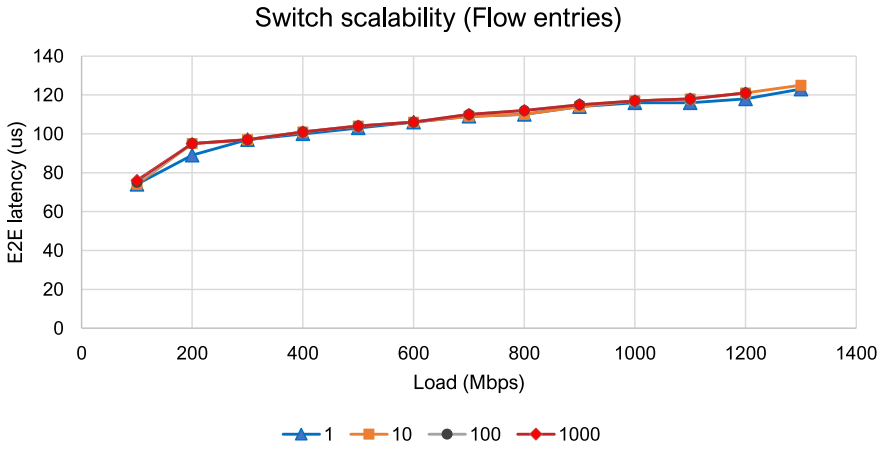


Fig. 11. Scalability in terms of installed forwarding flow rules.

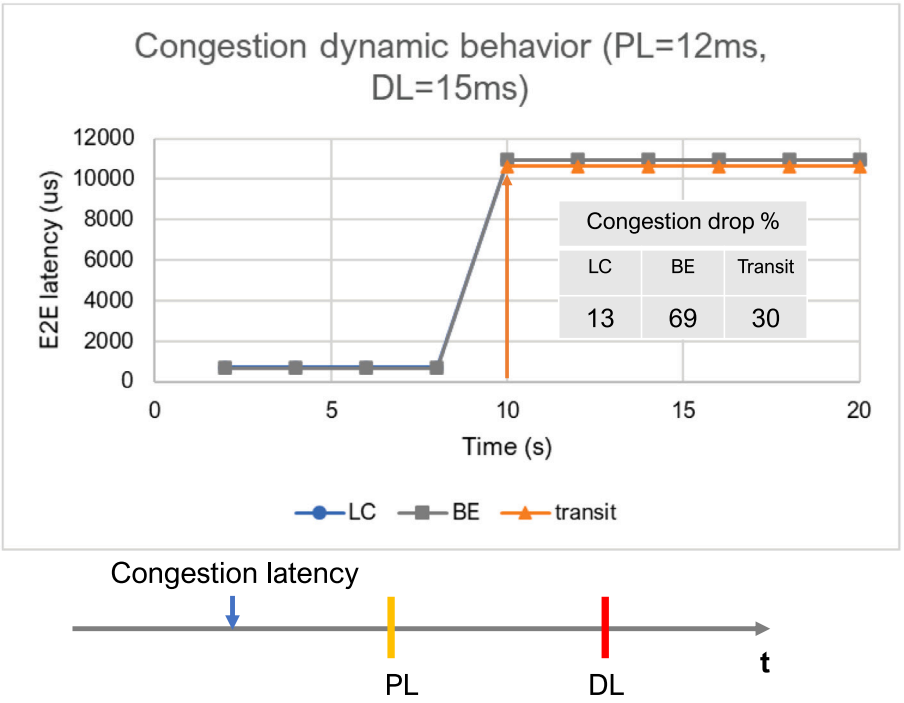


Fig. 12. Test 1.

a single CPU core, reaches 100% load, the switch becomes unstable and starts discarding packets in a random fashion. This condition is reached with a reduced throughput when an additional number of flow entries occur. This behavior is typical of software switches, while it is negligible in hardware programmed switches, since Ternary content addressable (TCAM) memories are used and matches are solved with a single CPU clock occurrence.

4.1.2. Dynamic behavior in case of congestion

In this set of tests, the switch behavior is evaluated in terms of different end-to-end latency and packet loss rate as a function of time when VNF traffic flows (LC and BE) are subject to congestion event due to the activation of high rate transit traffic (transit). The dynamic behavior of the switch depends on the values configured for DL and PL with respect to the actual latency values due to congestion at the switch. In these tests we have configured a standard scenario of congestion at interface 3 of the switch. LC and BE traffic flows are activated at time t = 0 with a transmission rate of 1000 packet/s each (corresponding to 12 Mbps each), while transit traffic is activated at time t = 10 s with a transmission rate of 7000 packet/s (corresponding to 84 Mbps). The congestion is created inside the switch by limiting the transmission rate of the P4 queue at 6000 packets/s for the only outgoing interface 3. This amount of congestion causes a packet latency of around 10 ms. Such congestion scenario is perfectly equivalent to a real congestion, because it induces an increased size of the average queue occupancy at the P4 switch. Three tests have been conducted, setting the DL and PL parameters in different working zones with respect to the average congestion latency.

In test 1, reported in Fig. 12, the congestion latency is lower than the configured PL (set to 12 ms) and DL (set to 15 ms). This means that both parameters allow for a latency control only when latency exceeds 12 ms. The results show that, before congestion, VNF traffic has a latency of around 600 μs (traffic generator → switch → VNF → switch → traffic analyzer). With congestion, all traffics are subject to the same treatment with a congestion drop of 13% (LC), 69% (BE) and 30% (transit). Most important, all latencies are around 11 ms, meaning that all traffic flows are treated with same priority option inside the switch.

In test 2, reported in Fig. 13, the congestion latency is lower than the configured DL (set to 15 ms) but higher than PL (set to 8 ms). This means that a latency lower than 8 ms is tolerated without priority, while a maximum allowed latency is set to 15 ms. The results show that, before congestion, VNF traffic has a latency of around 600 μs (traffic generator-switch-VNF-switch-traffic analyzer). With congestion, LC traffic is prioritized and not subject to congestion delay, thus maintaining the e2e latency under 1 ms. Moreover, no packet loss is experienced for LC traffic. The remaining traffic flows, both co-routed (BE) and transit, experience congestion and are both shaped and delayed. This is the expected effect of setting a reduced value of PL: allowing improved priority traffic to avoid incoming congestion events at the switch.

In test 3, reported in Fig. 14, the congestion latency is higher than the configured PL (set to 5 ms) and DL (set to 8 ms). The behavior is similar to the pattern observed in test 2, however in this case a significant LC traffic shape occurs with a drop rate of around 99%, given the strict latency constraint imposed by DL value.

