Received 25 January 2023, accepted 16 February 2023, date of publication 24 February 2023, date of current version 9 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3249105

RESEARCH ARTICLE

Enabling P4 Network Telemetry in Edge Micro Data Centers With Kubernetes Orchestration

DAVIDE SCANO[1], ALESSIO GIORGETTI[2,3], FRANCESCO PAOLUCCI[3], ANDREA SGAMBELLURI[1], JAVAD CHAMMANARA[4], JOHN ROTHMAN[4], MUSTAFA AL-BADO[5], EDGARD MARX[6,7], SEAN AHEARNE[5], AND FILIPPO CUGINI[3], (Member, IEEE)

[1]TECIP Institute, Scuola Superiore Sant'Anna, 56127 Pisa, Italy
[2]Institute of Electronics, Computers and Telecommunication Engineering, CNR-Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni (CNR-IEIIT), 56124 Pisa, Italy
[3]PNTLab, National, Inter-University Consortium for Telecommunications (CNIT), 56127 Pisa, Italy
[4]L3s Research Center, Leibniz University Hannover, 30167 Hanover, Germany
[5]Dell Technologies, Cork, P31 D253 Ireland
[6]eccenca GmbH, 04109 Leipzig, Germany
[7]AKSW, Leipzig University of Applied Sciences (HTWK), 04277 Leipzig, Germany

Corresponding author: Davide Scano (davide.scano@santannapisa.it)

This work was supported in part by the Big Data Processing and Artificial Intelligence at the Network Edge (BRAINE) Project, through Electronics Components and Systems for European Leadership (ECSEL) Joint Undertaking (JU), under Agreement 876967; in part by The JU, European Union (EU) Horizon 2020 Research and Innovation Programme; and in part by the Italian Ministry of Education, University, and Research (MIUR).

ABSTRACT Integrating computation resources with networking technologies is an hot research topic targeting the optimization of containers deployment on a set of host machines interconnected by a network infrastructure. Particularly, next generation edge nodes will offer significant advantages leveraging on integrated computation resources and networking awareness, enabling configurable, granular and monitorable quality of service to different micro-services, applications and tenants, especially in terms of bounded end-to-end latency. In this regard, SDN is a key technology enabling network telemetry and traffic switching with the granularity of the single traffic flow. However, currently available solutions are based on legacy SDN techniques, not enabling the matching of tunneled traffic, and thus require a tricky integration inside the hosts where containers are deployed. This work considers Kubernetes clusters deployed on next generation edge micro data center platforms and proposes an innovative SDN solution exploiting the P4 technology to gain visibility inside tunnelled traffic exchanged among pods. This way, the integration is achieved at the control plane level through the communication between Kubernetes and the SDN controller. The proposed solution is experimentally validated including a comprehensive framework enabling effective traffic switching and in-band telemetry at pod level. The major paper contributions consist in the design and the development of: (i) the networking applications at SDN control plane level; (ii) the P4 switch pipeline at the data plane level; (iii) the monitoring system used to collect, aggregate and elaborate the telemetry data.

INDEX TERMS SDN, P4, telemetry, micro data center, kubernetes.

I. INTRODUCTION

The relation between computation and networking technologies has become stronger and stronger especially in the cloud environment where each application is a composition of micro-services potentially running on different machines [1], [2]. Moreover, since the services are moving toward the network edge (e.g., due to stringent latency requirements), where limited resources are typically available, the joint optimization of computational and networking resources is currently a crucial challenge [3], [4]. To solve this issue both centralized and distributed approaches have been proposed in literature [5].

Focusing on computational resources orchestration, Kubernetes has recently become the de-facto framework to orchestrate containers in both data-centers and edge-cloud environments [6]. Kubernetes is built on the concept of *pod* (i.e., a deployment unit) consisting of one or more

The associate editor coordinating the review of this manuscript and approving it for publication was Cesar Vargas-Rosales.

containers running on the same host machine. Only high level indications are provided by the Kubernetes community about the way in which pod networking should be implemented, i.e., regarding pod-to-pod communication the only specified requirement is that *each pod can communicate with all other pods on any other node without Network Address Translation(NAT)* [7]. However, the actual networking implementation is delegated to third-party plugins, i.e., the Container Network Interface (CNI) plugins, which may adopt different configuration of the pods network interfaces for providing pod-to-pod connectivity (e.g., adopting different tunneling technologies). Moreover, CNI plugins typically do not account for network constraints in terms, for example, of required bandwidth or bounded latency. For this reason, deploying Kubernetes in edge computing environments requires specifically designed and comprehensive solutions able to provide flexible network control and traffic telemetry. This is especially true in the case Quality of service (QoS)-critical applications have to be supported and the cluster machines are distributed at different locations, e.g., over a metropolitan area network [8], [9].

The introduction of Software Defined Networking (SDN) provided the opportunity to enable effective networking resources control. Nowadays P4 (i.e., Programming Protocol-Independent Packet Processors) is the SDN solution enabling the most advanced programmability of the forwarding plane [10], also enabling innovative in-network function offloading [11]. P4 provides many additional features with respect to previous SDN solutions (e.g., OpenFlow [12]): (1) it defines a standard language to specify custom data plane pipelines; (2) it allows to define and manipulate custom packet headers through the definition of dedicated packet parsers; (3) it supports the utilization of registers enabling the development of stateful functionalities; (4) it provides improved network visibility, e.g., using In-band Network Telemetry (INT) [13]. Moreover, P4 is supported by several kind of targets, e.g., bare metal or software switches, smart-NICs, NetFPGAs. Thus, P4 enables advanced traffic telemetry in passive mode, with no need of dedicated probing packets. Three different INT techniques are proposed in the specification [14], highlighting the main technical details. In the three schemes, *report* packets are directly built in the data plane and sent toward a telemetry collection point, that can be used to perform the per-flow traffic analysis.

Considering all the aforementioned features, we believe that P4 offers a flexible framework for enabling effective integration of computational resources, orchestration tools and networking resources. Recently, OpenFlow has been considered for such integration [15], [16], [17]. However, since OpenFlow cannot have visibility inside the tunnels created by the CNI plugins, the proposed solutions require a deep integration with the plugins deploying dedicated software applications in all host machines composing the Kubernetes cluster. Conversely, exploiting P4 it is possible to define a dedicated packet parser, gaining visibility on pod-to-pod traffic flows inside the tunnels. This enables the deployment of fine granularity traffic engineering and telemetry techniques without direct interaction with the data plane of the Kubernetes pod network. Thus, and this is our proposal, the orchestration can be implemented through interactions at the control plane level exploiting proper interfaces between Kubernetes and the SDN controller. To enable this, we proposed a specific P4 pipeline to match and monitor each pod-to-pod traffic flow encapsulated in a tunnel by a CNI plugin. The design of the pipeline is generic, while the parser is specific for the Flannel CNI plugin, however it could be easily extended to support other plugins.

To achieve the aforementioned targets, P4 devices should be effectively controlled by the SDN controller. Regarding this point, the P4 consortium defined an interface called P4Runtime [18], suitable to configure and control P4 devices (e.g., to install pipelines and flow rules). However, in the real world, P4 compatible devices do not mandatory include a P4Runtime agent. Indeed, several P4 use-cases do not rely on the utilization of an SDN controller. Therefore, another important contribution of this work is the implementation of an application, at the SDN controller side, to configure and control P4 devices, including the specific parsers and pipelines that we have designed for enabling visibility inside tunnels established by the CNI plugins. Specifically, we have considered the SDN controller developed by Open Networking Foundation(ONF), i.e., the Open Network Operating System (ONOS) [19], that is characterized by an active development community.

Finally, the work goes beyond the integration of Kubernetes with the network connecting the worker nodes, implementing also a closed-loop control where telemetry data is used to detect possible Service Level Agreement (SLA) degradation that could be recovered providing a feedback to the SDN controller or directly to the applications generating the traffic. Specifically, this last contribution consists of a P4 pipeline supporting in-band telemetry, a Telemetry Collector used to aggregate generated telemetry data and a dedicated Telemetry and Monitoring Platform to elaborate collected telemetry data.

A preliminary version of this work has been presented as a practical demonstration in [20]. In addition, this work includes: (i) the detailed research background; (ii) the overall architecture of Edge Micro Data Center nodes as designed within the BRAINE project; (iii) the extended version of the ONOS NetApps to support matching of traffic generated inside/outside Kubernetes pods; (iv) a wide set of experimental results.

II. BACKGROUND AND RELATED WORK

The joint optimization of IT and networking resources is a well established research topic as demonstrated by the wide survey reported in [5] where both centralized and distributed solutions are resumed. The former approach exploits a central element collecting resource information (e.g., interacting with the cloud orchestrator and the network controller) and typically provides improved solutions introducing a coordination overhead layer. Differently, the distributed approach is more flexible and adaptive to dynamic environments, however typically provides less effective solutions. For instance, the work in [21] proposes a distributed approach to partition a pool of computational resources among multiple applications using a dynamic agreement. On the other hand, centralized approaches have recently gained attention because they can more easily leverage on Machine Learning (ML) techniques. In [22] a ML platform is developed for effective management of both computational and networking resources in a 5G mobile environment, where data are collected from both the Kubernetes orchestrator and the SDN controller.

The actual networking implementation in Kubernetes clusters is delegated to third-party CNI plugins. The most considered ones are Flannel, Calico, and Kube-router [23]. As explained in [24], Flannel provides a layer-3 IPv4 network among multiple nodes within the cluster, i.e., it does not control how pods are networked to the host machine. Several back-end mechanisms are supported (i.e., VXLAN, UDP, host-gw and additional experimental mechanisms) but VXLAN is recommended (see Sec.III for details). With respect to Flannel, Calico [25] also provides security and policy enforcement between pods supporting a wide range of deployment options. Kube-router [26] is a specific solution for Kubernetes pods networking with the aim of providing operational simplicity and high performance. Recent research has focused on CNI plugins performance comparison in different scenarios. In [27] the performance of the most popular plugins are compared in terms of latency and average TCP throughput. The results show that Flannel and Kube-router outperform in terms of latency, while for all the plugins the TCP average throughput is close to bare-metal capacity.

Regarding network programmability, the work in [28] reports a comparison using OpenFlow and P4 for the implementation of equivalent functionalities (e.g, packet header manipulation), showing similar results in terms of throughput. However, the major potential of P4 is its suitability for a variety of use cases. In [29], a new pipeline is deployed for providing stateful traffic engineering and cyber-security on an edge node designed for a multi-layer IP over optical network. Moreover, augmented firewalling capabilities are envisioned for mitigating Distributed Denial-of-Service (DDoS) cyber attack. Additional P4 use cases for multi-layer networks are reported in [30], including the telemetry of end-to-end optical performance indicators exchanged between packet-optical nodes and P4-defined neural networks targeting online cyber-security. In [31] programmable switches are leveraged for deploying a multi-purpose ML-based security applications. It collects the packet length/inter-packet timing frequency distributions, classifying the traffic flows directly on the switches. P4 can be also used for deploying an open source framework that combines the flexibility of software-based traffic generation with the accuracy of hardware packet time-stamping, as presented in [32]. In [33], P4 is leveraged for providing Bit Index Explicit Replication (BIER), proposed by Internet Engineering Task Force (IETF) for efficient transport of Internet Protocol (IP) multicast traffic [34]. The work in [35] presents an use case of P4 registers to store stateful information achieving autonomous forwarding and low-latency path discovery. Finally, [36] proposes solutions for providing network slicing in different networking environments.

