Segment Routing for Chaining Micro-Services at Different Programmable Network Levels

Bertrand Mathieu, Olivier Dugeon, Joël Roman Ky, Philippe Graff, Thibault Cholez

Segment Routing for Chaining Micro-Services at Different Programmable Network Levels

Bertrand Mathieu*, Olivier Dugeon*, Joël Roman Ky*, Philippe Graff†, Thibault Cholez†
*Orange Innovation, Lannion, France,
{bertrand2.mathieu, olivier.dugeon, joelroman.ky}@orange.com
†Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France, {first.last}@loria.fr

***Abstract*—Network Function Virtualization (NFV) is now well known for making network services deployable in a virtual environment, for instance in data-centers. On another hand, the programability of functions has come down to the network forwarding equipment itself, such as programmable switches, using the Programming Protocol-independent Packet Processors (P4) solution. Each of the two programmable concepts has its own advantages and drawbacks and micro-services should be preferably developed in one or the other solution depending on their constraints and requirements. In this paper, our objective is to combine both approaches. We propose to leverage Segment Routing (SR) to define a solution allowing to chain the micro-services to be executed at both levels. This signaling protocol is integrated within network equipment but not within a NFV infrastructure. To overcome it, we design an intermediary proxy between the P4 nodes and the VNFs. This proxy is in charge of managing the SR labels and their association with the related VNF. The demonstrator we have developed proves the feasibility of the approach and opens the way towards a composition of network services taking the best of the two levels of programmable networks.**

***Keywords*—Segment Routing, P4, Traffic Detection, Cloud Gaming, Virtualized Network Function**

I. Introduction

For several years, programmability has become increasingly important in network architectures. A first generation of programmable networks was born ten years ago with the Software Defined Networking (SDN) concept and its implementations (e.g. OpenFlow) which offers a first level of control plane programmability. Then, the Network Function Virtualization (NFV) [1] was introduced to enable the deployment of software functions that were initially provided by dedicated network equipment, and more recently to split them into finer micro-services. Today, the data plane programmability is approaching, mainly instantiated by P4 (a programming language for packet forwarding planes) that further extends the concept of network programmability [2][3]. As a consequence, it appears that standard traffic engineering functions, such as routing/switching, filtering, field translation, flow classification, etc. can be implemented through different means, according to these different software environments, and at different topological locations and different layers, thus opening the way to fully end-to-end programmable networks. However, each solution for network programmability (SDN, NFV, P4) exhibits advantages and limitations related to the intrinsic features of its networking environment that are, for instance, related to execution time, resource consumption, programming ability, protocol stack layer, ease of deployment, configuration, migration, etc. For instance, P4 modules can apply simple processing to packets transiting via a hardware P4 switch at high rates, but it cannot handle complex computational tasks, which must be done by application level modules [4]. This calls for a more versatile, multi-level and multi-technology, programmable network solution that defines an effective mean to chain the services deployed at one or another level.

Some orchestration solutions start to include networking programmable technologies in their design (i.e. ONOS integrating SDN and P4), but they do not clearly mention how to chain services running in the data plane or in the control plane. This is even more tricky with the split of network services into micro-services, a recent trend to obtain a finer granularity and to be able to deploy only the necessary micro-services where needed and take advantage of a better horizontal scalability. Indeed, if a few research works propose automatic methods to help monolithic applications to be split into micro-services, selecting essential network functions to convert into micro-services (and their level of division) is still a complex problem [5] and a convenient way to chain them, that would also be compatible with P4 data plane modules, is of high interest.

In this paper, we propose to use Segment Routing (SR) [6], [7], based on MPLS (Multi-Protocol Label Switching) labels, for the service chaining of micro-services. We advocate that label stack is used here to route packets according to the service composition, with the orchestration algorithms deciding how to compose the micro-services, and considering different parameters, such as the execution environment, the network topology, the required latency, the nodes' load, etc. There are consequently two main challenges addressed by our architecture:

1) mixing and routing the execution of micro-services at the two levels (P4 and NFV) and instantiating SR-MPLS labels for both levels;
2) managing the removal of SR-MPLS labels when packets are forwarded from a P4 node to a NFV compute node since code in the compute node does not have to manage SR-MPLS labels.

The solution we propose takes the form of a proxy module interacting with the compute nodes.

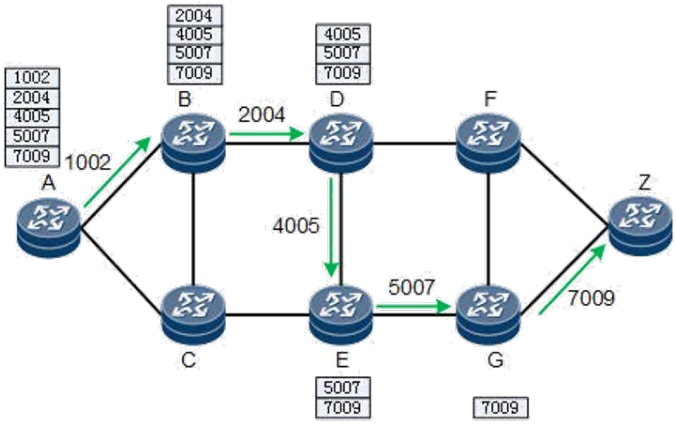

Fig. 1: Segment Routing Approach, from [11]

In this paper, we first present in Section II the background and related work of P4 and NFV for hosting network functions and SR-MPLS for service chaining. In Section III, we detail how SR-MPLS can be used for chaining micro-services running in data-plane (P4) and control plane (NFV). We then describe our SR-MPLS aware P4 switch in Section IV and our SR-MPLS aware proxy for VNFs in Section IV. We present in Section VI our demonstrator and its evaluation. Finally, we discuss the reliability and scalability of our solution in Section VII before concluding in Section VIII.