4.1.3. Impact of PL and DL tuning

The role of PL and DL configurations and the related switch behavior is understood better when the switch is tested against different latency conditions. The two main latency sources in UC1 are the following: the accumulated VNF latency measured between interface 1 and interface 2, and the congestion latency at the switch. In the following tests we evaluate such contributions and the role of PL and DL in the latency control performance. In these tests the transmission rates are the same of the previous tests (LC and BE set to 1000 packets/s, transit set to 7000 packets/s).

In a first test session, we evaluate only the impact of the VNF delay without any congestion event. To emulate such delay, we exploit the *tc* linux command to the VNF outgoing interface adding an artificial delay of 5 ms. Then, we measure the latency and the drop rate of VNF flows as a function of different values of DL (PL is set to 1 ms). The results are reported in Fig. 15. Results show that BE traffic is always forwarded with a constant end-to-end (e2e) latency of around 5.42 ms. However, LC traffic is fully dropped by the switch when DL < 5.5 ms.

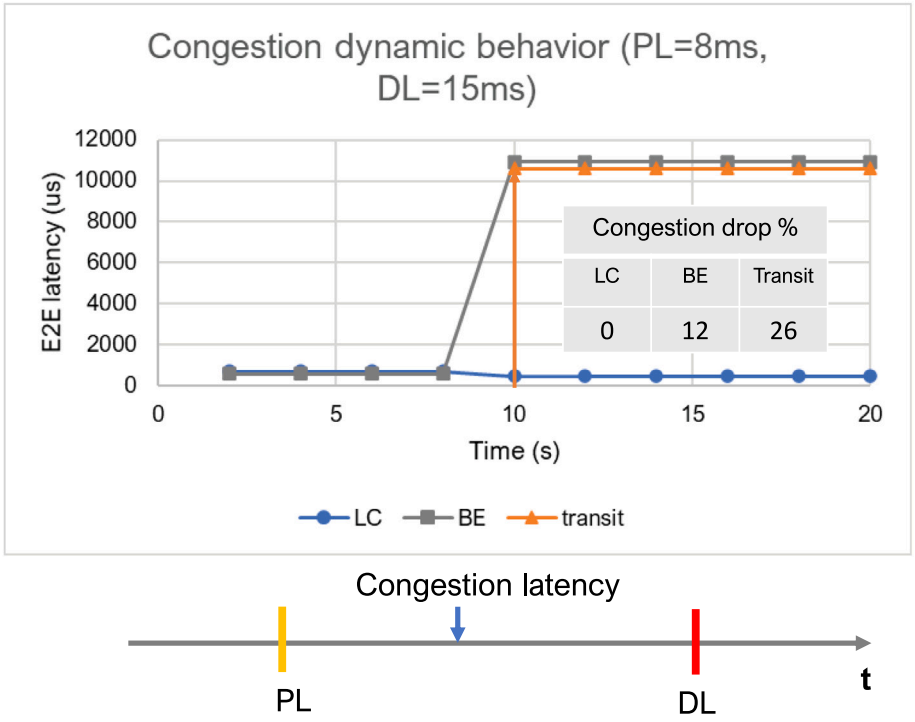


Fig. 13. Test 2.

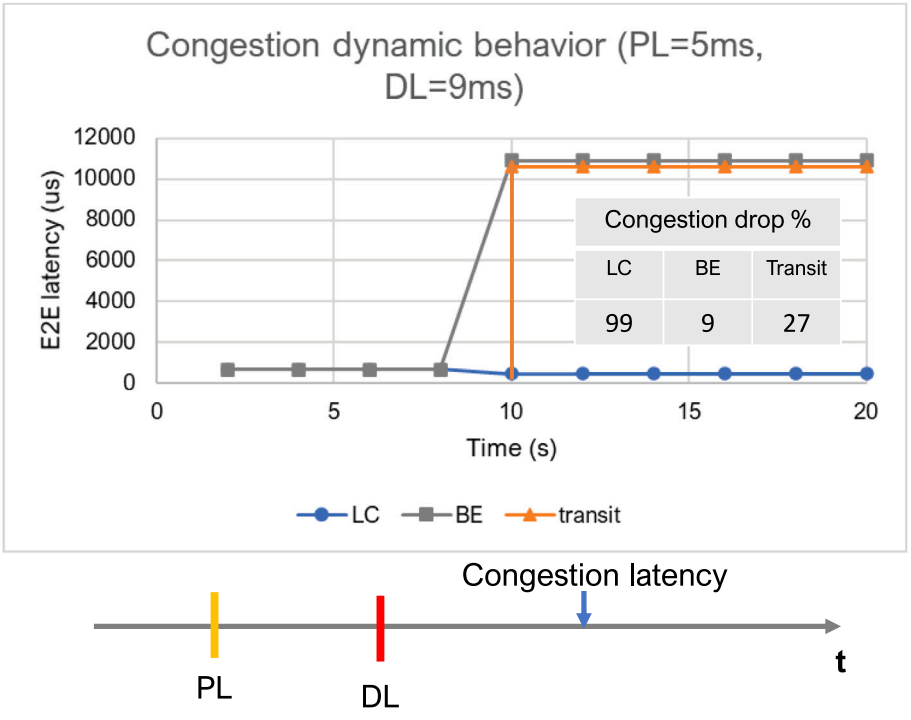


Fig. 14. Test 3.

This because the accumulated latency is always higher than the DL threshold, meaning that this portion of traffic is considered out-of-dated. Intermediate shaping is performed for DL in the range 5.5–7 ms, allowing only the packets with acceptable latency in the distribution. When DL > 7 ms, all LC traffic is allowed and forwarded. The PL configuration has not practical effect since priority is effective only in the case of congestion.