In addition to the previous use cases, in-band network telemetry is a key feature enabled by P4 that is achieved thanks to improved visibility on networks events provided by the P4 language [13]. The specification [14] proposes three different techniques, i.e., INT-XD, INT-MX, and INT-MD. In the INT-XD technique (also known as Postcard-based Telemetry (PBT)), the node directly exports, for each monitored packet, metadata from the data-plane to the monitoring system, based on the instructions configured in local flow tables. Collected metadata is inserted in a new packet called `report`, that is forwarded to the monitoring system. No packet modification is applied on the traffic packets. In INT-MX (INT-MD) instructions (and metadata) are written into traffic packets, adding a specific header (i.e, the INT header). The `report` packets are generated at each traversed node in the INT-MX technique, while in the INT-MD technique metadata are accumulated in the INT header while the packet is travelling in the network and `report` packets are generated only by sink nodes. The work in [37] provides a surveys of several INT implementations using different target devices and INT header encapsulation. Among them, it is worth to mention [38], the first implementation including the dynamic control of the monitored network using the ONOS controller. Specifically, such work proposed extensions to ONOS for supporting INT that later have been included in the ONOS official distribution. More recently, the work in [39], focused on 5G networks, proposes the extension of the INT-MD technique up to the user equipment to enable the evaluation of fully end-to-end (e2e) latency. In general, INT telemetry features the generation of a `report` packet for each data plane packet. However, this approach may uselessly overload the telemetry system. Thus, the work in [40] and [41] proposes two different approaches for reducing the amount of generated telemetry data. In [40], an event detection framework is used to generate `report` packets only when certain events are detected in the network. In [41] a flexible sampling mechanism is implemented so that only a configurable fraction of data plane packets actually generates a `report` packet.

The use of an SDN controller over a P4-based network (e.g., exploiting the P4Runtime interface) allows to take fully advantage of P4 capabilities. Several solutions are currently available to deploy a P4Runtime interface in a physical P4 device. The *Bmv2* software switch [42] is a tool for emulating P4-based devices that implements also the P4Runtime interface. It is typically used for developing, testing and debugging the P4 data plane and the related network applications to be used at the SDN controller. However, many commercially available P4 devices do not provide a P4Runtime interface. To deal with this issue the Stratum [43], [44] and PINS (P4 Integrated Network Stack) [45] open-source projects are currently on-going to provide a P4Runtime interface deployable on P4-based devices equipped with a Network Operating System (NOS). In particular, Stratum can run on top of a Debian-based NOS (and on top of the Bmv2 software switch), whereas PINS can run on top of the SONiC (i.e., Software for Open Networking in the Cloud [46]) NOS. Few works in literature use P4Runtime to control P4-based devices, most of them leverage on Bmv2 and ONOS SDN controller. The work in [47] demonstrates the compatibility of P4Runtime and Openflow devices operating in the same network under the control of the single SDN controller, guaranteeing performance isolation among multiple network slices. The work in [48] implements a benchmarking tool for P4Runtime-based controllers and applies the tool to evaluate the performance of the ONOS controller running in both OpenFlow and P4Runtime mode. The work in [49] proposes the extension of P4Runtime to support multi-tenant service on a switch and evaluates the solution in an international experimental P4 network. Finally, within the ONF community, the *SD-Fabric* [50] project is a full stack application, i.e., involving both data and control planes, that implements a P4 pipeline for the Industry 4.0. The proposed P4 pipeline supports basic L2/L3 forwarding capabilities, 4G/5G mobile user plane, and in-band telemetry, whereas at the control side an ONOS application is provided for managing and controlling the P4 devices using P4Runtime.

Regarding the integration of Kubernetes with advanced networking, to the best of our knowledge we did not find solutions exploiting P4. Some research studies propose OpenFlow-based SDN solutions integrated with Kubernetes, providing basic networking features. In [15] a framework is proposed to create network slices on-demand among containers, connected by an OpenFlow-based software switch (i.e., Open vSwitch, OVS) in the host machine, being programmed by the SDN controller. The work in [16] proposes to contemporaneously deploy two CNI plugins, i.e., Calico to maintain the connectivity between pods and the Kubernetes master, and Multus that defines additional interfaces for attaching each pod to the network avoiding tunneling techniques, i.e., it maps each pod on a specific IP/MAC addresses pair so that the SDN controller can locate each pod. In [17] a CNI plugin is developed to expose a virtual network to pods and configure network tunnels among them using the SDN controller. Finally, the work in [51] proposes a tool (i.e., Host-INT) enabling end-to-end monitoring of traffic flows within a Kubernetes cluster. Host-INT leverages on extended Berkeley Packet Filter (eBPF) [52] to extend the Linux network stack of host machines introducing the support of the INT header, that is then used to collect information related to the traffic flows (e.g., packet loss and latency). The work in [53] proposes the implementation of a load balancer for P4-based Network Interface Controllers (NICs) towards services deployed with Kubernetes; however, it does not consider the fact that pod-to-pod traffic may be encapsulated in a tunnel.

The aforementioned work confirms that end-to-end monitoring is required in Kubernetes clusters, especially if host machines are deployed in different locations in a fog environment. However, using OpenFlow is impossible to match on packet fields encapsulated within a tunnel, thus all the proposed solutions require a deep integration with the Kubernetes cluster at the data plane level (e.g., installation of dedicated software in the host machines). Conversely, our proposal considers the utilization of P4 enabling the matching of pod-to-pod traffic throughout the traversed network without modifying the Kubernetes deployment.

III. BRAINE ARCHITECTURE

This work has been conducted in the context of the Big data pRocessing and Artificial Intelligence at the Network Edge (BRAINE) project. Thus, this section provides an overview of the BRAINE architecture to better contextualize the proposed integration between Kubernetes and the P4-based programmable network.

The BRANE project targets the development of an energy efficient Edge Micro Data Center (EMDC) exploiting a modular architecture (e.g., including heterogeneous hardware such as Central Processing Units(CPU), Graphics Processing UnitS(GPU),and Field-Programmable Gate Arrays(FPGA) to offer computing, acceleration, storage, and 5G Network Function Virtualization(VNF)) at the network edge. The project involves many industrial partners and works in several fields, including design and fabrication of hardware boards and development of the software framework to be deployed on top of the EMDC for cluster resource orchestration.

Within the scope of this paper, the BRAINE EMDC node includes a set of CPU boards, providing the cluster computational resources (with one of these boards dedicated to the hosting of orchestration tools), and dedicated boards exploiting the Spectrum chipset, made by Mellanox/Nvidia, implementing the SDN programmable P4-based switches to provide the connectivity among CPU boards. Besides a representation of the EMDC physical infrastructure, Fig. 1 reports the main components of the software framework as currently designed by BRAINE, where Kubernetes and ONOS have been respectively selected to orchestrate the computational resources and to control the programmable switches aiming to connect pods deployed on different CPU boards.

The following sections detail the BRAINE components that have been integrated in this work to implement a closed-loop automation where pods are deployed on different CPU boards of the same EMDC node or even on different EMDC nodes inter-connected by an SDN-enabled network devices.

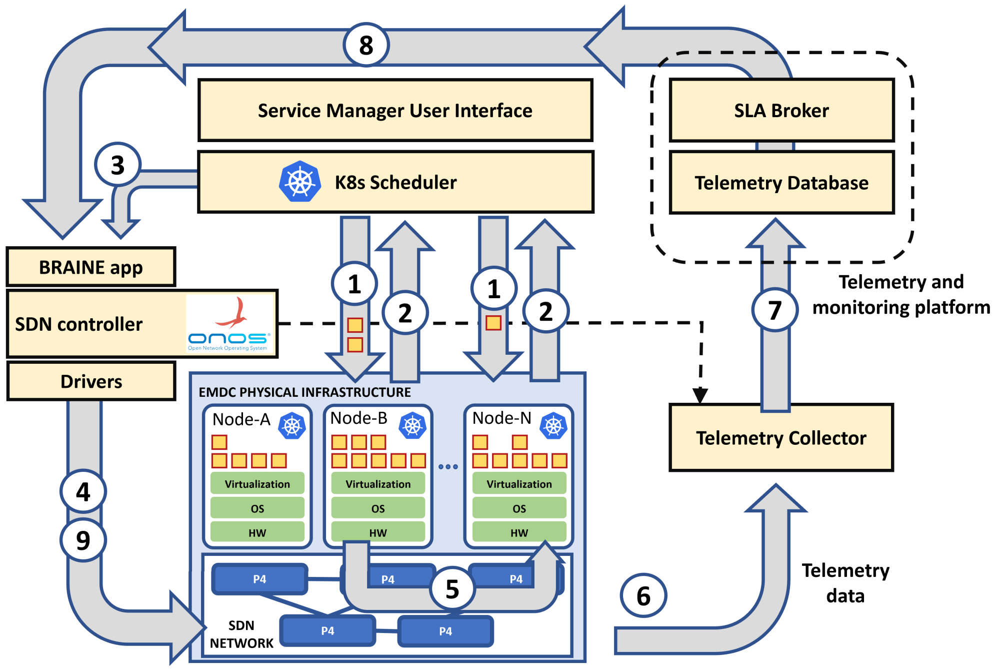


FIGURE 1. BRAINE EMDC main components and closed-loop telemetry workflow.

A. SERVICE MANAGER USER INTERFACE

The Service Manager User Interface (SMUI) is the north-bound interface of the EMDC node and provides a set of features to the users such as the unified view of services execution state through semantic knowledge graphs.

The SMUI is a containerized web-based application that can be deployed in the cloud or run locally on the EMDC [54]. It is built upon Kubernetes and Docker concepts such as images, containers, pods, worker nodes, services, workflows and their metadata. For instance, SMUI allows the user to identify the best node for running, training, or testing an Artificial intelligence (AI),service, with the possibility to choose the desirable execution architecture (e.g., CPU or GPU). Moreover, it enables the evaluation of resources availability across the system. With the collected running metadata, it is possible to check the presence of failures, the data accesses and further execution metadata.

More in detail, in the BRAINE data model, pods, services, and workflows are defined in a declarative way through Kubernetes definition language, using manifest files in `YAML` format, with the addition of the workflow data-type. The SMUI front-end enables service and workflow measurement/monitoring by accessing global and individual views on the multiple agents involved in the execution, while promptly taking corrective actions in case of failures such as redeploying the service to another node.

B. KUBERNETES SCHEDULER

The workloads defined using the SMUI are submitted to the EMDC by means of Kubernetes pods. A pod is the smallest execution unit in Kubernetes. Pods can contain one or more

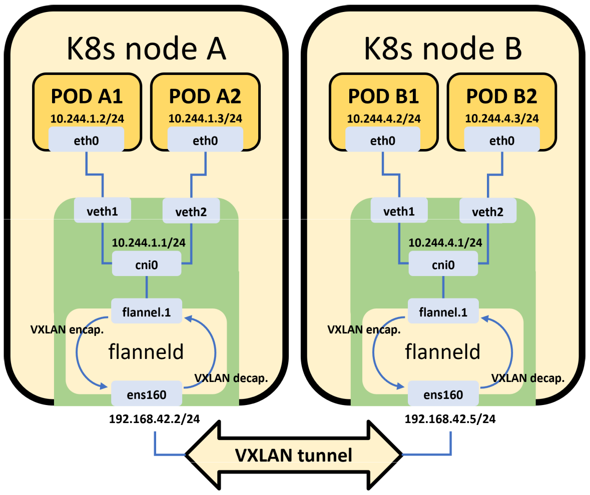


FIGURE 2. Inter-pod networking based on Flannel CNI using the VXLAN tunneling.

containers to run on a target worker node(s), all of the containers in a pod share the same IP address. Each service is a composition of pods that can claim different life-cycles as well as resources. For example, a service is provided by a number of pods identified by a cluster-wide Domain Name System (DNS), name, while the actual pods that compose the service may change during the execution, the clients of the service will still refer to the same endpoint.

Kubernetes is in charge of managing the pods. Upon admission of a pod, it runs mutation hooks, providing opportunities to validate, complete, and/or manipulate the pod according to the cluster's policies, e.g., replacing all container

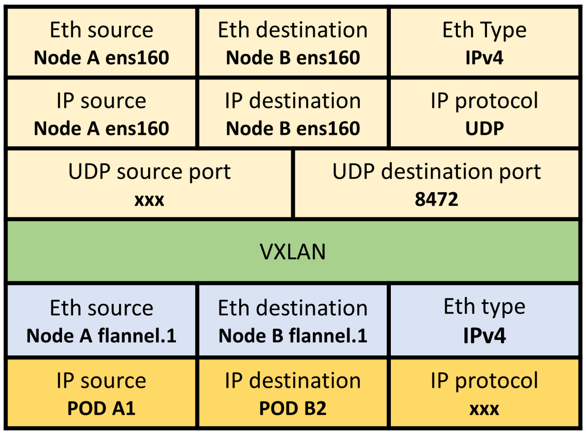


FIGURE 3. Protocol stack packets traveling from pod A1 to pod B2 (only meaningful fields are reported).

images with their latest version counterpart, or checking if they are hosted on a trusted image repository. Afterward, Kubernetes sends the pods into the scheduling pipeline, which consists of sorting, filtering, and scheduling. All these steps are plugin-based and can be extended or customized.