II. Background & Related Work

A. Network softwarization technologies

Most network devices, such as, routers, switches, etc. were not, at least initially, programmable. These devices were dedicated to specific network functions and, once deployed, only realized those. It was not possible to change their behavior. For several years from now, SDN (Software-Defined Networking), using the Open Flow protocol, allows to program the control plane of the network thanks to a controller that can send dynamic configurations to the managed switches. This gives an abstraction and a more global view of the network. SDN and NFV are the key enabling technologies for 5G networks [8]. More recently, P4 (Programming Protocol-Independent Packet Processor) language [3][9] has been designed to allow administrators and developers to fully configure the way packets are processed in a network programmable equipment. It introduces the ability to program the devices according to the controller's specific requirements, rather than being constrained by a traditional switch design based on address' prefix. With P4, it becomes easier to implement new packet processing and to quickly deploy them while guaranteeing a hardware enforced throughput. P4 is potentially becoming the disruptive technology [10] enabling the programming and customization of the data plane of next-generation SDN/NFV-based mobile networks. It opens the door for a finer management of network packets in network devices and allows a better communication between devices and their controller by introducing an interface between them.

NFV being known for a while, many research papers have been published in relation to the service chaining of VNFs (Virtualized Network Functions). In this section, we will not survey this exhaustive literature, but will rather focus on the few solutions that propose a conjoint use of NFV and P4. The very few papers dealing with NFV and P4 altogether do not address the service chaining between micro-services running at the two programmable levels as we do, but rather propose to use P4 as an helper for NFV. [12] proposes to use a P4 switch to forward the incoming packets to the right VNF in an efficient way, with a better performance than software solutions and in a more flexible way than OpenFlow. Their P4 switch uses tables that can be dynamically modified to fit new requirements for service chaining to manage the VNFs connected to it. [13] suggests to develop some network mobile services in P4 programs, instead of VNFs, for performance reasons. The authors present the use-case of the 5G mobile network and propose to offload some functions such as the User Plane Function (UPF) to P4 switches. They use modules at the 2 levels, but do not mention the chaining between them. [14] deals with P4 switches and SmartNICs deployed in data centers to accelerate packet processing and the challenges in managing them together. The paper proposes a host-local SDN Agent to improve the overall resource utilization, considering the network, host, and sNIC specific capabilities and constraints. Based on workloads and traffic characteristics, P4NFV determines the partitioning of the P4 tables and optimal placement of services to minimize the overall delay and maximize resource utilization. In [15], the authors mention that NFV services are not really designed to cope with low-level networking functions and propose a framework where network functions can be offloaded from VMs to the P4 infrastructure. The authors propose to use the Dynamic Optimization of Packet Flow Routing mechanism, defined by ETSI to forward some flows from the P4 node to the VNF connected to it, but it is not a service chaining solution for a whole composite chain of modules being identified as micro-services. Finally, in [16], the authors argue that service chaining for NFV is done either at the software layer, but with performance issues, or with NIC/FPGA, but with a limited processing capability. They propose to implement the service chaining in P4. Operators can define the service chains and configure the P4 programs accordingly to route packets. In their solution, the P4 program is limited to the chaining and does not offer any other network function.

In this paper, we promote the combined use of both data-plane programmable networks using P4 and control-plane programmable networks with VNFs and we argue that Segment Routing (SR) is a good candidate for chaining micro-services at different levels.

B. Segment Routing

Segment Routing (SR) is an architecture [6][7] that leverages the source routing paradigm, where a packet carries the path to reach its destination in its header, as illustrated in Figure 1. Segment Routing is currently being standardized

by the Internet Engineering Task Force (IETF) which has already published the main RFCs defining this technology. It can be instantiated over two existing data planes, namely MPLS (Multi-Protocol Label Switching) [17] and IPv6 [18]. SR-MPLS is mostly used in networks to route packets between nodes (i.e. IP routers). First, SR architecture defines Segment Identifier (SID) that can identify a node, a link, a service, or more generally whatever that is reachable in the network. For SR-MPLS, SID are represented by a MPLS label while for SRv6, SID are represented by a special IPv6 header (i.e. the SR header). At the control plane level, SR defines mechanisms to exchange SID between nodes. Based essentially on Internet Gateway Protocol (IGP), it simply reuses the prefix advertisement mechanism to associate SID to these prefixes. Thus, when a node receives such prefix + SID information, it configures its Label Forwarding Data-Base (L-FIB) to determine how to reach a given SID and how to route incoming SID.