In a second test session, we evaluate only the impact of the switch congestion, reproducing the scenario shown in the dynamic behavior tests. To this goal, the transmission queue rate of interface 3 has been limited to 6000 packets/s. In this test session, first we evaluate the role of PL in the behavior of the switch for VNF traffic, then we evaluate the role of DL. The impact of PL is shown the plot reported in Fig. 16. For these tests, DL was set to a very high value in order to be not effective (i.e., traffic is never dropped by the latency control pipeline). Results

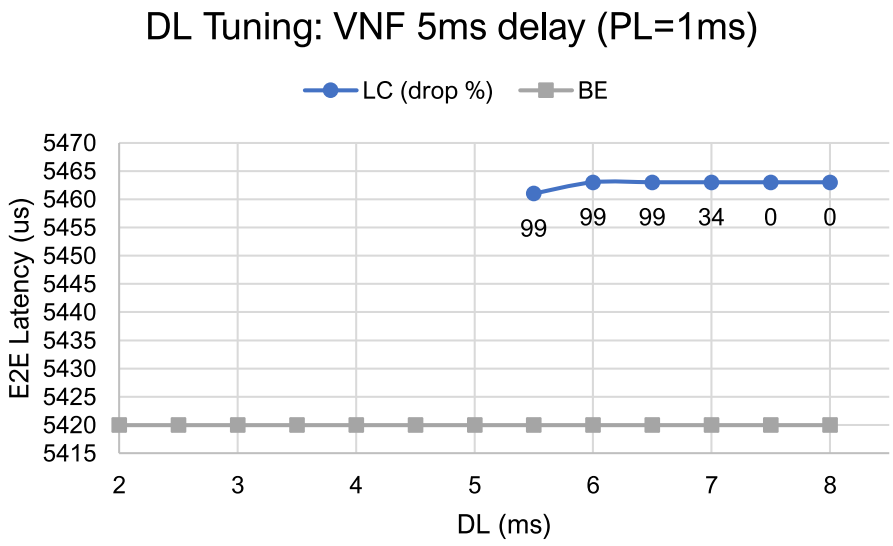


Fig. 15. Impact of DL tuning on delayed VNF.

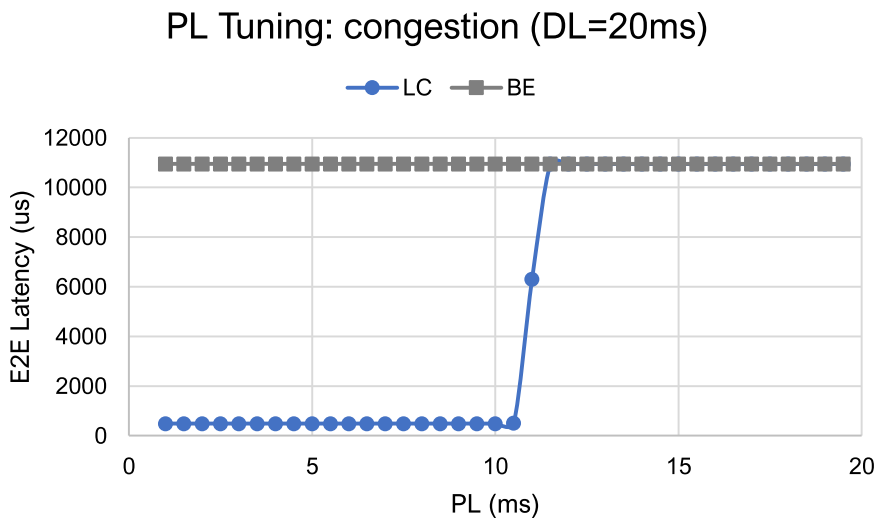


Fig. 16. Impact of PL tuning on a congested scenario.

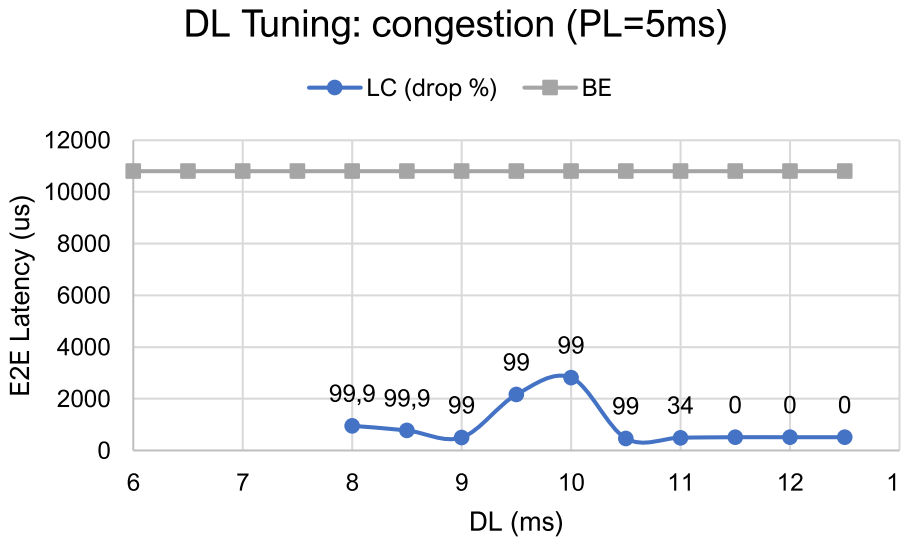


Fig. 17. Impact of DL tuning on a congested scenario.

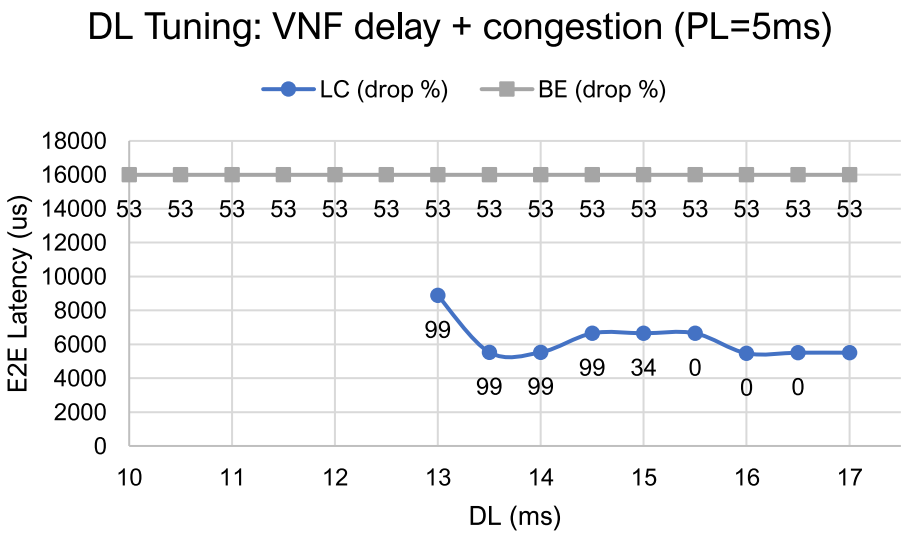


Fig. 18. Impact of DL tuning on a VNF-delayed and congested scenario.