At the sorting stage, Kubernetes provides options to prioritize the pods. At the filtering step, Kubernetes checks whether the pod requirements could be fulfilled by the available worker nodes, and if so, it lists the matching nodes as feasible nodes. During the scheduling phase, Kubernetes runs the pod through a set of plugins asking them to score the feasible nodes. The node with the highest score will be nominated for binding, that is the process of shipping the pod to the selected worker node and asking it to accept and run the workload. This is organized via communication between Kubernetes and an agent (i.e., Kubelet) running on every worker node that updates and reports the pods status after each event (e.g., pod admission, termination, resource change). The status updates are received, aggregated, and collected by Kubernetes and maintained in a distributed key/value database called `etcd`. This information is used during filtering and scheduling steps, and by every other plugins requiring information about deployed pods.

C. FLANNEL-BASED KUBERNETES NETWORKING

In BRAINE, Kubernetes works with the Flannel CNI plugin running in VXLAN mode. Within the cluster, Flannel essentially solves two problems: duplication of pod IP addresses and inter-node pod networking (i.e., inter-board pod networking inside a single EMDC). The VXLAN method is the most used, due to the low introduced latency (i.e., traffic encapsulation and forwarding operations are performed in the Linux kernel) and because, exploiting tunneling based on IP reachability, it can be used to assure communication also among worker nodes interconnected by a routed IP network (e.g., among two separate EMDC nodes).

With reference to Fig. 2, Flannel creates a VXLAN network card named `flannel.1` on each node that acts as VXLAN Tunnel End Point (VTEP). Such interface is attached to the bridge `cni0` that works as IP gateway for all the pods in the node. The daemon *flanneld* configures the UDP port 8472 as default for VXLAN on the `flannel.1` interface. When a new node joins the cluster, flanneld exploits the information stored in the `etcd` to: i) create a routing entry in the local node to route the traffic addressed to pods running in the new detected node toward `flannel.1` interface; ii) add the IP of the new node to the ARP cache mapping it on the MAC address of `flannel.1` interface of the detected node.

Thus, for instance, once an IP packet is generated in pod A_1 (10.244.1.2), located at node A, and is destined to pod B_2 (10.244.4.3), located at node B, the packet is sent to `cni0` in node A (i.e., 10.244.1.1) through the routing table of pod A_1. Then at `cni0` the packet is sent to the `flannel.1` interface. As a VTEP device, `flannel.1` receives the message, according to the VTEP configuration the *flanneld* deamon knows that the destination pod 10.244.4.3 belongs to node B and it knows the IP address of `flannel.1` interface on node B from `etcd`, moreover through the forwarding table in node A, it knows the MAC of the VTEP of node B. Thus, it performs VXLAN packet encapsulation according to the configured parameters (e.g., local IP, port) and sends the packet through the physical interface `ens160`. The resulting protocol stack for packets traveling from pod A_1 to pod B_2 when exiting the interface `ens160` is illustrated in Fig. 3.

At node B, the VXLAN packet reaches the interface `ens160` via port 8472, the VXLAN packet is forwarded to the VTEP device `flannel.1` for decapsulation. The unpacked IP packet matches the routing table (10.244.4.0) in node B, and is therefore forwarded to `cni0` that, in turn, forwards it to pod B_2.

D. SDN CONTROLLER

The BRAINE SDN network controller is based on ONOS [19]. Fig. 4 represents the components specifically developed for BRAINE and utilized in this work to implement traffic forwarding and in-band telemetry, i.e., the BRAINE app and the BRAINE P4 app.

ONOS implements the concept of intent-based networking [55], where *intents* generalize the concept of connectivity ensuring that target policies are met by enabling automatic reconfiguration as a consequence of network changes (i.e., reacting to network events following a Finite State Machine (FSM)-based implementation). Since the intent is expected to be the base connectivity request submitted to the SDN controller, we developed our SDN applications on top of the ONOS intent service. This way all the implemented solutions are automatically managed by the intent FSM. For instance, in case of network failures all affected traffic flows are automatically moved to an alternate path.

1) THE BRAINE APP

This application implements a set of functionalities exposed through REST APIs, enabling the interaction with

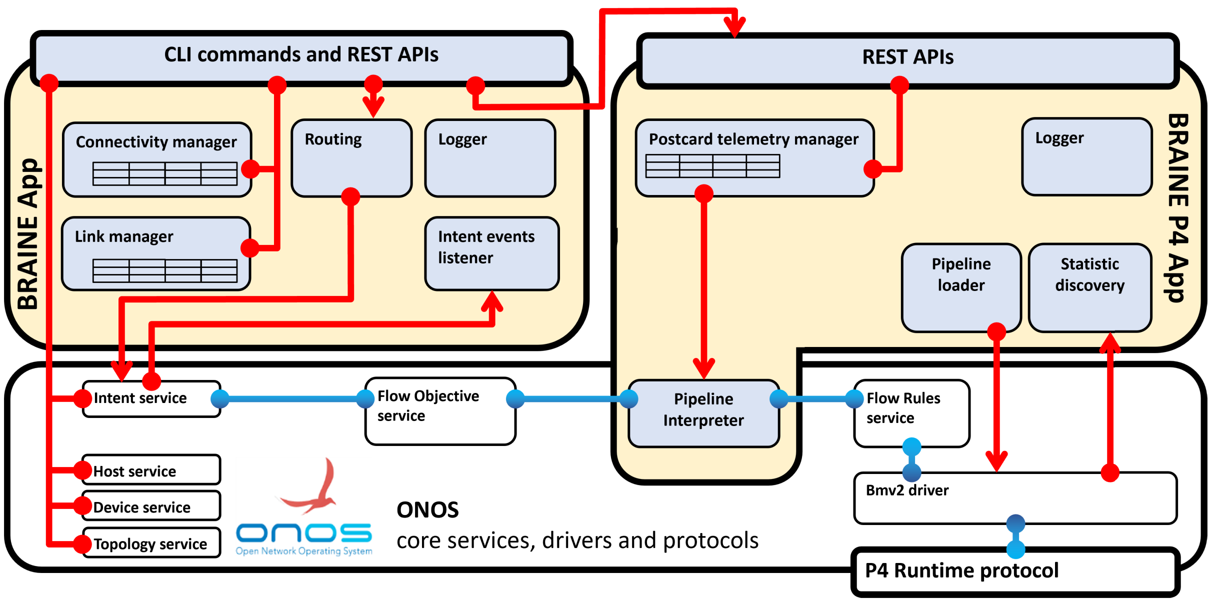


FIGURE 4. **Internal architecture of the ONOS apps developed for the BRAINE project, including relations with ONOS core services, drivers and protocols. Red connectors represent relations implemented within this work, blue connectors represent relations already present in the ONOS core.**

Kubernetes, and the SMUI. Also, the same functionalities can be manually accessed through a set of CLI commands. Moreover, the application utilizes the ONOS core services to enable the deployment of point-to-point connections between pods running in different worker nodes of the cluster. The two main functionalities supported at the data plane by the BRAINE app are: i) connection management (i.e., add/delete/modify), where each created connection can be specified up to the transport level (i.e., TCP/UDP ports); ii) activation of telemetry on selected active connection(s).

To support the aforementioned features, the BRAINE app is composed of several components (see left side of Fig. 4). In particular the application includes: i) two databases where connection and link state information is stored; ii) a routing module that performs redundant routing of requested connections and interacts with the ONOS intent service; iii) an intent listener that allows the application to react in case of network events affecting established connections; iv) a logger for tracing and debugging. Moreover, the BRAINE app supports a set of accessories features to facilitate the interaction with the network and the gathering of network state information. Specifically, the features supported by the app can be grouped in four categories: connections related commands, device related commands, host related commands and link related commands.

2) THE BRAINE P4 APP

The companion BRAINE P4 application has been developed to program the specific P4 pipeline to be used in the data plane switches. This application has two main roles: i) enabling the match of header field encapsulated within VXLAN tunnels; ii) activating the postcard telemetry on specific traffic flows.

The first objective is achieved through the implementation of a dedicated pipeline (described in Sec. V). For the latter objective, the application exposes a REST API that is dynamically consumed by the BRAINE app when a telemetry activation request is received from the orchestrator.

The internal architecture of the BRAINE P4 application is represented on the right side of Fig. 4. It includes the *pipeline loader* component which loads the P4 pipeline description via the P4Runtime protocol upon the discovery of P4-based switches. Once the request to activate a new postcard telemetry on a specific traffic is received through the REST interface, the *Postcard telemetry manager* identifies the devices traversed by the flow and sends them the flow rules to enable the postcard via the *pipeline interpreter*. Since the pipeline interpreter is the only component that is aware of the pipeline structure (e.g., number of tables and supported matching fields per table) it is also used for translating into flow rules the output of the intent service created to forward traffic. The *statistic discovery* component collects traffic related information from the P4-based devices to be visualized in the ONOS GUI (e.g., counters associated to flow rules). Finally, the *logger* component facilitates tracing and debug.

Both applications then rely on the Bmv2 P4 driver included in the master ONOS master distribution that has been demonstrated to be fully functional to perform the connection to P4 devices and to install all the required flow rules using the P4 Runtime protocol.

E. TELEMETRY COLLECTOR

The telemetry collector is the module in charge of receiving the telemetry `report` packets generated by the P4 switches. Typically, `report` packets are generated in a 1:1 ratio, with

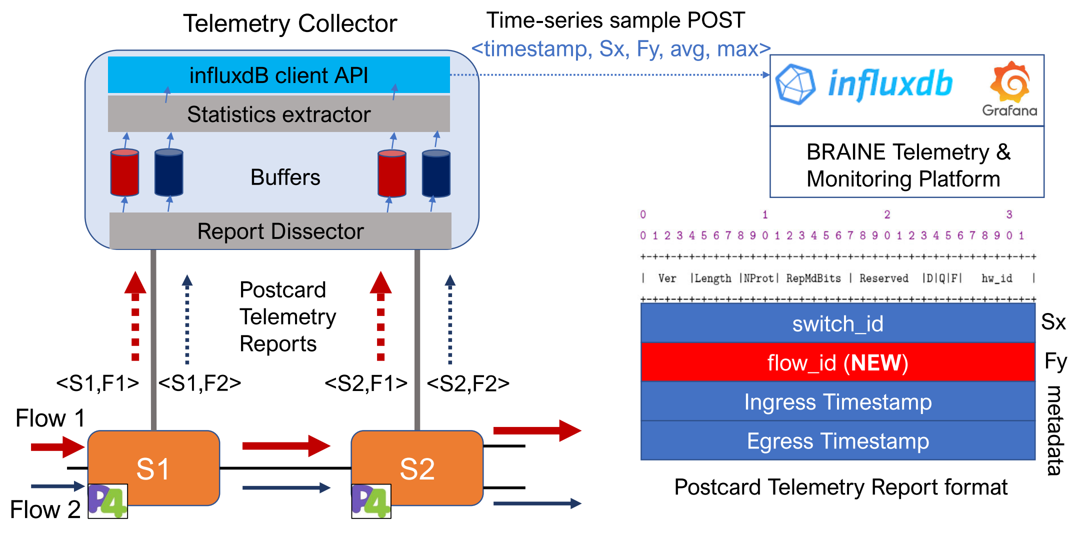


FIGURE 5. **Telemetry Collector module: operation, internal architecture and Report packet format.**

respect to the traffic packets belonging to telemetry-enabled flows, where each `report` provides metadata information (e.g., the latency experienced in the switch) related to a specific traffic packet. However, when the traffic rate increases, it is not possible for scalability reasons to populate the telemetry databases through direct processing of the report packets. Therefore, the telemetry collector module has been introduced for sampling the reports and providing low-rate telemetry statistics to the telemetry database. Such module classifies and aggregates the telemetry information at rates sustainable by the database layer, specific rates can be configured for different traffic flows through direct interaction with the SDN controller thus considering specific requirements of each flow (see Fig. 1). The aggregated per-flow and per-switch data samples are then sent to the telemetry time series database for monitoring purposes. Details regarding the implementation of a P4-based telemetry collector enabling full collector functionality at wire speed can be found in our previous work [56];

Fig. 5 shows the telemetry collector operation and its internal architecture. The different monitored traffic flows (i.e., F1 and F2 in the figure) generate the related telemetry `report` packets. The figure also details the `report` packet format showing the key fields utilized by the collector. In particular, the `switch_id` (*Sx*) field identifies the physical P4 switch *x* generating the `report`, while the `flow_id` (*Fy*) field discriminates the traffic flow *y*. The former field is defined in the P4 INT specifications [14], while the latter has been proposed as protocol extension in our previous work [57], in which the SDN controller computes and assigns the `flow_id` univocally and provides it along with the flow rules in the telemetry activation flow entry.