This architecture has generated a lot of enthusiasm [19] among Service Providers (SP), due to the simplification that it brings to their IP/MPLS networks. In fact, when Segment Routing is instantiated over the MPLS data plane, there is no need to pre-establish tunnels. Therefore, no signaling protocol such as LDP and/or RSVP-TE is required. Consequently, the number of states maintained in the network is considerably reduced and the per-flow states are maintained only at the edges of the network. In several defined use cases, Segment Routing has proven its superiority in comparison to standard IP/MPLS mechanisms [18]. In particular, it natively supports Equal-cost multi-path (ECMP) and delivers full network coverage of link and node protection for a fast recovery in case of failure. Segment routing simplifies the deployment of VPNs, allows Tactical Traffic Engineering by explicitly specifying the paths that respect the Quality of Service (QoS) requirements and eases network monitoring and measurement. More recently, IETF has standardized the Flexible Algorithm (Flex-Algo) [20] which allows a router to compute the next hop based on different metrics, in particular the delay metric to enable latency aware routing.

If the primary use case for Segment Routing is to ease instantiating IP/MPLS or IPv6 paths on IP networks, the mechanism can also be applied to Service Chaining for VNFs. In a first approach, VNFs can be linked together by enforcing the stack of MPLS labels [21] on top of the packet at the edge node. This requires that each intermediate node contains the corresponding VNF. Another approach consists to deploy VNFs that are SR-MPLS aware [22]. In both cases, VNF deployment is constrained by i) the routing node capacity, or ii) the nature of VNFs that should be SR-aware. Similarly to us, [23] proposes a SR-Proxy for VNFs which are not SR-compliant. However, this work differs a bit since they mainly address SRV6 and not MPLS and mostly because the authors propose a proxy directly integrated into the Linux kernel of the VNF node. In our paper, we rather propose a distinct node, which offers the SR-proxy function, such that the VNFs and their operating system do not need any modification.

In a previous work [4], we proposed an hybrid P4/NFV architecture for the detection of Cloud Gaming traffic. This paper proposes a significant improvement since the different micro-services are now properly chained using SR-MPLS, whereas in the previous paper, the chaining was static and all the VNFs were located on the same node, hardly connected. This new architecture offers more flexibility and scalability and the possibility to chain dynamically the micro-services.

III. Segment Routing for multi-level service chaining

In this paper, we propose to use SR-MPLS to chain services both for P4 programs and VNFs. As SID could represent any kind of reachable resource in the network, we propose here to associate each micro-service with a SID (i.e an MPLS label for SR-MPLS) and the global service is composed by using a stack of SID (a label stack for SR-MPLS). Service chaining consists then to route packets between micro-services according to the MPLS label that has been associated to the given micro-service and to stack of labels to chain micro-services. The first label will be related to the first micro-service to execute, then the second label, etc. until the end of the stack is reached for the final micro-service. When a micro- service is executed, its label is removed from the stack.

We can see on each side of Fig.2 the client and the server of the global service, and in-between the P4 nodes and NFV nodes that process the packets. Above, the orchestrator configures the nodes to define the service chaining to be carried out. In this version, we use a centralized controller which configures the different programmable nodes, but, as indicated in section II, it is possible that the nodes announce their identifier by themselves. This allows each local node to keep an association between the MPLS label and the destination of a programmable node which can carry out this processing. Thus, if several nodes can offer the service, it is possible to balance the load or select one node according to a strategy considering the network configuration at a given time.

Fig 2 shows the two levels of the programmable network, with at the low level, the P4 nodes, that are designed to carry out simple and very high-speed processing on all packets passing through them, and at the high level, the VNF application modules rather intended for complex/computation-intensive processing, but not necessarily at rate line. They can therefore carry out processing on only certain packets or work on some statistics reported by P4, for instance to apply a Machine Learning Model like in [4]. In between we see the proxy we developed which is responsible for handling SR-MPLS signaling for the VNF modules. The objective is that the VNF modules are not impacted by the architecture, and can be deployed as they currently are. It is therefore the responsibility of the proxy to analyze the MPLS signaling of the packet, to detect if a label corresponds to a VNF processing, to interact with this VNF (send the packet to it and receive the answer) and then send the packet to the next module, being either another VNF in the same data center or a more distant one,

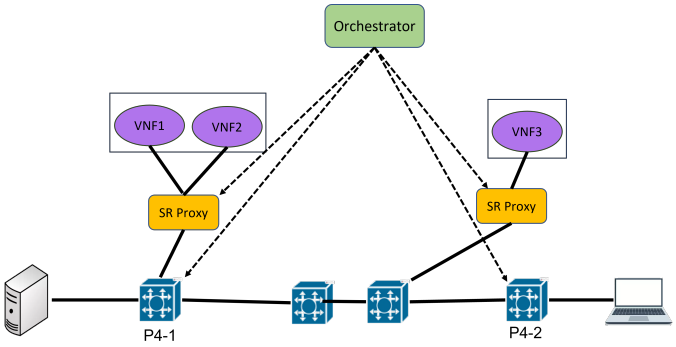

Fig. 2: Architecture of the 2 levels SR-MPLS based programmable network

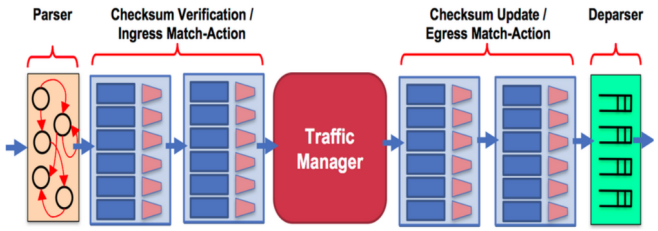

Fig. 3: Architecture of P4 programmable program, from [24]

or a P4 node. In the latter case, the packet is rerouted to the P4 node, which forwards it further in the network.