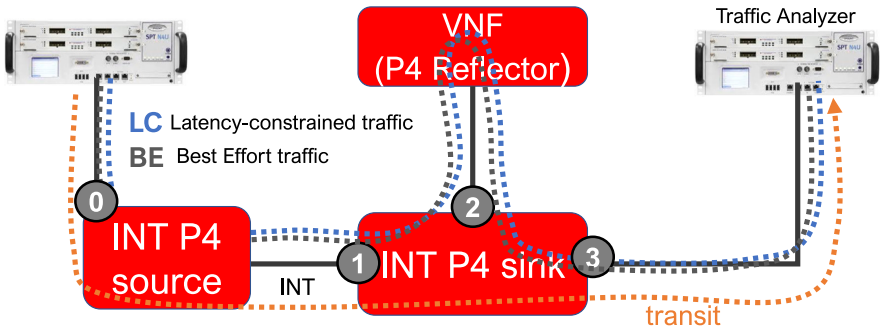


Fig. 19. Testbed used to evaluate Use Case 2.

show that low values of PL activate priority change for LC traffic with 0% drop rate until PL reaches the average congestion time. For higher values, priority is not activated and LC latency equals BE latency with drop rates higher than zero, due to the queue saturation. The impact of DL is shown in Fig. 17. In this case PL is set to a low value, so that all LC packets are prioritized. The plot shows that LC is shaped with different drop rates by the latency control policy until DL < 11.5 ms, always maintaining the latency under 5 ms. Then, when DL > 11.5 ms, all traffic is forwarded with priority and reduced latency (around 400 μs) due to the priority.

The last test session includes both VNF delay and congestion and results are plotted in Fig. 18. Traffic is delayed at the VNF and then congested at the P4 switch. The BE traffic is always subject to 53% drop and 16 ms latency. Latency control on LC traffic induces a total drop until DL < 13 ms latency, decreasing traffic shaping with 13 ms < DL < 15 ms. Then, for DL > 15 ms all traffic is forwarded with the minimum achievable latency, provided that the delay accumulated at the VNF may not be reduced.

The whole tests show that the impact of PL is extremely important when congestion occurs at the switch level, otherwise it does not provide any clear advantage. Conversely, the impact of DL is important in each context, since it sets a threshold for the forwarding of packets having a maximum tolerated amount of accumulated latency, acting as a selective latency-based traffic shaper.

4.2. UC2 setup and results

The setup shown in Fig. 19 is utilized to perform the UC2 results. An additional P4 switch is inserted in the network device chain acting as INT source node. The second P4 switch acts as INT sink node and as latency control switch. The generator is attached to two 10 Gb Ethernet interfaces and is configured to generate and analyze the same three different flows considered in UC1 tests. INT, forwarding and latency control configurations are done using specific flow entries at the P4 switches using the BMv2 command line interface (CLI). The P4 code used for INT P4 source and INT P4 sink switches is the same code, putting together INT and latency control functions. The VNF reflector code is the same described in UC1 tests.

The flow entries configured in the sink switch are shown in Fig. 20. The entries related to latency control are the same with respect to UC1. The additional entries are related to the INT sink operation. In particular, the switch considers interface 1 as the incoming interface to perform INT operation, moreover it acts as sink switch for traffic destined to interface 2 (instruction int_set_sink). This means that traffic destined to interface 2 will perform INT header pop operation. The switch id has been set to 0 × 0d (13).

The results of UC2 are incremental with respect to UC1, meaning that the general behavior of the switch, the tuning behavior of PL and DL is the same of UC1.

The traffic captured at the outgoing interface of INT source node is shown in Fig. 21. The capture shows the three traffic flows: the LC traffic (exploded in the capture), the BE traffic (tagged as DIS packets) and the transit traffic (tagged as IP traffic, since no L4 header has been added). In particular, LC traffic is tagged with a new header setting the total packet length to 1020 bytes (it was 996 in the previous capture). The whole INT header is 24-byte long. The header is placed after the UDP header and can be observed by noting that it is not recognized by Wireshark and is embedded within the layer 4 payload. In fact, the

```
#MBDA FLOW RULES UC2 SINK

#MAX_ALLOWED_ACCUMULATED_LATENCY_TO_MAINTAIN_STANDARD_PRIORITY(PL)
register_write bounded_latency 0 12750

#MAX_ALLOWED_ACCUMULATED_LATENCY_TO_PERMIT_FORWARDING(DL)
register_write drop_latency 0 12750

#FLOW RULE CONGESTION EMULATION
set_queue_rate 6000 3

#QOS TRAFFIC FORWARDING
table_add table0 set_egress_port 1 00:04:aa:00:00:00 00:04:aa:00:00:01 => 2
table_add table0 set_egress_port 2 00:04:aa:00:00:01 00:04:aa:00:00:03 => 3
table_add table0 set_egress_port 1 00:04:00:00:00:00 00:04:00:00:00:01 => 2
table_add table0 set_egress_port 2 00:04:00:00:00:01 00:04:00:00:00:03 => 3
#STANDARD TRAFFIC FORWARDING
table_add table0 set_egress_port 1 00:04:00:00:00:10 00:04:00:00:00:03 => 3

#QOS LATENCY TREATMENT
table_add table_store store_timestamp1 1 10.10.10.1 10.10.10.2 43000 45000 =>
table_add table_enforce enforce 2 10.10.10.2 10.10.10.3 45000 47000 =>

#EGRESS CONGESTION CHECK
table_add table_congestion_check do_update_congestion_reg 1 00:04:00:00:00:10 00:04:00:00:00:03 =>

#INT PROCESSING SINK NODE
table_add tb_set_source int_set_source 1 =>
table_add tb_set_sink   int_set_sink   2 =>
table_add tb_int_source int_source_dscp 10.10.10.1 10.10.10.2 17 => 0x40 0x03 0x0a 0x04
table_add tb_int_inst_0003  int_set_header_0003_i10 0x0a =>
table_add tb_int_inst_0407  int_set_header_0407_i4  0x04 =>
table_add tb_int_insert     init_metadata 0x01  => 0x0d
```

Fig. 20. Sink P4 switch flow entries for UC2 tests.