In this work, the considered metadata information retrieved by the programmable P4 switch is the intra-switch packet latency (i.e., the time spent by a packet in the switch queue). Each switch generates one `report` packets for each monitored flow packet, thus the `report` rate is equal to the flow rate. The collector receives and processes the `report` packets using specific internal modules. The dissector module extracts and stores the metadata of each `report` within different memory buffer arrays. The statistics extractor performs the aggregation of metadata samples in a pre-defined time window, computing the average (`avg`) and the maximum (`max`) latency experienced by packets belonging to flow *y* when crossing switch *x*. Specifically, the telemetry collector deployed in the experiments performs average and max latency value computation over the last 1000 report packets on a per-flow and per-switch basis.

The result is passed to the InfluxDB client API, that computes the overall statistics timestamp, and sends a REST POST message to the central InfluxDB database with the new time series sample. This way, specific latency analysis are possible at the BRAINE Telemetry and Monitoring Platform for each monitored flow at different switches.

F. TELEMETRY AND MONITORING PLATFORM

The telemetry and monitoring platform is a containerized application composed of the following open-source tools: InfluxDB [58], Prometheus [59], Node Exporter [60], and Grafana [61].

Specifically, the application collects from the worker nodes a set of metrics (e.g., CPU and RAM utilization) via Node Exporter and other custom-built exporters (e.g., the Telemetry Collector acts as a network resources exporter). Collected metrics are then harvested by Prometheus, which in turn, pushes them to the InfluxDB for storage. InfluxDB is an open-source time series database management system for the storage of metrics during the retention period. Moreover, it makes the collected metrics available for querying by other components of the system. InfluxDB can be queried via

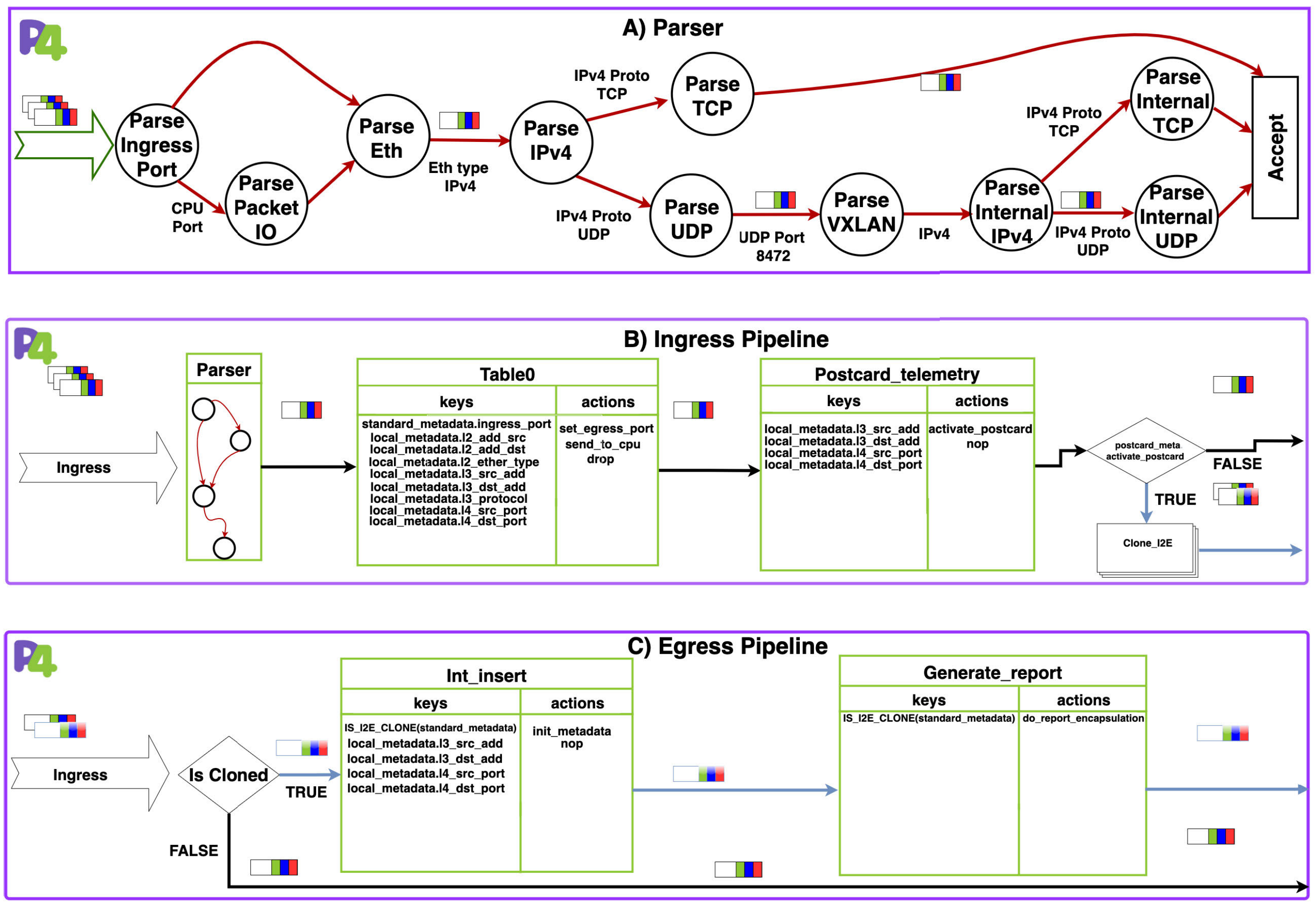

A) Parser
Parse Ingress Port
CPU Port
Parse Packet IO
Parse Eth
Eth type IPv4
Parse IPv4
IPv4 Proto TCP
Parse TCP
IPv4 Proto UDP
Parse UDP
UDP Port 8472
Parse VXLAN
IPv4
Parse Internal IPv4
IPv4 Proto TCP
Parse Internal TCP
IPv4 Proto UDP
Parse Internal UDP
Accept
B) Ingress Pipeline
Ingress
Parser
Table0
keys
actions
standard_metadata.ingress_port
local_metadata.l2_add_src
local_metadata.l2_add_dst
local_metadata.l2_ether_type
local_metadata.l3_src_add
local_metadata.l3_dst_add
local_metadata.l3_protocol
local_metadata.l4_src_port
local_metadata.l4_dst_port
set_egress_port
send_to_cpu
drop
Postcard_telemetry
keys
actions
local_metadata.l3_src_add
local_metadata.l3_dst_add
local_metadata.l4_src_port
local_metadata.l4_dst_port
activate_postcard
nop
postcard_meta activate_postcard
FALSE
TRUE
Clone_I2E
C) Egress Pipeline
Ingress
Is Cloned
TRUE
FALSE
Int_insert
keys
actions
IS_I2E_CLONE(standard_metadata)
local_metadata.l3_src_add
local_metadata.l3_dst_add
local_metadata.l4_src_port
local_metadata.l4_dst_port
init_metadata
nop
Generate_report
keys
actions
IS_I2E_CLONE(standard_metadata)
do_report_encapsulation

FIGURE 6. Proposed pipeline architecture for traffic forwarding and telemetry: a) Parser; b) Ingress pipeline; c) Egress pipeline.

external tools or its own REST APIs. Grafana is used as the default client of the database to provide a visual and interactive representation of the metric time series. However, there are other clients in the system that can interact with InfluxDB to obtain metric data. For instance, in BRAINE an important role is played by the the SLA Broker, that monitors the incoming metrics and validates them against the agreed SLA terms, to take proper corrective action in case of violations, i.e., activating the responsible actuators (e.g., the SDN controller).

In the BRAINE project, the InfluxDB is utilized as the single-point-of-truth metric database that persists the recorded data via Kubernetes volumes and provides a service endpoint for interacting with other components of the infrastructure. For instance, as described above, the telemetry collector pushes data directly to InfluxDB via the built-in APIs.

IV. TELEMETRY WORKFLOW

This work integrates the aforementioned BRAINE components in a closed-loop telemetry workflow. Specifically, the idea is to establish a connectivity between a pair of pods deployed on different worker nodes belonging to the same Kubernetes cluster, thus passing through a network composed of P4-based switches. The traffic flow exchanged between the two pods is then monitored activating in-band telemetry. When a latency degradation is detected along the path, the SDN controller is notified to find an alternative path, e.g., avoiding the switch that is introducing excessive delay.

The steps of the implemented telemetry workflow are represented in Fig. 1. Step 1: upon the trigger from the SMUI, Kubernetes places a number of pods with their own requirements on different worker nodes. Step 2: Kubernetes retrieves the network parameters of the deployed pods within the `etcd`. Step 3: Kubernetes submits a connectivity request to the SDN controller using the REST APIs provided by the ONOS BRAINE app including the network parameters of the deployed pods (i.e., the request typically contains MAC and IP addresses and TCP/UDP ports). Step 4: The SDN controller performs the configuration of the connectivity, sending the required flow rules to the involved P4-based devices (using P4-Runtime protocol), in the same step the SDN controller activates the postcard telemetry for the specific traffic flow, relying on the BRAINE P4 app that is dynamically queried by the BRAINE app (the telemetry could be also started/stopped in a subsequent step).

Step 5: Once the connectivity is configured, the traffic starts to flow into the network. Step 6: The related postcard telemetry is generated toward the Telemetry Collector. Step 7: When the Telemetry and Monitoring Platform detects a service level degradation (e.g., increased latency in a specific P4-based switch) it triggers a service upgrade request to the SDN controller using a dedicated method of the BRAINE app REST APIs. Step 8: The SDN controller modifies the network connectivity parameters in accordance with the received request (e.g., modify the traffic routing avoiding the degraded switch).

V. P4 PIPELINE IMPLEMENTATION

The developed P4 program is written in $P4_{16}$ for the target architecture `v1model` [62] that includes a parser and two pipelines (ingress and egress). With the proposed approach the P4 device can be programmed by the SDN controller to forward both traffic exchanged among pods (i.e., encapsulated using VXLAN) and traffic exchanged among host machines (i.e., not encapsulated). Moreover, the controller can activate in-band telemetry (i.e., postcard telemetry, INT-XD) on selected traffic flows, that can be specified up to transport layer details (i.e., TCP/UDP ports).

The proposed architecture is working only in conjunction with the Flannel plugin operating in the VXLAN mode. However, it is easily extensible to other tunneling techniques applied by different CNI plugins only requiring the upgrade of the parser module. For instance, the parser can be extended to support Calico operating with the *IP in IP* overlay networking by adding a specific check during the parsing of the IP header, i.e., to recognize the IP protocol code 0×04. Thus, a parser supporting multiple tunneling techniques could be deployed on the same network infrastructure.