IV. Segment Routing in P4 switch

A P4 programmable switch is made up of 4 main components that constitute the complete pipeline to process and forward packets, namely the Parser, the Ingress control, the Egress control and the Deparser. Fig.3 shows the 4 main components of a P4 program. Being given this architecture, we describe below where we add the specific processing to support SR-MPLS in P4.

- The Parser allows to select the different headers of the packets on which the program will be able to make actions (Ethernet, IPV4, UDP, ...). The fields of these headers should be declared in the P4 program or in an external file, referenced by the P4 program. To integrate SR-MPLS in our architecture, we add here the parsing of the MPLS protocol, a recursive loop of the MPLS labels, until the Bottom of Stack (BoS) field is 0 (what indicates the bottom of the label stack).
- Actions can then be applied in the Ingress part, using a match-action table. The controller configures entries of the table, with different actions to perform, based on input values of the packet headers. This match-action table system is at the heart of P4. The controller can, at any time, decide to change the configuration of the table, the entries, the actions to process, etc. This allows a dynamic configuration of the network. Then, the packets are placed in a queue (Traffic Manager). This part is not programmable currently and this is one of the limitations of P4, which is not able to program queues dynamically. For the SR-MPLS use-case, the orchestrator configures the P4 program to check if the incoming packet should be processed by the current P4 node or forwarded to a next node. In the former case, the P4 program applies the function related to the expected processing. Then it applies the function to forward the packet to the next node.
- Leaving the traffic manager queue, packets are processed with actions and tables at the Egress part, in a similar way to the Ingress.
- Finally, the Deparser reconstitutes the packets, adding the extracted (eventually modified) headers to the packet payload, and sends them to the network from the Egress port.

V. Segment Routing Proxy for VNFs

The major part of this work concerns the SR-MPLS proxy placed between the P4 nodes and the VNF modules. Indeed, SR- MPLS is a network protocol intended to be used between network equipment, but which is not understood by end points. We could have integrated the SR-MPLS protocol into the protocol stack of the VNF endpoints to make them compatible, but our objective being to provide an architecture requiring no modification of the current VNFs for a seamless integration, we opted for another solution. We define a proxy which analyzes MPLS labels, associates them with the local VNF, sends the received packets to the correct VNF, receives the packets back (possibly modified by the VNF), and then forwards them to the next VNF. Fig.4 describes the operational architecture of the proxy.

As for the P4 node, the first task of the proxy is to analyze the first MPLS label of each packet arriving via the P4 switch, and to evaluate this label according to the configuration table sent by the controller. Then, the proxy sends this packet to the VNF associated with the label, removes this first label from the MPLS stack and stores the remaining MPLS fields if there are any, to be able to chain the remaining micro-services afterwards.

The most tricky part of this proxy is to keep the state of the connections with the VNFs and the labels stack associated with the sessions, to maintain the chaining of the following services to be carried out on the packets. To achieve this, we use a hash function for each packet based on the session identifiers, such as the IP addresses and port numbers. The hash is computed and stored for packets sent to the VNFs. When packets are sent back by the VNFs, the proxy first computes the hash of the incoming packet, compares it with the stored hashes to identify the session, and then adds to the packet the MPLS fields of the original packet that were previously stored.

Regarding the communication between the VNFs and the proxy, sockets are used. This choice is motivated by the fact that they allow to establish connections between applications running on separate machines and are already implemented in current VNFs modules (for example running inside containers), then without requiring any modification.