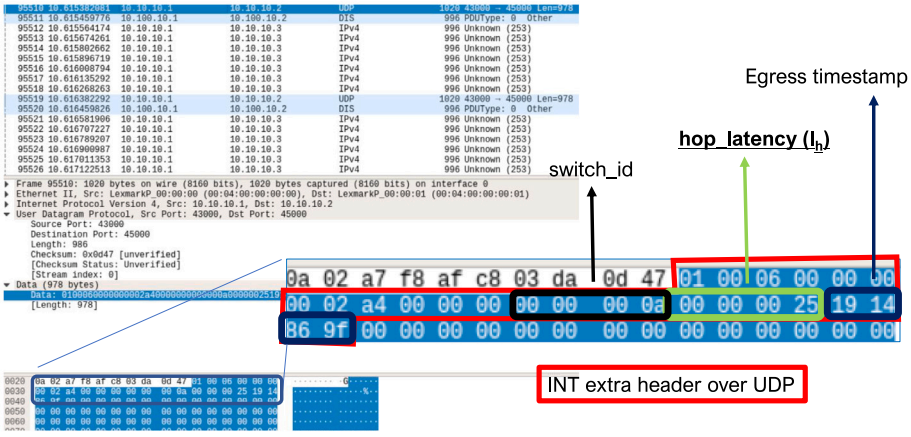


Fig. 21. Wireshark capture of UC2 traffic showing the INT header.

first 24 bytes of the payload are exactly the INT header fields. The fields are shown in the figure: the first 4 bytes are the shim header, the second eight bytes are the INT header (the value 0 × a4 represents the Instruction bitmap selecting switch_id, hop latency and ingress timestamp), then the switch id reports the value 0 × 0a (switch id 10), the hop latency of the packet (0 × 25, corresponding to 37 μs) and the ingress timestamp of the packet inside the source node (0 × 1914869f, corresponding to the value of 420.775.583 μs, around 7 min, after the bootstrap of the switch). Additional captures and measurements have been performed to derive the measured latency introduced by the VNF when no artificial delay is added. The baseline VNF transit time contribution is around 300 μs.

4.2.1. *Results with no congestion*

This set of results evaluate the role of DL tuning due to the upstream INT chain in absence of congestion. The traffic conditions are the same of UC1 in order to allows fair comparison between the two use cases. Transit traffic rate is set to 7000 packet/s, while LC and BE rates are set to 1000 packet/s each. No congestion is configured. The tuning effect of DL is shown in Fig. 22. Transit traffic latency is the lowest one (around 200 μs), due to the shortest path. BE traffic has a constant latency of around 680 μs, higher than transit traffic due to the longest paths including the VNF processing. The LC traffic latency is tuned by the configuration of DL. In fact, LC is dropped when DL < 350 μs, meaning that all packets exhibit accumulated latency always higher than 350 μs. Then, intermediate shaping is performed by the sink switch for 350 μs < DL < 700 μs. That is, the switch drops only the LC packets that do not satisfy the DL condition. Drop rate passes from 99% at DL = 350 μs to 0.07% at DL = 650 μs with a shaping behavior of the latency from 550 μs to 700 μs. The behavior shows a very good accordance between the DL configuration and the average obtained packet latency values. Some latency offset is present due to the measurement setup: traffic analyzer measures the e2e latency between the interfaces of the generator, including the interfaces transmission time and the propagation time of each link, while the DL threshold considers only the hop latency of the source switch and the switch+VNF segment. It is worth to note that all the latency contributions not considered in the DL threshold are constant (interface transmission time, link propagation time) and may be computed in advance and added to the DL threshold budget. The same considerations are applicable to the PL parameter in the case of congestion.

Table 2 show the comparison, in terms of average end-to-end latency of the three traffic flows in steady state conditions, i.e. in absence of congestion and VNF delay, measured in both UC1 and UC2 scenario. The results show that UC2 code increases the average latency

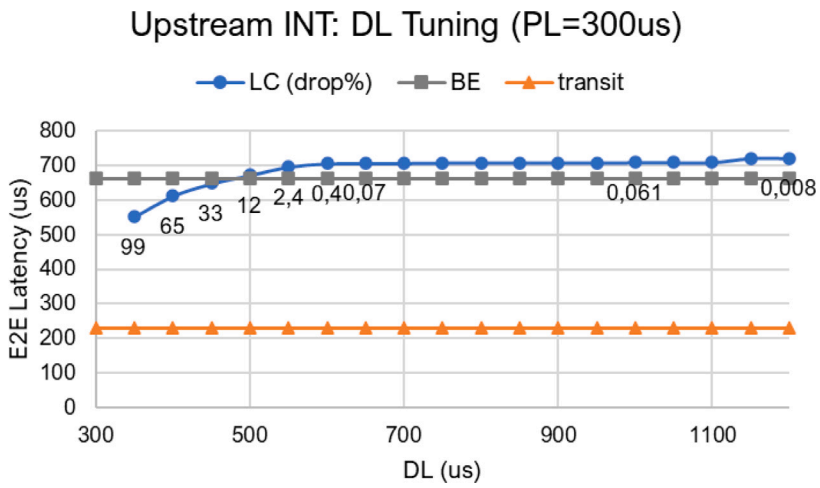


Fig. 22. UC2 without congestion: End-to-end latency and traffic shaping results as a function of DL tuning.

Table 2
End-to-end latency results: steady state.

Traffic flow	UC1 end-to-end latency (μs)	UC2 end-to-end latency (μs)
Overall LC	433	664
LC - VNF	133	364
Overall BE	379	557
BE - VNF	79	257
Transit	70	187

of the switch. Subtracting the VNF processing contribution, the UC2 code impacts increasing the latency of a factor 3. This is mainly due to the additional source switch and to the increased complexity of INT processing. This is demonstrated by the baseline results of the transit traffic, subject to a latency increase rate of 2.6. The obtained results demonstrate that the latency control contribution to P4 processing is kept limited even with respect to INT processing, confirming the lightweight burden introduced by latency control functions. With respect to the same traffic route experienced by packets without considering the VNF delay, UC1 LC code introduces additional 54 μs (i.e., 27 μs average at each switch transit), while UC2 LC code introduces additional 107 μs (i.e., 35 μs average at each switch transit). Therefore, the impact of the additional P4 code processing is limited with the assumption that the latency control granularity is in the order of few milliseconds.

4.2.2. Results with congestion

The results of Fig. 23 show the behavior of the observed maximum packet latency as a function of the configured DL = PL in the case of switch congestion, evaluated in the same conditions evaluated for UC1. In this case the maximum evaluated end-to-end latency is measured by the traffic analyzer as a single packet latency occurrence evaluated in the measurement set obtained by evaluating the behavior of 100 million of transmitted frames. The behavior confirms that the tune of DL allows a shaping of delayed packets in a reasonably accordance with the imposed DL threshold.