Each pipeline is composed by a number of tables, operating with a match/action policy. Each table supports a specific set of keys and actions. In each table, a ternary match policy is used where the selected mask allows to ignore a key (i.e.,0×0000) or apply an exact match (i.e., `0xffff`).) All keys are defined using custom metadata (i.e., `local_metadata.*`) that are initialized loading the proper packet header fields during the parsing procedure. This way, depending on the detected tunneling technique, different packet fields can be copied in the metadata enabling the support of multiple CNI plugins. For the traffic not exploiting a tunnel, e.g., traffic among host machines or pod traffic generated in a cluster adopting flat networking (e.g., using the Calico default behaviour), metadata are filled considering the most external packet header.

A. P4-BASED MATCHING OF POD-TO-POD TRAFFIC

The parser, detailed in Fig. 6(a), is the first module of the ingress pipeline, as shown in Fig. 6(b). While the packet passes through the parser stages, the metadata fields are gradually filled. The first stage of the parser writes the ingress port index into the specific metadata field. Then, the *Parse Packet IO* stage is executed only for packets received from the CPU port (i.e., P4 Runtime `packet_out` messages received from the controller) to retrieve the `packet_out` header. The *Parse Eth* stage extracts the Ethernet header and fills the corresponding metadata fields with the MAC source, MAC destination and Ethernet type fields values. Then, in case of IP packets, the *Parse IPv4* stage parses the IPv4 header and fills the corresponding metadata fields with the IP source, IP destination and IP protocol fields value. Subsequently, the packet is sent to one of the *Parse TCP/UDP* stages where the metadata fields `local_metadata.l4_src_port` and `local_metadata.l4_dst_port` are filled.

If the UDP destination port is 8472, it means that the packet belongs to a pod-to-pod traffic flow encapsulated within a VXLAN tunnel by Flannel (see Fig. 3). In this case, the *Parse VXLAN* stage is executed parsing VXLAN header, subsequently IP and TCP/UDP headers are parsed by *Parse Internal* stages. During these stages, the aforementioned `local_metadata.*` fields are overwritten with the corresponding fields enclosed in the internal headers. This way, if the packet is encapsulated in a VXLAN tunnel, the ingress pipeline will match the internal header fields, thus enabling pod-to-pod traffic forwarding.

As illustrated in Fig. 6(b), after parsing, the packets are forwarded to the ingress pipeline and processed by `table0` where the egress port is assigned based on the flow rules installed by the SDN controller. The actions supported in this table are: i) `set_egress_port`, ii) `send_to_cpu` and iii) `drop`. The `set_egress_port` action is typically applied to packets matching a forwarding flow rule and assigns the output port on which the packet will be transmitted. Action `send_to_cpu` is used to forward packets to the SDN controller through the control plane port (e.g., it is used for LLDP packets matching specific flow rules); finally, the default `drop` action is applied to packets not matching any flow rules.

B. P4-BASED POSTCARD TELEMETRY IMPLEMENTATION

The subsequent tables in both the ingress and the egress pipelines are used to implement the postcard telemetry. The *Postcard_Telemetry* table, see Fig. 6(b), matches on metadata fields and is intended to contain flow rules for matching each traffic flow requiring postcard telemetry. Two actions are supported: `activate_postcard` and `nop`. The action `activate_postcard` is executed for each matching packet (i.e., to packets belonging to traffic flows for which the SDN controller has activated the telemetry), setting a specific metadata field (i.e., `postcard_meta_activate_postcard`) that is later evaluated by an *if* condition to clone the packet using the `cloneI2E` external feature. If a packet is not matched, the default action `nop` is executed resulting in the packet forwarded to the egress pipeline without cloning. The cloned packet will be manipulated in the egress pipeline to generate a report packet. Cloning the packet is mandatory because P4 devices cannot create packets from scratch [10].

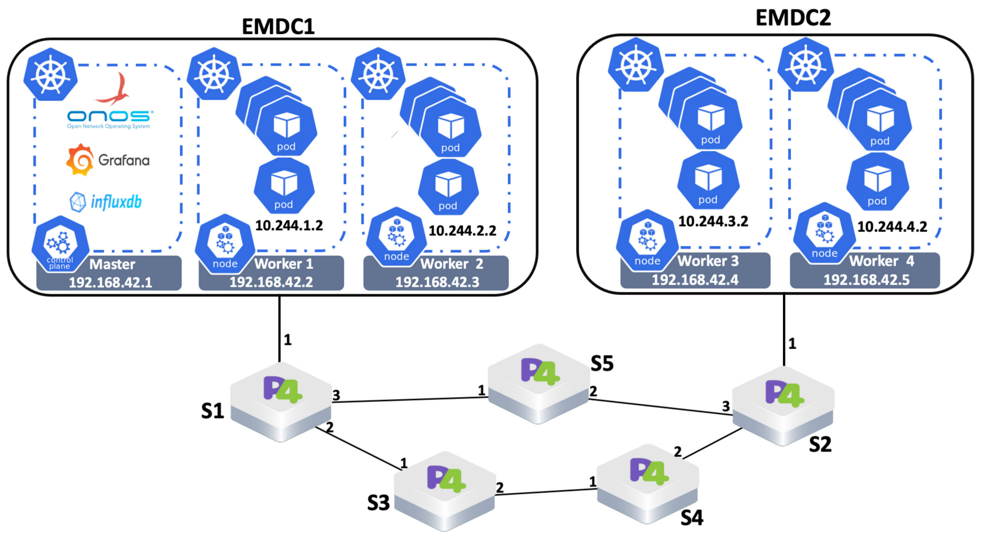
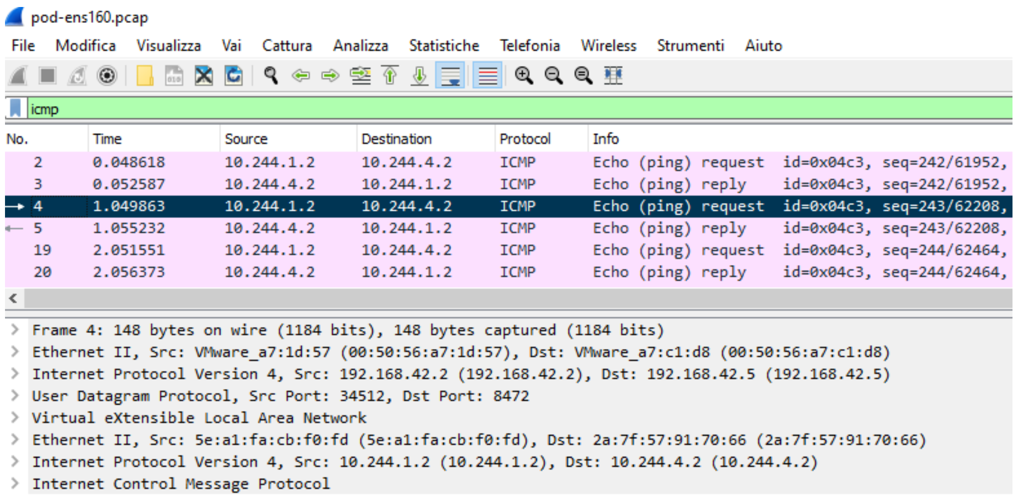


FIGURE 7. Experimental testbed encompassing computational and networking resources.


pod-ens160.pcap
File Modifica Visualizza Vai Cattura Analizza Statistiche Telefonia Wireless Strumenti Aiuto
icmp

```
No.   Time       Source       Destination  Protocol  Info
  2   0.048618   10.244.1.2   10.244.4.2   ICMP      Echo (ping) request  id=0x04c3, seq=242/61952,
  3   0.052587   10.244.4.2   10.244.1.2   ICMP      Echo (ping) reply    id=0x04c3, seq=242/61952,
→ 4   1.049863   10.244.1.2   10.244.4.2   ICMP      Echo (ping) request  id=0x04c3, seq=243/62208,
← 5   1.055232   10.244.4.2   10.244.1.2   ICMP      Echo (ping) reply    id=0x04c3, seq=243/62208,
 19   2.051551   10.244.1.2   10.244.4.2   ICMP      Echo (ping) request  id=0x04c3, seq=244/62464,
 20   2.056373   10.244.4.2   10.244.1.2   ICMP      Echo (ping) reply    id=0x04c3, seq=244/62464,

> Frame 4: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits)
> Ethernet II, Src: VMware_a7:1d:57 (00:50:56:a7:1d:57), Dst: VMware_a7:c1:d8 (00:50:56:a7:c1:d8)
> Internet Protocol Version 4, Src: 192.168.42.2 (192.168.42.2), Dst: 192.168.42.5 (192.168.42.5)
> User Datagram Protocol, Src Port: 34512, Dst Port: 8472
> Virtual eXtensible Local Area Network
> Ethernet II, Src: 5e:a1:fa:cb:f0:fd (5e:a1:fa:cb:f0:fd), Dst: 2a:7f:57:91:70:66 (2a:7f:57:91:70:66)
> Internet Protocol Version 4, Src: 10.244.1.2 (10.244.1.2), Dst: 10.244.4.2 (10.244.4.2)
> Internet Control Message Protocol
```

FIGURE 8. Wireshark capture of ICMP traffic between two pods.

The egress pipeline is illustrated in Fig. 6(c). All the metadata fields `local_metadata.*` must be re-initialized because P4 does not allow the propagation of custom metadata from the ingress pipeline to the egress pipeline. No actions are applied to the original packet that leaves the switch through the port assigned in *table0*. Instead, the cloned packet is processed by the two tables: `int_insert` and `generate_report`. The former table, with a null default action (i.e., `nop`), applies the action `init_metadata` to matching packets. This action is the one that actually retrieves the information to be included in the report message that is written in the `local_metadata.postcard_*` fields.

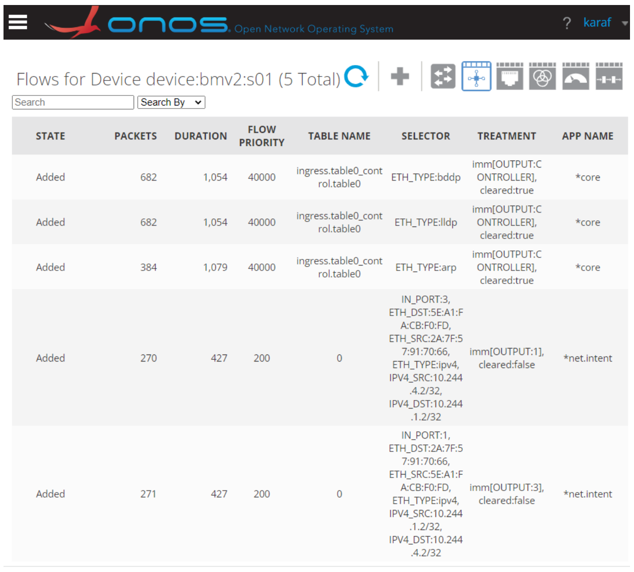
The latter table generates the in-band telemetry report message using the action `do_report_encapsulation` manipulating the cloned packet. More in detail, the header of the cloned packet is modified as following. The Ethernet and IP source addresses are set to the local switch values, while the destination addresses are set to the telemetry collector values. The UDP source and destination ports are set to a specific values to easily recognize report packets at the telemetry collector. Finally, the report header is added as UDP payload that includes the metadata retrieved in the previous table, i.e., `switch_id`, `flow_id` and all other metadata required by the SDN controller using the `instruction_mask` as defined in [14].

C. APPLICABILITY TO HARDWARE P4-BASED DEVICES

The proposed P4 pipeline is suitable for switches adopting the Protocol Independent Switch Architecture (PISA), e.g., the Tofino chipset is based on PISA. However, different

PISA-based switches can be characterized by different accessory features, e.g., *externs*. Specifically, the key P4 capabilities needed to run our proposed pipeline are extra header processing, timestamp metadata support, and the extern used to clone packets from the ingress to egress pipeline. All of them are available on most of the currently commercialized PISA-based switches.