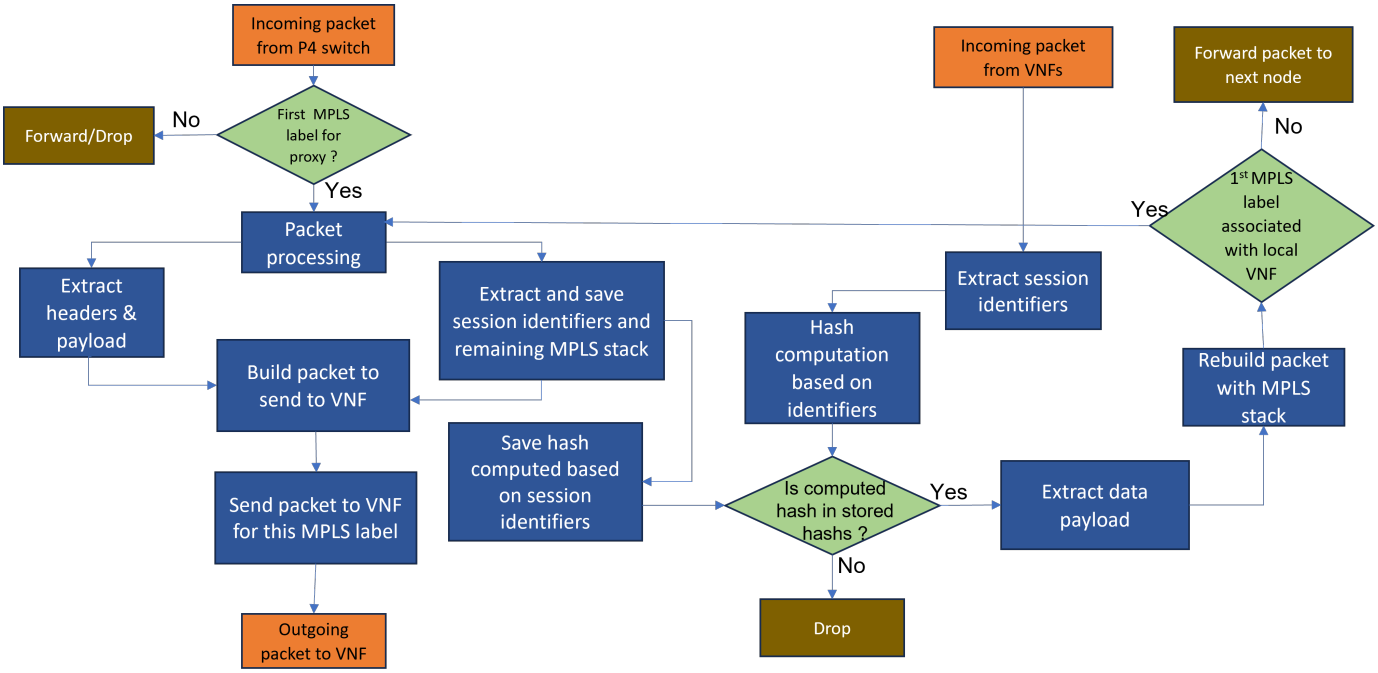

Fig. 4: Algorithm of the proxy for managing MPLS labels and VNFs

Afterwards, the proxy restarts the whole process and first analyses the 'new' first MPLS label. If it is associated to another VNF the proxy is in charge of, the same processing applies. Otherwise, the packet is sent back to the P4 switch to be forwarded on the network to the next SR-MPLS node.

VI. Demonstrator

For illustrating our service chaining solution based on SR-MPLS, we implemented a use-case of in-network Cloud Gaming (CG) traffic detection [4]. The goal is to identify cloud gaming sessions, in order to prioritize this traffic which requires together a high bit-rate and a low-latency, compared to other best effort traffic. Typically, cloud gaming flows should constitute a traffic class that would benefit from a Hierarchical Token Bucket queuing discipline featuring a small (to avoid queuing delays) but dedicated queue (to avoid other applications' loss-based transport protocols to fulfil the queue). More details about this use-case can be found in previous papers [4], [25]. Fig. 5 presents our demonstrator, with the different deployed programs (P4 and VNF) and the MPLS stack to chain the global service to offer.

A. P4 modules

We developed two P4 modules and deployed them in one hardware switch. The first module extracts session information from all incoming packets from end-users and computes several features per 33ms time window such as the number of bytes, the average packet inter-arrival time, the number of packets, etc. At the end of a 33ms window, the P4 module mirrors one incoming packet and adds all computed features to the packet to be sent to the NFV modules for further computational processing which results in the binary classification of the flow as cloud gaming traffic or not. The destination of this mirrored packet is the first NFV module, being identified by the next MPLS label in the incoming packet. Regarding other received packets that do not have to be transmitted to NFV modules, but just forwarded to the next network node, the P4 switch removes the MPLS labels related to the NFV modules in addition to the removal of its own label. The second P4 module prioritizes cloud gaming traffic versus the best effort traffic using a dedicated queue. This P4 module is aware of the nature of the session thanks to the controller which sends the 5-tuple of the session to be prioritized.

The P4 hardware switch we used is an Edgecore Wedge 100BF-32X, with 32 QSFP28 ports supporting each 100 GbE, and embedding a P4 programmable Intel/Tofino1 chipset. The P4 modules are developed with the TNA (Tofino Native Architecture) SDE (Software Development Environment), a set of tools to create and test network functions for Intel Tofino switches. It includes a P4 compiler, a network simulator and a set of testing tools. The version used for this work is SDE 9.9.1.

