An extended and flexible SDN control plane

Jean-Michel Sanner, Pierrick Louin, Yassine Hadjadj-Aoul, Meryem Ouzzif

An extended and flexible SDN control plane

Jean-Michel Sanner*, Pierrick Louin*, Yassine Hadjadj-Aoul†, Meryem Ouzzif*,
*Orange Labs, Cesson Sévigné, France
†INRIA, Rennes, France

***Abstract*—In this paper we demonstrate the feasibility of an extended and flexible SDN control plane that allows to overcome the limitations of the Openflow protocol by achieving distributed and intelligent network services in SDN networks. This extended control plane is designed according to the following reference guidelines;**
1) the concept of generic and programmable network nodes usually known as "white boxes". They integrate a generic engine to execute the service and a library of elementary components as basic building blocks of any services;
2) a fine grained decomposition logic of network services into elementary components, which allows the services to be designed and customized on the fly using these building blocks available on each network node in libraries;
3) a mechanism for re-configuring or redefinition on the fly of the network services on generic nodes without service interruption;
4) some smart elementary agents called SDN controllers elements to provide and distribute the intelligence necessary to interact with the data plane at different levels of locality.

This SDN control plane is illustrated in a proof of concept with the implementation of a distributed monitoring service use case. The monitoring service can act and evolve in a differentiated manner in the network depending on traffic requirements and monitoring usage.

***Index Terms*—Programmable, Data plane, Controller, SDN**

I. Introduction

During the last decades, several network architectures, such as the IETF Policy Based Networking and ForCes, have emerged to make the network more flexible. The most emblematic evolution is certainly that which consists in separating the control plane from the data plane, which is proposed in Software-Defined Networking (SDN).

By centralizing the control function, SDN makes it easier to instantiate new services and applications. In fact, SDN eliminates the need for distributed protocols, replacing them with a simpler protocol, like OpenFlow [1], for communication between controllers and switching equipments. The effectiveness of the Openflow protocol, which does not require equipments' renewal, is certainly at the origin of its wide deployment by infrastructure operators, making it the de facto SDN protocol. However, OpenFlow based SDN architectures are strongly limited in their functionalities according to different dimensions. In its essence, the SDN approach is centralized with one controller controlling a large cluster of very simple switches in the network. It is therefore difficult to implement efficiently in a distributed way some usual and classical functions currently used in networks. This is the case for instance with the functions which manage large amounts of traffic for firewalling, security or packet inspection, or the functions which need some locality like fast rerouting to react on time, or finally those that require intelligent analysis on network nodes like monitoring.

We propose in this paper an extended, flexible SDN control plane which is able to overcome these limitations. It allows to manage dynamically, on the fly, and in a differentiated way, networks functions which are distributed in network nodes. We illustrate this concept through a monitoring use case and we demonstrate that we are thus able to introduce flexibility, dynamicity and adaptability in such functions to implement various scenarios of service deployment.

The remainder of this paper is organized as follows. Section II details the related works. Section III introduces the considered design guidelines. Section IV describes the monitoring use case. Section V gives more details about the proof of concept implementation details. Section VI discusses the obtained results. Finally, the paper concludes in Section VII.

II. Related works

As we mentioned, the Openflow protocol has inherent limitations. There are some drawbacks to shifting all logic to the controller level. Indeed, taking all decisions on a remote point can present problems of latency and scalability [2].

Different solutions were proposed to tackle the issues introduced beforehand. The distribution of controllers was one of the first approaches considered to solve these problems while increasing the reliability of these networks. In [3], the authors proposed a new framework, named *Elasticon*, for the dynamic placement of controllers. Other approaches, quite similar, have been proposed in the literature, some of which have been very successful with network operators such as ONOS [4].

To optimize operation, some approaches present a hierarchical architecture, in which one or more high-level controllers handle events requiring a global vision and lower-level controllers handle events requiring a local vision [2]. In [5], H. Yeganeh et al. proposed *Kandoo*, a hierarchical set of controllers compatible with OpenFlow. They define a notion of local controllers directly managing a set of switches and a root controller that benefit from a global view of the network. Although the paper shows impressive results, it is to notice that the application scope of this solution is restricted to environments where local decisions are predominant such as Data Centers' environments. A similar approach, named *Orion*, was presented, in [6]. One main difference with Kandoo's architecture lies in the role of each control plane. In *Orion*, the local controllers are managing an entire area and propose an abstract view of their areas to a global area manager. Another difference with *Kandoo* is the intrinsic distribution of the Domain Controllers. Whereas *Kandoo* pleads for a "logically" centralized global controller, *Orion* designed its domain controller layer with several global controllers interacting on the inter-sub-domain management through a distributed protocol. While efficient and powerful, these solution inherit the shortcomings of Openflow. This means that the provision of on-demand functions remains difficult to implement and localized new decision-making is still very limited.

Enriching switches, traditionally lacking intelligence in Openflow, with a stateful per flow processing, as proposed in [7], allows for much more elaborated functionality. However, even if the concept generalizes the Openflow match/action rules and offloads the central controller, the type of supported functions remains restricted to rather simple functions.

To go further in the complexity of the supported functions, some recent contributions suggest data plane programming. Based on the open source Cisco's Vector Packet Processing (VPP) [8], the authors proposed in [9] an extension using P4 language [10] to create plug-ins, which can be dynamically swapped. Similarly, the authors of [11] proposed the *BPFabric* platform to centrally program and monitor the data plane using extended Berkeley Packet Filter (eBPF). In order to ease interaction with such low level framework, the authors proposed the use of a high level language, like P4, and to compile it to eBPF[1].