In addition, an estimation of the maximum number of installable flow rules can be performed considering the typical size of the memory modules in commercial switches and the flow rule structure defined in Sec.V-A. Specifically, hardware switches are typically composed by the following modules: Static Random Access Memory (SRAM), Ternary Content Addressable Memory (TCAM), Hash, Arithmetic Logical Units (ALU) and stateful ALUs. The TCAM, used to implement ternary match, is the most expensive and flexible module and is therefore the scarcest resource. The P4 pipeline implemented in our work uses ternary match only for Table0, while other tables, applying exact match, can be mapped on the less critical SRAM. Thus, considering the fields to be matched (illustrated in Fig. 6(b)), each flow rule occupies about 300 bits. A Table0 with 1000 rules occupies 300 Kbits (i.e., about 40 KBytes). Since modern switches have a TCAM size in the order of 1 MByte, they can typically contain 25.000 flow-rules.



Flows for Device device:bmv2:s01 (5 Total)

Search | Search By

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	682	1,054	40000	ingress.table0_control.table0	ETH_TYPE:bddp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	682	1,054	40000	ingress.table0_control.table0	ETH_TYPE:lldp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	384	1,079	40000	ingress.table0_control.table0	ETH_TYPE:arp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	270	427	200	0	IN_PORT:3, ETH_DST:5E:A1:FA:CB:F0:FD, ETH_SRC:2A:7F:57:91:70:66, ETH_TYPE:ipv4, IPV4_SRC:10.244.4.2/32, IPV4_DST:10.244.1.2/32	imm[OUTPUT:1], cleared:false	*net.intent
Added	271	427	200	0	IN_PORT:1, ETH_DST:2A:7F:57:91:70:66, ETH_SRC:5E:A1:FA:CB:F0:FD, ETH_TYPE:ipv4, IPV4_SRC:10.244.1.2/32, IPV4_DST:10.244.4.2/32	imm[OUTPUT:3], cleared:false	*net.intent

FIGURE 9. **ONOS view of rules installed on switch *S*1.**

VI. EXPERIMENTS

A. EXPERIMENTAL SETUP

The experimental testbed encompasses both computing and networking resources. Computing resources are deployed on two dedicated servers, i.e., $EMDC_1$ and $EMDC_2$ in Fig. 7. The hardware of both servers is a DELL PowerEdge R740, 56 CPUs Intel Xeon Gold 6238R @ 2.20GHz, 256 GB RAM. Three virtual machines (VMs) are deployed in $EMDC_1$, while two VMs are deployed in $EMDC_2$. One of the VMs deployed on $EMDC_1$ hosts the management and control software including the Kubernetes master, the ONOS SDN controller, the telemetry collector and the telemetry and monitoring platform. The other VMs act as Kubernetes worker nodes, where each node runs a number of pods (i.e., each pod encompasses a plain Ubuntu 20.04 distribution with basic networking tools).The Telemetry and Monitoring platform includes the telemetry database deployed into an influxdB container, and the SLA Broker, implemented as a set of configurable queries and threshold-based alarms through dedicated Grafana panels.

Networking resources encompass five P4-based switches, all of them emulated using Bmv2. Since we are using emulated devices, measured latency values are expected to significantly reduce using hardware devices. However, this is not relevant for our purpose, since our work does not target to improve the achievable latency, it only targets to measure the latency and triggers a network reconfiguration when a certain threshold is crossed. Switches $S1$, $S2$, $S3$, $S4$ are emulated on a dedicated DELL server (Intel Xeon E5-2643 v3 6-core 3.40 GHz clock, 32 GB RAM) using physical Ethernet interfaces. Switch $S5$ is emulated by deploying a dockerized Bmv2 on a Mellanox/Nvidia SN2010, running SONiC. In particular, the Mellanox/Nvidia SN2010 is a switch exploiting the Spectrum chipset, where it is possible to install the SONiC NOS. In turn, SONiC enables the deployment of the dockerized Bmv2 switch within in the SN2010.

The traffic report generated by the network nodes is received by the Telemetry Collector, hosted by the Kubernetes master node. As depicted in Fig. 5 the report packet contains: the `switch_id` field that identifies the switch, the `flow_id` field that discriminates traffic flows, `Ingress_Timestamp` and `Egress_Timestamp` needed to evaluate the hop latency.

B. EXPERIMENTAL RESULTS

1) POD TRAFFIC FORWARDING VALIDATION

This section functionally validates the proposed solution to process the traffic exchanged between a pair of Kubernetes pods. Specifically, the traffic is generated between two pods respectively deployed on node $EMDC_1$ and $EMDC_2$, thus traversing the P4-based network.

Fig. 8 illustrates the Wireshark capture, including the VXLAN encapsulation and the protocol stacking as shown in Fig. 3. Specifically, the *ping* application is used to generate ICMP request/reply messages between pod 10.244.1.2 deployed on worker node 1 and pod 10.244.4.2 deployed on worker node 4. The packets are captured in VM Worker 1 on interface 192.168.42.2. The presence of both ICMP request and reply proves that packets are correctly switched by the network in both directions. The experienced round-trip time is around 5 milliseconds.

Fig. 9 shows a screenshot of the ONOS web GUI illustrating the flow rules installed in switch $S1$ where the rules counters show that the traffic exchanged between the two pods is correctly matched.

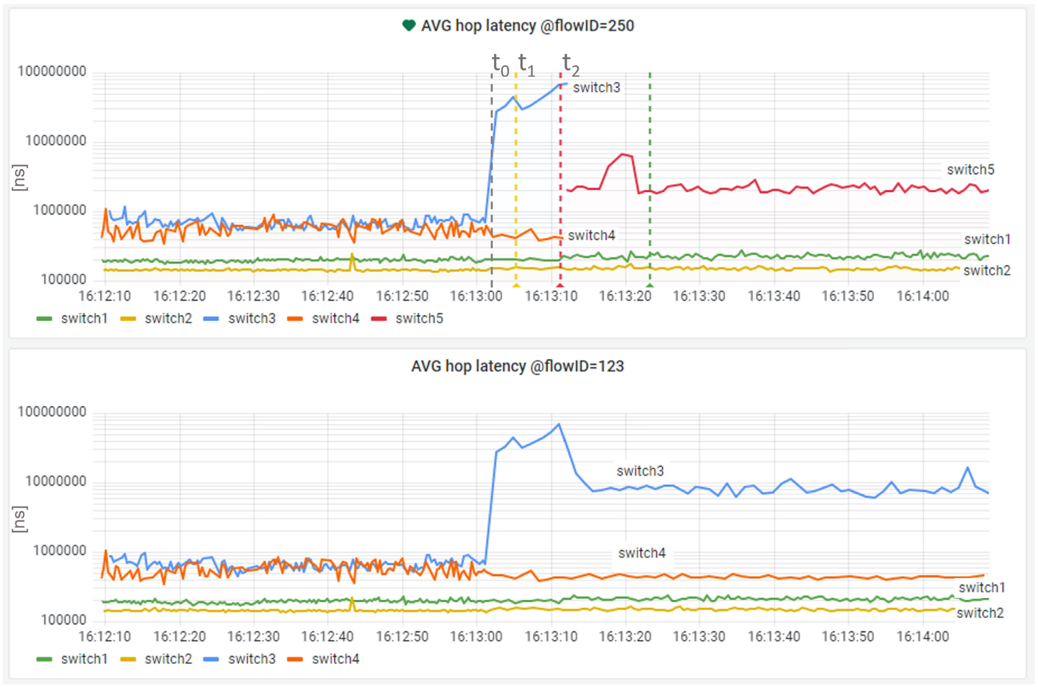


FIGURE 10. **Telemetry and Monitoring Platform: view of switch latency for traffic flows 250 and 123. Latency [ns] as a function of experiment time.**

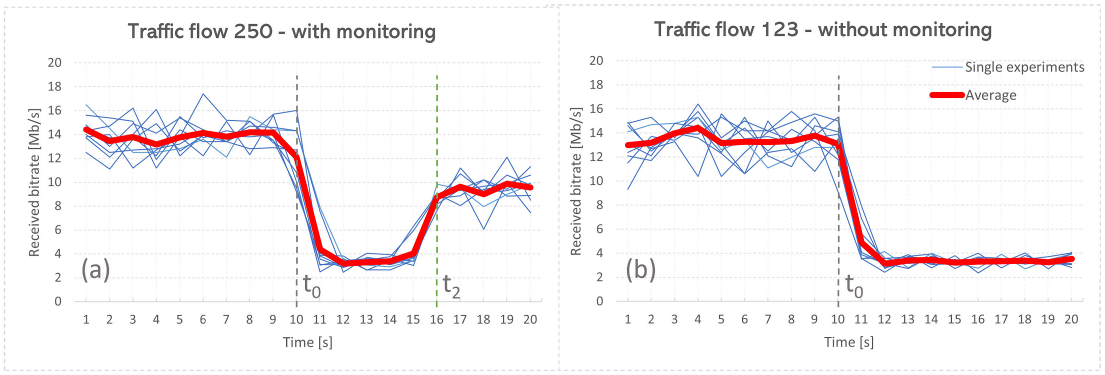


FIGURE 11. **Flow bitrate: (a) flow 250; (b) flow 123. Mbps as a function of experiment time.**

2) POD TRAFFIC TELEMETRY VALIDATION

This section functionally validates the whole telemetry workflow as described in Fig. 1. Specifically, two separate traffic flows are activated between two different pairs of pods: flow IDs 250 and 123. The two flows consist of five parallel TCP sessions generated with the *iperf3* application. Telemetry is active in both flows; however, the SLA Broker is configured to generate the feedback to ONOS (step 8 in Fig. 1) only for flow 250.

Fig. 10 reports the latency data as collected by the SLA Broker panels during the network reconfiguration. Both flows are initially routed along the path $S1$, $S3$, $S4$, $S2$, thus both plots report four latency lines, one per traversed switch. At time t_0 switch $S3$ transmission rate is manually degraded, thus increasing the switch latency for both flows. The SLA Broker performs a threshold-based control over the per switch latency of flow 250 and triggers an alert if the degradation persists for 4 seconds. This behaviour is reflected in the SLA Broker panel as depicted in Fig. 10.

In the actual experiment, degradation is detected at t_1 and the alert is triggered back to ONOS at t_2. As described in the previous sections, ONOS reacts by rerouting the affected flow (i.e., flow 250) on path $S1$, $S5$, $S2$, i.e., after t_2, Fig. 10 reports the latency of those switches. It is worth noting that $S5$ is characterized by a higher latency compared to other switches; indeed, $S5$ is emulated on less performance hardware. Conversely, flow 123 is not involved in the reconfiguration, showing that the implemented framework is able to select the single traffic flow.

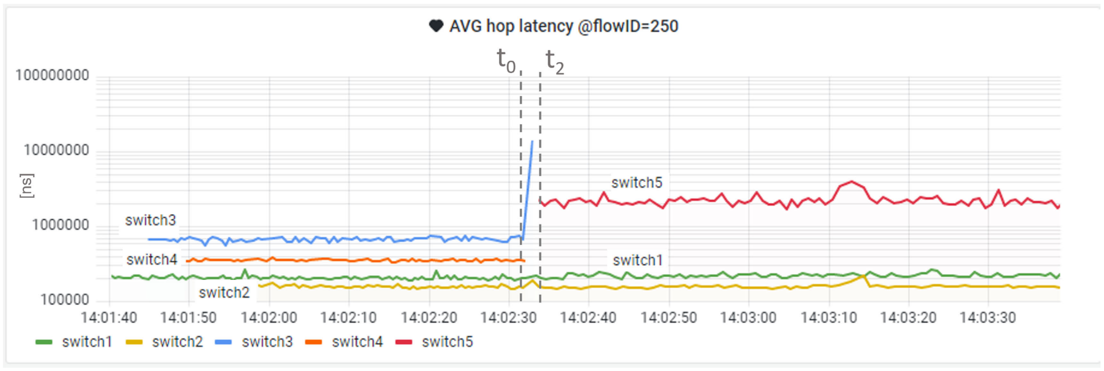


FIGURE 12. **Auxiliary panel view of switch latency for traffic flows 250 experiencing network failure recovery excluding the Telemetry and Monitoring Platform. Latency [ns] as a function of experiment time.**

The telemetry workflow experiment has been repeated 10 times collecting also the achieved end-to-end bit-rate of both flows. The results are illustrated in Fig. 11, including ten cyan lines reporting the specific result for each experiment and a single red line reporting the average trend. Specifically, Fig. 11(a) is related to traffic flow 250, it shows that after t_0 the rate is degraded, then it is partially recovered at time t_2 when the traffic is switched on the alternate path. It is worth noting that rerouting the traffic does not guarantee the recovery of the overall bit-rate. In fact, the recovery path includes switch $S5$ emulated on a less performing hardware with limited traffic capabilities. Fig. 11(b) is related to traffic flow 123 that is not involved in the reconfiguration, thus after t_0 the bit-rate results to be degraded and never recovered.