B. NFV micro-services

For the complex computational task that is traffic classification, we implemented 3 NFV modules[1]. A first one is in charge of checking the validity of the digest packet and selecting the next processing node to perform load balancing and horizontal scalability. The next one executes a ML model and gives a value of true or false regarding the incoming analysed digest packet that reports the extracted statistical

[1]The source code of the VNFs is available: https://github.com/mosaico-anr/CG_Classifier

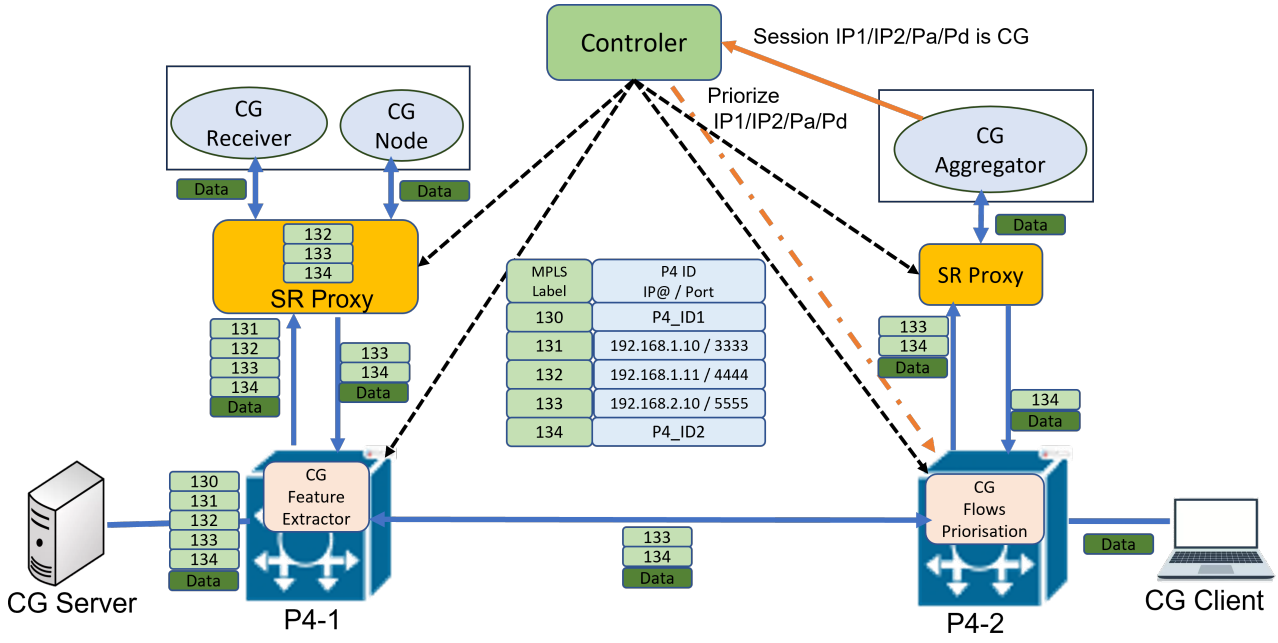

Fig. 5: Demonstrator using SR-MPLS to chain P4 and VNF micro-services for cloud gaming traffic detection and prioritization

feature of a flow (true in case of a cloud gaming traffic pattern recognized by the ML model, false otherwise). Finally, the last NFV module aggregates information from the previous classification decisions of a given flow's samples and makes the final decision. In our demonstrator, the 3 modules are distributed in 3 virtual environments, such as we could have, for instance, 2 VNFs in a same data center, and the third one in another (as shown in Fig. 5). In this configuration, we have 2 SR-proxies, one for each data center.

C. Proxy

We have developed the proxy in Python (version 3.8). For processing packets, we use the Scapy library for its flexibility for creating, sending, capturing, modifying and analyzing packets. It is not meant to provide high performance but it was deemed sufficient for a proof of concept. We use Hashlib to calculate the hashes, with the goal of specific security and data integrity. JSON (JavaScript Object Notation) is used for the serialization and deserialization of data into a format that can be easily stored, transmitted, or exchanged between different applications or systems. Finally, for performance issues, the proxy program is multi-threaded.

The demonstrator is working as expected. On Fig. 6, we show that the Aggregator (final NFV micro-service of the chain) has well classified 2 different flows transiting on the network at the same time : 1 is a cloud gaming session, for which 88% of samples have been detected at as such resulting in the correct labelling of the flow, the other one is a file transfer, for which no sample (0%) is considered as CG. For each session, we made the test for a time duration leading to the computation of 1000 reports (containing the statistical features per time window and that are computed by the first

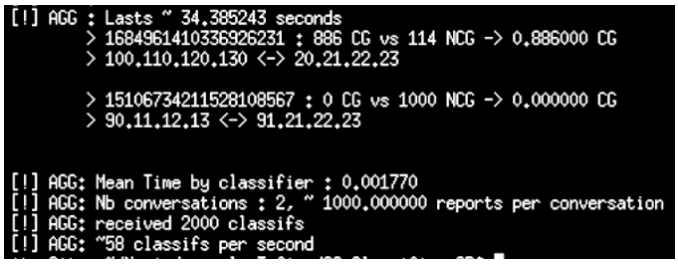

Fig. 6: Classification of 2 flows: 1 CG and 1 not CG

P4 module) and we can see that all the packets have been received, without loss, in the right order of the service chain.

With this demonstrator, we only aim to prove the feasibility of our approach and let the evaluation of performance metrics for future works. Indeed, the hardware P4 is not an issue because its performance is guaranteed at compilation time of the P4 programs, and will thus operate at its specifications' line rate. Regarding the VNFs, we developed them simply in Python and evaluated them in Linux namespaces, without leveraging DPDK (Data Plane Development Kit). It is then not representative of current NFV deployment, and as such, performance tests are not relevant.