4.3. Applicability to hardware programmable platforms

The results reported in the previous subsections are related to BMv2 software switch and to traffic loads in the order of tens of Mb/s. Results show that the impact of latency control with respect to standard forwarding is limited, scalable in terms of flow entries and effective in terms of LC latency control, either in static and dynamic conditions. However, the proposed mechanism is conceived to be applicable in P4 switches featuring hardware programmability and sustaining line rate interface throughput (e.g., 100 Gb/s and more). A discussion is hereafter needed to justify the adoption of such workflows on these innovative platforms.

The key P4 capabilities needed to run LC workflows are the conditional pipeline execution, the extra header processing (in UC2), the queueing metadata support and the availability of stateful registers. Most of the currently available hardware programmable switches support such capabilities. The key applicability constraint relies on the dimension of the mobile window timestamp registers.

The Intel Tofino has the capability to program the P4 logic in the ASIC utilizing up to 12 Match Action Units (MAU) for the ingress and egress pipelines. Each MAU has the availability of Shadow Random Access Memory (SRAM), Ternary Content Addressable Memory (TCAM), Hash, Arithmetic Logical Units (ALU) and stateful ALUs [39–41]. Thanks to the flexible pipeline design, flow tables of the different pipelines may be split in different MAU, thus the dimension of each flow table does not impact significantly in the architecture. The only scalability issue may be represented by the dimension of the registers storing packet timestamps. In the proposed P4 implementation, two arrays of registers are used to store $t1$ and $t2$ timestamps. Further code optimization may be considered by deleting the $t2$ register without significant performance issues. Assuming a 48-bit long timestamp (as implemented in BMv2) and a 100 Gb/s line rate interface with 10% total LC traffic (i.e., 10 Gb/s) with an average packet length of 500 byte and an upper bound VNF latency of 20 ms, the minimum register array memory size preventing packet timestamp overloading is around the 2% of the overall SRAM and TCAM resources available in the switch dataplane. This means that a switch has sufficient memory resources to instantiate the registers properly and provide latency control of a large set of LC flows with an aggregate 10 Gb/s load.

5. Conclusions

In this paper we proposed the adoption of P4-based data plane programmability to control the latency of Service Chained flows at the network switches in both intra- and inter-data center service deployment. A detailed set of policy algorithms and P4 codes have been presented in order to guarantee proactive actions at the switches, such as priority change and drop, on a per-packet fashion when flows related to the same service chaining crossing the same switch multiple times. The proposed methods were applied in both a stand-alone switch scenario and a switch domain supporting in-band telemetry capabilities. Extensive results over a network testbed deploying BMv2 switches were reported, showing that the presented mechanisms assure a fine per-packet end-to-end latency control at the millisecond granularity with good scalability margins in terms of admitted forwarding flow

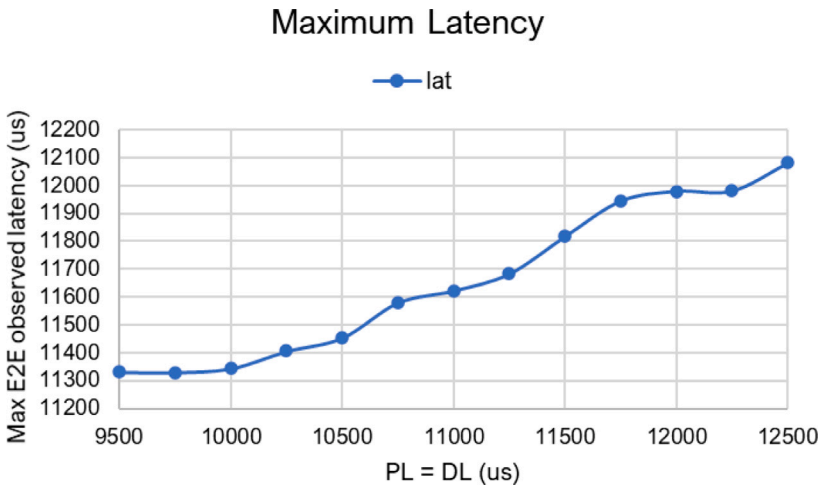


Fig. 23. UC2 with congestion: maximum end-to-end latency as a function of DL and PL tuning.

entries. Moreover, the impact of the latency control operation in terms of introduced intra-switch latency is kept limited, below 40 μs, with respect to the working switch latency values. The applicability of such methods have been analyzed for hardware bare metal switches such as the Tofino platform, showing that the amount of required resources in terms of pipelines and memory are more than sufficient to enable wirespeed transmission at line rate interface for a significant fraction of latency-controlled service chained traffic.

CRediT authorship contribution statement

Francesco Paolucci: Conceptualization, Methodology, Investigation, Visualization, Validation, Writing – original draft, Writing – review & editing. **Davide Scano:** Software, Validation. **Piero Castoldi:** Conceptualization, Supervision. **Emiliano De Paoli:** Conceptualization, Methodology.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Francesco Paolucci: reviewers with affiliation to CNIT and Scuola Superiore Sant'Anna.

Davide Scano: reviewers with affiliation to CNIT and Scuola Superiore Sant'Anna.

Piero Castoldi: reviewers with affiliation to CNIT and Scuola Superiore Sant'Anna.

Emiliano De Paoli: reviewers with affiliation to MBDA, Leonardo, BAE Systems, Airbus.

Data availability

The authors do not have permission to share data.

References

[1] L. Cui, F.P. Tso, W. Jia, Federated service chaining: Architecture and challenges, IEEE Commun. Mag. 58 (3) (2020) 47–53, http://dx.doi.org/10.1109/MCOM.001.1900627.

[2] S. Van Rossem, W. Tavernier, D. Colle, M. Pickavet, P. Demeester, VNF performance modelling: From stand-alone to chained topologies, Comput. Netw. 181 (2020) 107428, http://dx.doi.org/10.1016/j.comnet.2020.107428, URL https://www.sciencedirect.com/science/article/pii/S1389128620311178.

[3] H. Hantouti, N. Benamar, T. Taleb, Service function chaining in 5G amp; beyond networks: Challenges and open research issues, IEEE Netw. 34 (4) (2020) 320–327, http://dx.doi.org/10.1109/MNET.001.1900554.