Fig. 11(a) shows that the whole workflow takes about 6 seconds to be performed (i.e., $t_2 - t_0$). However, most of this time is expended within the telemetry and monitoring platform (i.e., SLA Broker) as a result of our configuration to trigger the alert. This time could be reduced by configuring the SLA Broker with higher SLA checking rates on the InfluxDB filled by the Telemetry Collector. Therefore, to better evaluate the achievable performance of the system, we have measured the re-configuration time excluding the telemetry and monitoring platform from the workflow, i.e., the feedback to the ONOS controller is directly generated by the Telemetry Collector.

Fig. 12 reports the latency data collected by an auxiliary Grafana panel during the network reconfiguration, when the reconfiguration is triggered directly by the Telemetry Collector (i.e., thus excluding the influxdB and the SLA Broker). The experiment has been repeated 10 times and the average time for performing the reconfiguration is 1.95 seconds that includes: the detection of the latency degradation at the Telemetry Collector, all control plane procedures performed in ONOS (e.g., computation of an alternate path), and P4 Runtime message exchange towards the involved switches.

VII. CONCLUSION AND FUTURE DIRECTIONS

The integration between IT and networking technologies is fundamental for effective micro-services deployment on next generation edge nodes interconnected by a network infrastructure. However, currently available solutions considering Kubernetes orchestrated clusters and programmable networks are based on legacy SDN techniques and thus require deep integration at the data plane level.

In this work we proposed a P4-based solution able to gain visibility inside tunnelled traffic, and thus enabling such integration at the control plane level through communication between the Kubernetes orchestrator and the ONOS SDN controller. Also we experimentally demonstrated the first comprehensive framework enabling effective traffic telemetry, at pod level, building upon a closed-loop workflow among (i) the Kubernetes orchestrator, (ii) the ONOS SDN Controller, (iii) the enhanced P4-based data plane, and (iv) the telemetry system. The integrated system is able to orchestrate Kubernetes micro-service chains and automatic P4 switch configuration including configurable telemetry. Moreover, the closed-loop BRAINE telemetry and monitoring system is able to enforce automatic network recovery of specific flows violating latency SLA in less than 2 seconds.

The proposed framework paves the way toward even more advanced closed-loop strategies for the dynamic reconfiguration of flows (e.g., traffic prioritization, control of generated traffic rates at the application level, etc.) depending on the performance measured on the network.

ACKNOWLEDGMENT

(Davide Scano and Alessio Giorgetti contributed equally to this work.)

REFERENCES

[1] R. Morabito, I. Farris, A. Iera, and T. Taleb, "Evaluating performance of containerized IoT services for clustered devices at the network edge," *IEEE Internet Thing J.*, vol. 4, no. 4, pp. 1019–1030, Aug. 2017.

[2] D. Adami, B. Martini, A. Sgambelluri, L. Donatini, M. Gharbaoui, P. Castoldi, and S. Giordano, "An SDN orchestrator for cloud data center: System design and experimental evaluation," *Trans. Emerg. Telecommun. Technol.*, vol. 28, no. 11, p. e3172, Nov. 2017.

[3] R. Cziva and D. P. Pezaros, "Container network functions: Bringing NFV to the network edge," *IEEE Commun. Mag.*, vol. 55, no. 6, pp. 24–31, Jun. 2017.

[4] R. Muñoz, R. Vilalta, N. Yoshikane, R. Casellas, R. Martínez, T. Tsuritani, and I. Morita, "Integration of IoT, transport SDN, and edge/cloud computing for dynamic distribution of IoT analytics and efficient use of network resources," *J. Lightw. Technol.*, vol. 36, no. 7, pp. 1420–1428, Apr. 1, 2018.

[5] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, ‘‘Resource scheduling in edge computing: A survey,’’ *IEEE Commun. Surveys Tuts.*, vol. 23, no. 4, pp. 2131–2165, 4th Quart., 2021.

[6] P. Kayal, ‘‘Kubernetes in fog computing: Feasibility demonstration, limitations and improvement scope : Invited paper,’’ in *Proc. IEEE 6th World Forum Internet Things (WF-IoT)*, Jun. 2020, pp. 1–6.

[7] *Kubernetes Documentation*. Accessed: Feb. 27, 2023. [Online]. Available: https://kubernetes.io/docs/home/

[8] I. Pelle, F. Paolucci, B. Sonkoly, and F. Cugini, ‘‘Latency-sensitive edge/cloud serverless dynamic deployment over telemetry-based packet-optical network,’’ *IEEE J. Sel. Areas Commun.*, vol. 39, no. 9, pp. 2849–2863, Sep. 2021.

[9] A. S. Muqaddas, R. S. Tessinari, R. Casellas, M. Garrich, E. Hugues-Salas, Ó. G. D. Dios, L. Luque, A. Giorgetti, A. Sgambelluri, F. Cugini, F.-J. Moreno-Muro, R. Morro, K. Farrow, A. Wonfor, M. Channegowda, P. Pavón-Mariño, A. Lord, R. Nejabati, and D. Simeonidou, ‘‘NFV orchestration over disaggregated metro optical networks with end-to-end multi-layer slicing enabling crowdsourced live video streaming,’’ *J. Opt. Commun. Netw.*, vol. 13, no. 8, pp. D68–D79, 2021.

[10] The P4 Language Consortium. (May 2017). *P4 16 Language Specification*. Accessed: Jul. 26, 2021. [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html

[11] F. Paolucci, F. Cugini, P. Castoldi, and T. Osinski, ‘‘Enhancing 5G SDN/NFV edge with P4 data plane programmability,’’ *IEEE Netw.*, vol. 35, no. 3, pp. 154–160, May 2021.

[12] *OpenFlow Switch Specification Version 1.5.1*, Open Netw. Found., Menlo Park, CA, USA, Mar. 2015.

[13] E. F. Kfoury, J. Crichigno, and E. Bou-Harb, ‘‘An exhaustive survey on P4 programmable data plane switches: Taxonomy, applications, challenges, and future trends,’’ *IEEE Access*, vol. 9, pp. 87094–87155, 2021.

[14] P4.org Applications Working Group. *In-Band Network Telemetry (INT) Dataplane Specification Version 2.1*. Accessed: Feb. 27, 2023. [Online]. Available: https://p4.org/p4-spec/docs/INT_v2_1.pdf

[15] A. Rafiq, A. Mehmood, and W.-C. Song, ‘‘Intent-based slicing between containers in SDN overlay network,’’ *J. Commun.*, vol. 15, no. 3, pp. 237–244, 2020.

[16] R. Botez, J. Costa-Requena, I.-A. Ivanciu, V. Strautiu, and V. Dobrota, ‘‘SDN-based network slicing mechanism for a scalable 4G/5G core network: A kubernetes approach,’’ *Sensors*, vol. 21, no. 11, p. 3773, May 2021.

[17] R. Figueiredo and K. Subratie, ‘‘Demo: EdgeVPN.io: Open-source virtual private network for seamless edge computing with kubernetes,’’ in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Nov. 2020, pp. 190–192.

[18] P4.org Applications Working Group. *P4Runtime Specification Version 1.3.0*. Accessed: Feb. 27, 2023. [Online]. Available: https://p4.org/p4-spec/p4runtime/v1.3.0/P4Runtime-Spec.pdf

[19] *Open Network Operating System, ONOS Official Site*. Accessed: Feb. 27, 2023. [Online]. Available: https://opennetworking.org/onos/

[20] A. Giorgetti, D. Scano, J. Chamanara, M. Albado, E. Marx, S. Ahearne, A. Sgambelluri, F. Paolucci, and F. Cugini, ‘‘Kubernetes orchestration in SDN-based edge network infrastructure,’’ in *Tech. Dig. Opt. Fiber Commun. Conf. (OFC)*, 2022, pp. 1–3.

[21] G. Castellano, F. Esposito, and F. Risso, ‘‘A distributed orchestration algorithm for edge computing resources with guarantees,’’ in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Apr. 2019, pp. 2548–2556.

[22] E. Zeydan, J. Mangues-Bafalluy, and Y. Turk, ‘‘Intelligent service orchestration in edge cloud networks,’’ *IEEE Netw.*, vol. 35, no. 6, pp. 126–132, Nov. 2021.

[23] *Kubernetes CNI*. Accessed: Feb. 27, 2023. [Online]. Available: https://www.cni.dev/

[24] *Flannel*. Accessed: Feb. 27, 2023. [Online]. Available: https://github.com/flannel-io/flannel

[25] *Calico*. Accessed: Feb. 27, 2023. [Online]. Available: https://github.com/projectcalico/calico

[26] *Kuberouter*. Accessed: Feb. 27, 2023. [Online]. Available: https://github.com/cloudnativelabs/kube-router

[27] N. Kapocius, ‘‘Performance studies of kubernetes network solutions,’’ in *Proc. IEEE Open Conf. Electr., Electron. Inf. Sci. (eStream)*, Apr. 2020, pp. 1–6.

[28] P. Zanna, P. Radcliffe, and K. G. Chavez, ‘‘A method for comparing OpenFlow and P4,’’ in *Proc. 29th Int. Telecommun. Netw. Appl. Conf. (ITNAC)*, Nov. 2019, pp. 1–3.

[29] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, and P. Castoldi, ‘‘P4 edge node enabling stateful traffic engineering and cyber security,’’ *J. Opt. Commun. Netw.*, vol. 11, no. 1, pp. A84–A95, 2019.

[30] F. Cugini, D. Scano, A. Giorgetti, A. Sgambelluri, L. D. Marinis, P. Castoldi, and F. Paolucci, ‘‘Telemetry and AI-based security P4 applications for optical networks [invited],’’ *J. Opt. Commun. Netw.*, vol. 15, no. 1, pp. A1–A10, Jan. 2023.

[31] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. V. Ramos, and A. Madeira, ‘‘FlowLens: Enabling efficient flow classification for ML-based network security applications,’’ in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1–18.

[32] R. Kundel, F. Siegmund, J. Blendin, A. Rizk, and B. Koldehofe, ‘‘P4STA: High performance packet timestamping with programmable packet processors,’’ in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, Apr. 2020, pp. 1–9.

[33] D. Merling, S. Lindner, and M. Menth, ‘‘Hardware-based evaluation of scalable and resilient multicast with BIER in P4,’’ *IEEE Access*, vol. 9, pp. 34500–34514, 2021.

[34] I. Wijnands, E. Rosen, A. Dolganow, T. Przygienda, and S. Aldrin, *Multicast Using Bit Index Explicit Replication (BIER)*, document RFC 8279, 2018.

[35] J. Alvarez-Horcajo, I. Martínez-Yelmo, D. Lopez-Pajares, J. A. Carral, and M. Savi, ‘‘A hybrid SDN switch based on standard P4 code,’’ *IEEE Commun. Lett.*, vol. 25, no. 5, pp. 1482–1485, May 2021.

[36] Y. Yan, A. F. Beldachi, R. Nejabati, and D. Simeonidou, ‘‘P4-enabled smart NIC: Enabling sliceable and service-driven optical data centres,’’ *J. Lightw. Technol.*, vol. 38, no. 9, pp. 2688–2694, May 1, 2020.

[37] P. Manzanares-Lopez, J. P. Muñoz-Gea, and J. Malgosa-Sanahuja, ‘‘Passive in-band network telemetry systems: The potential of programmable data plane on network-wide telemetry,’’ *IEEE Access*, vol. 9, pp. 20391–20409, 2021.