In this paper, we do not focus on the orchestrator/controller part. In our demonstrator, it is a simple Python program that sends configuration data (mapping between MPLS labels and node destination). But we can imagine to integrate it with ONOS (Open Networking Operating System) or other popular orchestrator solutions.

VII. Reliability and Scalability

In a first approach, we use an SDN controller to configure the SID for each micro-service and compute the service chaining as a stack of MPLS labels. However, in a standard SR-MPLS network, the IGP (Interior Gateway Protocol, e.g. OSPF or IS-IS) is used to exchange the SID between routers. In turn, each router configures its Label FIB (Forwarding Information Base) to mark packets with the corresponding SR-MPLS label for a given prefix, or to forward packets based on their SR-MPLS label. Applied to micro-services, the IGP could also exchange the micro-service SID in the same way it exchanges prefix SID. For that purpose, the SDN controller should just add new SIDs configuration in P4 nodes. These SIDs are associated to the interfaces:

- where the compute node is connected to a micro-service running on a dedicated node;
- to a given loopback interface when the micro-service is associated to the P4 code itself.

The SR architecture offers natively two main advantages that our architecture inherits: the possibility to auto-repair packet forwarding in case of failure without creating micro-loop and the possibility to manage Equal Cost Multipath (ECMP) as well as Anycast routing. The first one relies on Loop Free Alternate (LFA) and Topology Independent Loop Free Alternate (TI-LFA). Routers or nodes are able to automatically compute an alternate route in case of failure and install this route into the LFIB. The alternate route is chosen to minimise the failure and avoid packet loss. Once detected by the IGP, and the routing protocol convergence done, a new default route is installed and a new alternate one is computed until the failure is resolved. Such failure mitigation is done automatically and within the so-known sub 50 ms objective. The second one allows, in an SR-MPLS network, the possibility to advertise the same SID from different locations. Thus, it is then possible to deploy the same micro-service in different locations in the network and configure them with the same SID. Automatically, the routing protocol will compute the path to the nearest SID location by using its Shortest Path First algorithm associated with the routing protocol. Again, in case of failure of a micro-service itself, packets will be automatically redirected to the next micro-service that has been advertised with the same SID. Finally, ECMP could be also used to load-balance packets between similar micro-services. For that purpose, similar micro-services using the same SIDs should be deployed in the network at an equal distance from the users. Flows using the same micro-services will then be automatically shared between the micro-services.

In addition, a new feature provided by SR-MPLS is Flex-Algo. This function gives a router the possibility to advertise several SID for a same prefix but with different routing objectives. In fact, a router computes the Shortest Path First (SPF) of the SID to install the next-hop route in its FIB and L-FIB based on the standard metric. The Flex-Algo standard allows the definition and exchange of Traffic Engineering (TE) metric and Delay metric in order for a router to compute SPF based on these TE or Delay metrics instead of standard metrics. Applying again to a micro-service chaining, delay based Flex-Algo will allow to automatically chain micro-services with a low latency objective without the need of an external SDN controller for that purpose. Like with other standard metrics, if a delay metric evolves in the network over time, routers will automatically re-compute the new SPF based on the new delay metric value without the need of an external SDN controller.

So, our solution takes advantage of routing protocol and native SR-MPLS network features such as TI-LFA, Anycast, ECMP and Flex-Algo, so that we can easily manage the reliability and scalability of the network functions that compose the service chain.

VIII. Conclusion

The P4 and NFV architectures both enable programmable networks, but run at different levels and both having their own intrinsic strengths and weaknesses. In this paper, we proposed to connect these two levels of programability to allow the execution of a global service chaining micro-services operated at both levels. This allows to make the most of each of the programmable technology, ie. P4 programs for simple and rapid processing of all packets, NFV modules to carry out more complex processing tasks, typically on a smaller number of packets. The proposed solution is based on SR-MPLS. To support SR-MPLS, we developed a proxy interfacing with the P4 modules and with the VNFs to avoid modifying them and thus keep their initial nature of reusable components. We developed a demonstrator to detect Cloud Gaming traffic based on our solution chaining P4 and VNF micro-services, with the objective to prioritize this traffic over other best effort traffic, which proved the feasibility of the approach.

In this paper, the orchestrator is a simple python program, responsible for configuring the P4 nodes and the proxy to know how to process the MPLS labels present in the packets. A possible future work would be to integrate this feature into a widely used orchestrator such as ONOS or MANO. Another evolution would be to use Segment Routing features, such as the automatic announcement of the identifiers to populate MPLS routing tables in the P4 and proxy nodes.

Acknowledgments

This work is partially funded by the French ANR MOSAICO project, No ANR-19-CE25-0012. The authors would like to thank Nazim Ayad, who contributed to this work.

References

[1] J. d. J. Gil Herrera and J. F. Botero Vega, "Network functions virtualization: A survey," *IEEE Latin America Transactions*, vol. 14, no. 2, pp. 983–997, 2016.