[4] A.A. Barakabitze, A. Ahmad, R. Mijumbi, A. Hines, 5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges, Comput. Netw. 167 (2020) 106984, http://dx.doi.org/10.1016/j.comnet.2019.106984, URL https://www.sciencedirect.com/science/article/pii/S1389128619304773.

[5] K. Kaur, V. Mangat, K. Kumar, A comprehensive survey of service function chain provisioning approaches in SDN and NFV architecture, Comp. Sci. Rev. 38 (2020) 100298, http://dx.doi.org/10.1016/j.cosrev.2020.100298, URL https://www.sciencedirect.com/science/article/pii/S1574013720303981.

[6] P. Zheng, W. Feng, A. Narayanan, Z.-L. Zhang, NFV performance profiling on multi-core servers, in: 2020 IFIP Networking Conference (Networking), 2020, pp. 91–99.

[7] R. Kawashima, H. Nakayama, T. Hayashi, H. Matsuo, Evaluation of forwarding efficiency in NFV-nodes toward predictable service chain performance, IEEE Trans. Netw. Serv. Manag. 14 (4) (2017) 920–933, http://dx.doi.org/10.1109/TNSM.2017.2734560.

[8] M. Faltelli, G. Belocchi, F. Quaglia, S. Pontarelli, G. Bianchi, Metronome: Adaptive and precise intermittent packet retrieval in DPDK, in: Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies, in: CoNEXT '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 406–420, http://dx.doi.org/10.1145/3386367.3432730.

[9] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker, P4: Programming protocol-independent packet processors, SIGCOMM Comput. Commun. Rev. 44 (3) (2014) 87–95, http://dx.doi.org/10.1145/2656877.2656890.

[10] P4: https://p4.org.

[11] S. Han, S. Jang, H. Choi, H. Lee, S. Pack, Virtualization in programmable data plane: A survey and open challenges, IEEE Open J. Commun. Soc. 1 (2020) 527–534, http://dx.doi.org/10.1109/OJCOMS.2020.2990182.

[12] E.F. Kfoury, J. Crichigno, E. Bou-Harb, An exhaustive survey on P4 programmable data plane switches: Taxonomy, applications, challenges, and future trends, IEEE Access 9 (2021) 87094–87155, http://dx.doi.org/10.1109/ACCESS.2021.3086704.

[13] F. Paolucci, F. Cugini, P. Castoldi, T. Osiński, Enhancing 5G SDN/NFV edge with P4 data plane programmability, IEEE Netw. 35 (3) (2021) 154–160, http://dx.doi.org/10.1109/MNET.021.1900599.

[14] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, P. Castoldi, P4 edge node enabling stateful traffic engineering and cyber security, IEEE/OSA J. Opt. Commun. Networking 11 (1) (2019) A84–A95.

[15] F. Musumeci, A. Fidanci, F. Paolucci, F. Cugini, M. Tornatore, Machine-learning-enabled DDoS attacks detection in P4 programmable networks, J. Netw. Syst. Manage. 30 (21) (2022) early–access, http://dx.doi.org/10.1007/s10922-021-09633-5.

[16] Behavioral Model version 2: https://github.com/p4lang/behavioral-model.

[17] F. Cugini, P. Gunning, F. Paolucci, P. Castoldi, A. Lord, P4 in-band telemetry (INT) for latency-aware VNF in metro networks, in: Optical Fiber Communication Conference (OFC) 2019, Optical Society of America, 2019, p. M3Z.6, http://dx.doi.org/10.1364/OFC.2019.M3Z.6, URL http://www.osapublishing.org/abstract.cfm?URI=OFC-2019-M3Z.6.

[18] L. Linguaglossa, S. Lange, S. Pontarelli, G. Rétvári, D. Rossi, T. Zinner, R. Bifulco, M. Jarschel, G. Bianchi, Survey of performance acceleration techniques for network function virtualization, Proc. IEEE 107 (4) (2019) 746–764, http://dx.doi.org/10.1109/JPROC.2019.2896848.

[19] A. Leivadeas, M. Falkner, N. Pitaev, Analyzing service chaining of virtualized network functions with SR-IOV, in: 2020 IEEE 21st International Conference on High Performance Switching and Routing (HPSR), 2020, pp. 1–6, http://dx.doi.org/10.1109/HPSR48589.2020.9098975.