[38] N. Van Tu, J. Hyun, and J. W.-K. Hong, ‘‘Towards ONOS-based SDN monitoring using in-band network telemetry,’’ in *Proc. 19th Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Sep. 2017, pp. 76–81.

[39] D. Scano, F. Paolucci, K. Kondepu, A. Sgambelluri, L. Valcarenghi, and F. Cugini, ‘‘Extending P4 in-band telemetry to user equipment for latency- and localization-aware autonomous networking with AI forecasting,’’ *J. Opt. Commun. Netw.*, vol. 13, no. 9, p. D103, 2021.

[40] J. Vestin, A. Kassler, D. Bhamare, K.-J. Grinnemo, J.-O. Andersson, and G. Pongracz, ‘‘Programmable event detection for in-band network telemetry,’’ in *Proc. IEEE 8th Int. Conf. Cloud Netw. (CloudNet)*, Nov. 2019, pp. 1–6.

[41] D. Suh, S. Jang, S. Han, S. Pack, and X. Wang, ‘‘Flexible sampling-based in-band network telemetry in programmable data plane,’’ *ICT Exp.*, vol. 6, no. 1, pp. 62–65, Mar. 2020.

[42] *Behavioral Model (BMV2)*. Accessed: Feb. 27, 2023. [Online]. Available: https://github.com/p4lang/behavioral-model

[43] *Stratum, Stratum Official Site*. Accessed: Feb. 27, 2023. [Online]. Available: https://opennetworking.org/stratum/

[44] B. O’Connor, Y. Tseng, M. Pudelko, C. Cascone, A. Endurthi, Y. Wang, A. Ghaffarkhah, D. Gopalpur, T. Everman, T. Madejski, J. Wanderer, and A. Vahdat, ‘‘Using P4 on fixed-pipeline and programmable stratum switches,’’ in *Proc. ACM/IEEE Symp. Architectures Netw. Commun. Syst. (ANCS)*, Sep. 2019, pp. 1–2.

[45] *P4 Integrated Network Stack, PINS Official Site*. Accessed: Feb. 27, 2023. [Online]. Available: https://opennetworking.org/pins/

[46] *Software for Open Networking in the Cloud, SONiC Official Site*. Accessed: Feb. 27, 2023. [Online]. Available: https://sonic-net.github.io/SONiC/

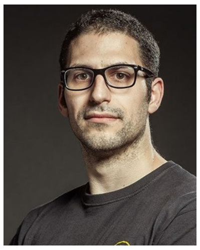
[47] C.-Y. Chang, T. G. Ruiz, F. Paolucci, M. A. Jiménez, J. Sacido, C. Papagianni, F. Ubaldi, D. Scano, M. Gharbaoui, A. Giorgetti, L. Valcarenghi, K. Tomakh, A. Boddi, A. Caparrós, M. Pergolesi, and B. Martini, ‘‘Performance isolation for network slices in industry 4.0: The 5Growth approach,’’ *IEEE Access*, vol. 9, pp. 166990–167003, 2021.

[48] H. Harkous, K. Sherkawi, M. Jarschel, R. Pries, M. He, and W. Kellerer, ‘‘P4RC probe for evaluating the performance of P4 runtime-based controllers,’’ in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2021, pp. 74–80.

[49] B. Chung, C.-C. Tseng, J. H. Chen, and J. Mambretti, ‘‘P4MT: Multi-tenant support prototype for international P4 testbed,’’ in *Proc. ACM/IEEE Symp. Architectures Netw. Commun. Syst. (ANCS)*, Sep. 2019, pp. 1–2.

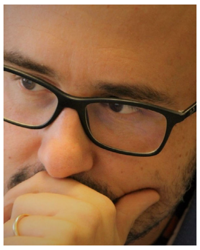
[50] *SD-Fabric*. Accessed: Feb. 27, 2023. [Online]. Available: https://opennetworking.org/sd-fabric/

[51] T. Osiński and C. Cascone, "Achieving end-to-end network visibility with host-INT," in *Proc. Symp. Architectures Netw. Commun. Syst. (ANCS)*, Dec. 2021, pp. 140–143.
[52] *eBPF Project*. Accessed: Feb. 27, 2023. [Online]. Available: https://ebpf.io/
[53] N. Jain, V. K. C. Mohan, A. Singhai, D. Chatterjee, and D. Daly, "Kubernetes load-balancing and related network functions using P4," in *Proc. Symp. Architectures Netw. Commun. Syst.*, Dec. 2021, pp. 133–135.
[54] *5Growth AIMLP Code Repository*. Accessed: Jul. 26, 2021. [Online]. Available: https://github.com/eccenca/braine/tree/main/webclient/
[55] L. Velasco, M. Signorelli, O. G. De Dios, C. Papagianni, R. Bifulco, J. J. V. Olmos, S. Pryor, G. Carrozzo, J. Schulz-Zander, M. Bennis, R. Martinez, F. Cugini, C. Salvadori, V. Lefebvre, L. Valcarenghi, and M. Ruiz, "End-to-end intent-based networking," *IEEE Commun. Mag.*, vol. 59, no. 10, pp. 106–112, Oct. 2021.
[56] F. Alhamed, D. Scano, P. Castoldi, F. Paolucci, F. Cugini, I. Verschkov, and J. J. V. Olmos, "P4 postcard telemetry collector in packet-optical networks," in *Proc. Int. Conf. Opt. Netw. Design Modeling (ONDM)*, May 2022, pp. 1–3.
[57] F. Paolucci, D. Scano, F. Cugini, A. Sgambelluri, L. Valcarenghi, C. Cavazzoni, G. Ferraris, and P. Castoldi, "User plane function offloading in P4 switches for enhanced 5G mobile edge computing," in *Proc. 17th Int. Conf. Design Reliable Commun. Netw. (DRCN)*, Apr. 2021, pp. 1–3.
[58] *Influx Database*. Accessed: Feb. 27, 2023. [Online]. Available: https://influxdata.com
[59] *Prometheus*. Accessed: Feb. 27, 2023. [Online]. Available: https://prometheus.io
[60] *Node Exporter*. Accessed: Feb. 27, 2023. [Online]. Available: https://github.com/prometheus/node_exporter
[61] *Grafana*. Accessed: Feb. 27, 2023. [Online]. Available: https://grafana.com
[62] *V1Model*. Accessed: Feb. 27, 2023. [Online]. Available: https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4



DAVIDE SCANO received the B.S. degree in telecommunication engineering from the University of Pisa, in 2017, and the M.S. degree in computer science and networking from the University of Pisa and Scuola Superiore Sant'Anna, Pisa, in 2019, with a research thesis on SDN for guaranteeing QoS in network slicing. He is currently pursuing the Ph.D. degree in emerging digital technologies with Scuola Superiore Sant'Anna. In 2020, he got a Research Scholarship at Scuola Superiore Sant'Anna. His research interests include software defined networking, next generation software defined networking, optical networks, and disaggregated networks.

ALESSIO GIORGETTI received the Ph.D. degree from Scuola Superiore Sant'Anna (SSSA), Pisa, Italy, in 2006. In 2007, he was a Visiting Scholar at the Centre for Advanced Photonics and Electronics, University of Cambridge, U.K. He was an Assistant Professor at SSSA, from 2007 to 2020. He is currently a Researcher with IEIIT-CNR, Italy. He is the author of more than 100 publications including international journal articles, conference proceedings, and patents. His research interests include optical network architectures and control plane, industrial networks design, software-defined networking, and quantum communications. He is an active software contributor to Open Network Foundation projects.



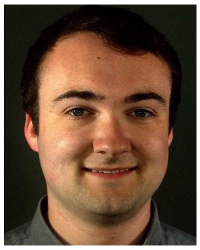
FRANCESCO PAOLUCCI is currently a Senior Researcher with CNIT, Pisa, Italy. His main research interests include network control plane and service orchestration for edge platforms, traffic engineering, network disaggregation, advanced monitoring/telemetry, and SDN data plane programmability. He has been involved in many industrial and European research projects on next generation control networking (E-Photon/ONe+, BONE, NOBEL, STRONGEST, IDEALIST, PACE, 5GEx, 5GTRANSFORMER, 5Growth, METROHAUL, and BRAINE). He is the coauthor of three IETF Internet Drafts, more than 200 publications in international journals, conference proceedings and book chapters, and filed four international patents. He is an Associate Editor of the IEEE/OSA Journal of Optical Communications and Networking (JOCN) and an Executive Editor of the *Transactions on Emerging Telecommunications Technologies (ETT)*.

ANDREA SGAMBELLURI received the Ph.D. degree from Scuola Superiore Sant'Anna, Pisa, Italy, in 2015. He is currently an Assistant Professor with TECIP Institute, Scuola Superiore Sant'Anna. In 2016, he was a Postdoctoral Researcher with the KTH Royal Institute of Technology (Optical Networks Laboratory (ONLab)). He is the coauthor of two IETF Internet Drafts, more than 130 publications in international journals, conference proceedings, and book chapters. His main research interests include control plane for packet and optical networks, SDN/NFV, segment routing, YANG/NETCONF, traffic engineering, network disaggregation, new generation monitoring, and data telemetry. In March 2015, he won the Grand Prize at 2015 OFC Corning Outstanding Student Paper Competition with the article: Demonstration of SDN-Based Segment Routing in Multi-Layer Networks.

JAVAD CHAMMANARA received the B.Sc. degree in software engineering from the University of Science and Culture, Tehran, Iran, the M.Sc. degree in software engineering from the University of Polytechnic, Tehran, and the Ph.D. degree in computer science from Friedrich Schiller University, Jena. He is currently a Researcher, the Project Manager, and a Lecturer with Leibniz University Hannover, Germany, working on distributed knowledge graphs and machine learning techniques. He has developed a new query language (QUIS) for data retrieval from heterogeneous data sources, introduced, JenPlane, a new data life cycle for scientific data management projects. He is Charing the Semantic Industries Community Group, W3C.

JOHN ROTHMAN received the B.S. degree in electrical engineering from Eastern Washington University, with specialization in digital signal processing and robotics, in 2014, and the M.S. degree in data analytics from Hildesheim University, in 2020, with focus in image processing. His thesis topic was in the field of image similarity sampling techniques. He is currently a Research Engineer with the Leibniz University Hannover, L3s Research Institute, and main body of work involves training and deploying Machine Learning applications for scheduling, as well as telemetry monitoring and storage. He has coauthor of two peer-reviewed publications. He was the IEEE Student Chapter Chair, in 2013.

MUSTAFA AL-BADO is currently a Senior Research Scientist with the Dell Technologies with a demonstrated history of working in industries and higher education institutes. He has contributed to several granted patents and standards about 5G technologies. His research interests include virtualization, 5G technologies, network slicing, as-a-service, and cloud and edge computing.



SEAN AHEARNE is currently a Senior Research Scientist with the Dell Technologies and a Technical Lead of the ECSEL BRAINE Project. His background includes extensive work on Software Defined Networking for both optical and wireless networks including 5G and THz with several published works. He has a further background and interest in system virtualization both hypervisor and container-based, hardware acceleration with GPU's and FPGA's, and hardware and software architectures and design.

EDGARD MARX is currently the Project Manager with the Leipzig University of Applied Science and a Linked Data Expert with eccenca GmbH, Germany. His main research interests include theoretical and experimental studies in information retrieval, databases, and knowledge graphs. He has been involved on various international research projects such as DFG DINOBBIO, BMWK COYPU, ECSEL BRAINE, and CLEVER. He has coauthored over 40 peer-reviewed international publications.

FILIPPO CUGINI (Member, IEEE) is currently the Head of Research Area with CNIT, Pisa, Italy. He is the coauthor of 14 patents and more than 300 international publications. His main research interests include theoretical and experimental studies in communications and networking. He serves as a Coordinator of the ECSEL BRAINE Project, an EU-Funded Project aiming to boosting Artificial Intelligence at the Network Edge (www.braine-project.eu).

Open Access funding provided by 'Scuola Superiore "S.Anna" di Studi Universitari e di Perfezionamento' within the CRUI CARE Agreement