[2] W. L. Costa Cordeíro, J. A. Marques, and L. P. Gaspary, "Data plane programmability beyond openflow: Opportunities and challenges for network and service operations and management," *J. Netw. Syst. Manage.*, vol. 25, no. 4, p. 784–818, oct 2017. [Online]. Available: https://doi.org/10.1007/s10922-017-9423-2

[3] P. W. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. E. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: programming protocol-independent packet processors," *Comput. Commun. Rev.*, vol. 44, pp. 87–95, 2013.

[4] J. R. Ky, P. Graff, B. Mathieu, and T. Cholez, "A Hybrid P4/NFV Architecture for Cloud Gaming Traffic Detection with Unsupervised ML," in *2023 IEEE Symposium on Computers and Communications (ISCC)*, 2023, pp. 733–738.

[5] G. Toffetti, S. Brunner, M. Blöchlinger, F. Dudouet, and A. Edmonds, "An architecture for self-managing microservices," in *Proceedings of the 1st International Workshop on Automated Incident Management in Cloud*, ser. AIMC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 19–24. [Online]. Available: https://doi.org/10.1145/2747470.2747474

[6] C. Filsfils, S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir, "Segment Routing Architecture," RFC 8402, Jul. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8402

[7] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. François, "The segment routing architecture," *2015 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:6211975

[8] M. C. Nkosi and A. A. Lysko, "The use of p4 for 5g networks," 2018.

[9] M. Budiu and C. Dodd, "The p416 programming language," *ACM SIGOPS Operating Systems Review*, vol. 51, no. 1, pp. 5–14, 2017.

[10] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, and P. Castoldi, "P4 edge node enabling stateful traffic engineering and cyber security," *Journal of Optical Communications and Networking*, vol. 11, no. 1, pp. A84–A95, 2019.

[11] "Segment Routing," [Online; last accessed 10/2023]. [Online]. Available: https://support.huawei.com/enterprise/en/doc/EDOC1100278569/3ee10304/understanding-segment-routing-mpls

[12] M. Elangovan, C.-C. Chen, and J. cheng Chen, "A flexible vcpe framework to enable dynamic service function chaining using p4 switches," *2021 22nd Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 342–347, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:238993970

[13] F. Paolucci, F. Cugini, P. Castoldi, and T. Osiński, "Enhancing 5g sdn/nfv edge with p4 data plane programmability," *IEEE Network*, vol. 35, pp. 154–160, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:234970474

[14] A. Mohammadkhan, S. Panda, S. G. Kulkarni, K. K. Ramakrishnan, and L. N. Bhuyan, "P4nfv: P4 enabled nfv systems with smartnics," *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 1–7, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:210832792

[15] T. Osiński, H. Tarasiuk, L. Rajewski, and E. Kowalczyk, "Dppx: A p4-based data plane programmability and exposure framework to enhance nfv services," *2019 IEEE Conference on Network Softwarization (NetSoft)*, pp. 296–300, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:201622248

[16] X. Chen, D. Zhang, X. Wang, K. Zhu, and H. Zhou, "P4sc: Towards high-performance service function chain implementation on the p4-capable device," *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 1–9, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:84836618

[17] A. Bashandy, C. Filsfils, S. Previdi, B. Decraene, S. Litkowski, and R. Shakir, "Segment Routing with the MPLS Data Plane," RFC 8660, Dec. 2019. [Online]. Available: https://www.rfc-editor.org/info/rfc8660

[18] C. Filsfils, P. Camarillo, J. Leddy, D. Voyer, S. Matsushima, and Z. Li, "Segment Routing over IPv6 (SRv6) Network Programming," RFC 8986, Feb. 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc8986

[19] P. L. Ventre, S. Salsano, M. Polverini, A. Cianfrani, A. Abdelsalam, C. Filsfils, P. Camarillo, and F. Clad, "Segment routing: A comprehensive survey of research activities, standardization efforts, and implementation results," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 1, pp. 182–221, 2021.

[20] P. Psenak, S. Hegde, C. Filsfils, K. Talaulikar, and A. Gulko, "IGP Flexible Algorithm," RFC 9350, Feb. 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9350

[21] E. Battiston, D. Moro, G. Verticale, and A. Capone, "Chima: a framework for network services deployment and performance assurance," in *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*, 2022, pp. 163–170.

[22] C. Portegies, L. Boldrini, M. Kaat, and P. Grosso, "Experience with implementing vnf chains with segment routing and pcep," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2021, pp. 797–811.

[23] A. Mayer, S. Salsano, P. L. Ventre, A. Abdelsalam, L. Chiaraviglio, and C. Filsfils, "An efficient linux kernel implementation of service function chaining for legacy vnfs based on ipv6 segment routing," in *2019 IEEE Conference on Network Softwarization (NetSoft)*, 2019, pp. 333–341.

[24] "P4 Consortium," 2021, [Online; last accessed 10/2023]. [Online]. Available: https://p4.org/

[25] P. Graff, X. Marchal, T. Cholez, B. Mathieu, and O. Festor, "Efficient Identification of Cloud Gaming Traffic at the Edge," in *NOMS 2023 - 36th IEEE/IFIP Network Operations and Management Symposium*, Miami, United States, May 2023, p. 10. [Online]. Available: https://inria.hal.science/hal-04056607