[20] A. Ben Hamed, A. Leivadeas, M. Falkner, N. Pitaev, VNF chaining performance characterization under multi-feature and oversubscription using SR-IOV, Informatics 7 (3) (2020) http://dx.doi.org/10.3390/informatics7030033, URL https://www.mdpi.com/2227-9709/7/3/33.
[21] M.S. Castanho, C.K. Dominicini, M. Martinello, M.A.M. Vieira, Chaining-box: A transparent service function chaining architecture leveraging BPF, IEEE Trans. Netw. Serv. Manag. 19 (1) (2022) 497–509, http://dx.doi.org/10.1109/TNSM.2021.3122135.
[22] C.K. Dominicini, G.L. Vassoler, R. Valentim, R.S. Villaca, M.R. Ribeiro, M. Martinello, E. Zambon, KeySFC: Traffic steering using strict source routing for dynamic and efficient network orchestration, Comput. Netw. 167 (2020) 106975, http://dx.doi.org/10.1016/j.comnet.2019.106975, URL https://www.sciencedirect.com/science/article/pii/S138912861930194X.
[23] C. Sun, J. Bi, Z. Zheng, H. Yu, H. Hu, NFP: Enabling network function parallelism in NFV, in: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, in: SIGCOMM '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 43–56, http://dx.doi.org/10.1145/3098822.3098826.
[24] D. Zhang, X. Chen, Q. Huang, X. Hong, C. Wu, H. Zhou, Y. Yang, H. Liu, Y. Chen, P4SC: A high performance and flexible framework for service function chain, IEEE Access 7 (2019) 160982–160997, http://dx.doi.org/10.1109/ACCESS.2019.2950446.
[25] D. Wu, A. Chen, T.S.E. Ng, G. Wang, H. Wang, Accelerated service chaining on a single switch ASIC, in: Proceedings of the 18th ACM Workshop on Hot Topics in Networks, in: HotNets '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 141–149, http://dx.doi.org/10.1145/3365609.3365849.
[26] F. Civerchia, A. Sgambelluri, F. Paolucci, L. Maggiani, P. Castoldi, F. Cugini, Hardware acceleration for processing function virtualization, in: Proceedings of the IEEE International Mediterranean Conference on Communications and Networking (MeditCom), IEEE, 2021.
[27] J. Ma, S. Xie, J. Zhao, Flexible offloading of service function chains to programmable switches, IEEE Trans. Serv. Comput. (2022) 1, http://dx.doi.org/10.1109/TSC.2022.3162701.
[28] A. Stockmayer, S. Hinselmann, M. Häberle, M. Menth, Service function chaining based on segment routing using P4 and SR-IOV (P4-SFC), in: H. Jagode, H. Anzt, G. Juckeland, H. Ltaief (Eds.), High Performance Computing, Springer International Publishing, Cham, 2020, pp. 297–309.
[29] D.R. Mafioletti, C.K. Dominicini, M. Martinello, M.R.N. Ribeiro, R.d.S. Villaça, PIaFFE: A place-as-you-go in-network framework for flexible embedding of VNFs, in: ICC 2020 - 2020 IEEE International Conference on Communications (ICC), 2020, pp. 1–6, http://dx.doi.org/10.1109/ICC40277.2020.9149240.
[30] J. Lee, H. Ko, H. Lee, S. Pack, Flow-aware service function embedding algorithm in programmable data plane, IEEE Access 9 (2021) 6113–6121, http://dx.doi.org/10.1109/ACCESS.2020.3048421.
[31] D. Scano, F. Paolucci, K. Kondepu, A. Sgambelluri, L. Valcarenghi, F. Cugini, Extending P4 in-band telemetry to user equipment for latency- and localization-aware autonomous networking with AI forecasting, J. Opt. Commun. Netw. 13 (9) (2021) D103–D114, http://dx.doi.org/10.1364/JOCN.425891, URL http://opg.optica.org/jocn/abstract.cfm?URI=jocn-13-9-D103.
[32] I. Pelle, F. Paolucci, B. Sonkoly, F. Cugini, Latency-sensitive edge/cloud serverless dynamic deployment over telemetry-based packet-optical network, IEEE J. Sel. Areas Commun. 39 (9) (2021) 2849–2863, http://dx.doi.org/10.1109/JSAC.2021.3064655.
[33] D. Cho, J. Taheri, A.Y. Zomaya, L. Wang, Virtual network function placement: Towards minimizing network latency and lead time, in: 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 2017, pp. 90–97, http://dx.doi.org/10.1109/CloudCom.2017.12.
[34] G. Sallam, G.R. Gupta, B. Li, B. Ji, Shortest path and maximum flow problems under service function chaining constraints, in: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, 2018, pp. 2132–2140, http://dx.doi.org/10.1109/INFOCOM.2018.8485996.
[35] Y. Wang, C.-K. Huang, S.-H. Shen, G.-M. Chiu, Adaptive placement and routing for service function chains with service deadlines, IEEE Trans. Netw. Serv. Manag. 18 (3) (2021) 3021–3036, http://dx.doi.org/10.1109/TNSM.2021.3086977.
[36] T. Mitra, J. Teich, L. Thiele, Time-critical systems design: A survey, IEEE Des. Test 35 (2) (2018) 8–26, http://dx.doi.org/10.1109/MDAT.2018.2794204.
[37] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, P. Stenström, The worst-case execution-time problem—Overview of methods and survey of tools, ACM Trans. Embed. Comput. Syst. 7 (3) (2008) http://dx.doi.org/10.1145/1347375.1347389.
[38] In-band Network Telemetry Specifications v.2.1. https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_1.pdf.
[39] P4_16 for Intel Tofino using Intel P4 Studio: https://opennetworking.org/wp-content/uploads/2021/05/2021-P4-WS-Vladimir-Gurevich-Slides.pdf.
[40] N. Budhdev, R. Joshi, P.G. Kannan, M.C. Chan, T. Mitra, FSA: Fronthaul slicing architecture for 5G using dataplane programmable switches, in: Proceedings of the 27th Annual International Conference on Mobile Computing and Networking, in: MobiCom '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 723–735, http://dx.doi.org/10.1145/3447993.3483247.
[41] SD Fabric TNA: https://github.com/stratum/fabric-tna.

Francesco Paolucci (M) received the Laurea degree in telecommunications engineering from the University of Pisa in 2002, and the Ph.D. degree from Scuola Superiore Sant'Anna, Pisa, in 2009. In 2008 he was granted a research Merit Scholarship at the Istitut National de le Recherche Scientifique (INRS), Montreal, Quebec, Canada. Currently, he is Senior Researcher at CNIT, Pisa Italy. His main research interests are in the field of network control plane, orchestration for edge/cloud platforms, traffic engineering, network disaggregation, advanced network telemetry, SDN/P4 data plane programmability. He has been involved in many European research projects on next generation control networking (E-Photon/ONe+, BONE, NOBEL, STRONGEST, IDEALIST, PACE, 5GEx, 5GTRANSFORMER, METROHAUL, 5Growth, BRAINE). He is co-author of 2 IETF Internet Drafts, more than 170 publications in international journals, conference proceedings and book chapters, and filed 4 international patents. He is Associate Editor of the IEEE/OSA Journal of Optical Communications and Networking (JOCN).

Davide Scano received his B.S. in telecommunication engineering from the University of Pisa (2017) and his M.S in computer science and networking from the University of Pisa and Scuola Superiore Sant'Anna (2019), discussing a research thesis on software defined networking for guarantee QoS in network slicing. In 2020 he got a Research Scholarship at Scuola Superiore Sant'Anna, Pisa. Currently, he is Ph.D. student at Scuola Superiore Sant'Anna. His research interests are software defined networking, network slicing, optical networks.

Piero Castoldi is a Full Professor and leader of the "Networks and Services" research area at the TeCIP Institute of Scuola Superiore Sant'Anna, Pisa, Italy. He has been involved with various responsibilities in several national and EU FP7 and H2020 projects and he has managed several corporate-sponsored projects with the italian Railway Infrastructure Company, Ericsson, Telecom Italia. He is the Director of the Erasmus Mundus Master on Photonic Integrated Circuits, Sensors and Networks (PIXNET). His most recent research interests lie in the areas of optical network architectures, interconnection networks for Data Centers, networks for industrial applications, 5G networking. He is author of more than 400 technical papers published in international journals and international conference proceedings. He has also filed more than 20 patents and he has authored a book on multiuser detection for CDMA mobile terminals published by Artech House.

Emiliano De Paoli is CyberSecurity Technical Expert at MBDA Italy. In the Engineering department of the Company, he is leading the Research and Development national team for cybersecurity and communications domains. He follows collaborations with universities, research centers and SMEs.