These two contributions are in line with what we proposed earlier [12]. Indeed, we proposed to model services using a high level language based on Petri networks, which allows not only to optimize services, as a P4 compiler would do, but also to extract qualitative properties. On the other hand, the architecture we have proposed allows a global and hierarchical management of a network, by delegating local processing logic to local controllers. This distribution, in the operation, allows to reduce the complexity in the management and to better manage networks at scale.

In this paper, we propose a proof of concept using the Click platform [13], in which we have previously developed different modules [14]. Note that our solution could have been developed based on eBPF or even VPP, which are more adapted to virtualized environments.

III. Design guidelines

A. Guidelines and principles of the proposed architecture

We proposed a general framework of a service driven SDN architecture in [12]. At a high level, the architecture consists of:

- An orchestrator whose goal is to assemble, deploy and carry out network services in the network based on templates describing elementary network services, which are stored in a repository.
- Distributed controllers that drive the network elements and are in charge of executing the network services. They form together what we call an extended and distributed SDN control plane.
- Simplified, generic, programmable and efficient network elements (white boxes) in charge of the forwarding plane, which are driven by the local controllers in order to implement the different network services.

Two principles underlie and guide the definition of this framework.
Firstly, the introduction of models enabling the dynamic definition of network services. These models rely on templates that describe elementary network services. They are used and assembled by the orchestrator to compose, validate and deploy the targeted service.
Secondly, a generic SDN controller function is available and ready to be instantiated and activated in all the network components. These controllers functions can be organised hierarchically at different levels to manage the control plane logic and different levels of locality.

Figure 1 depicts a set of generic networks nodes commonly referred to as "white boxes" carrying distributed and differentiated network services modeled with components. These network services components are composed of a set of elementary network functions assembled using a graph modeling approach and Petri nets models as described in [12]. Petri nets modeling provides the necessary tools to check the consistency of the adopted models. A communication channel carries management and control messages between the orchestrator and the network nodes.

1) SDN management plan: The deployment of these components is ensured on demand by the orchestration function through the control & management channel. The management channel, represented by a yellow arrow in figure 1, is used to set up new components on nodes corresponding to different data plane functions or to update the current components running on the data plane. The orchestrator function is in charge of network functions deployment decisions based on human requirements or based on some automatic processes. It is also in charge of the effective deployment and of the update of the network components. The network service component can be completely updated, partially updated, i.e. some network elements in the graph, or completely or partially removed and modified.

2) SDN control plane: The SDN control plane channel is encapsulated in the communication channel like the management channel. It is represented by the green arrows which connect the upper controller and the local controllers integrated in the different components nodes. The SDN control plane is also distributed between the upper controllers and the local controllers connected to the elements of components which compose the data plane. Upper controllers and local controllers carry the smart functions of the data plane. These functions are distributed between the local and upper controllers to manage hierarchically different levels of locality.

B. The distributed controller function

To implement our distributed SDN controller function we use the programmable controller for unified management of virtualized network infrastructures proposed in [14]. The role

[1]IOvisor Project. https://www.iovisor.org/.

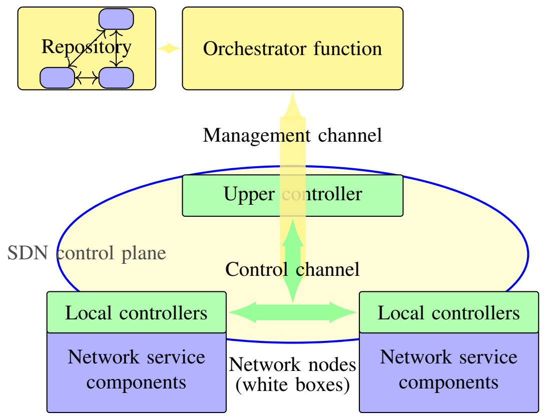

Fig. 1: Extended SDN control & management plane principles

of this controller is to expose heterogeneous resources in a unified way to the control and management entities. It is also able to provide computation resources to manage local or global configuration decisions. In the reference paper, the concept is illustrated in the context of a networking usecase based on the Click software router environment [13]. We reuse this Click controller element as a SDN controller in our own design. This controller can be inserted in the data plane and triggered through packets arrivals as other Click elements or, it can operate autonomously and independently of the packets sequences. It is able to manipulate elements handlers making read and write functions on individual Click elements or on sets of elements like Click components. It also integrate a simple syntax langage to describe computing operations executed using read parameters.

Thus, this flexible controller function can be used for our SDN control plane in a distributed or in a hierarchical way. We however extend its functionalities in programmability and make it able to connect and exchange with other controllers to distribute analysis and decision functions.

IV. A DISTRIBUTED MONITORING USE CASE

A. *Use case motivations*

The architectural concepts of the proposed extended SDN control plane are illustrated through a distributed monitoring function as it presents the following characteristics:

- A monitoring function is an elementary subset of any network service function. It is necessary in the design of a network service. It must be available on any node.
- A monitoring function can involve some intelligence at the local node level. Such intelligence cannot be managed with standard Openflow switches.
- Monitoring functions can be more or less complex depending on operator requirements and context scenarios and can be designed as context aware and upgraded dynamically, on demand.
- A monitoring function can generate a lot of control traffic particularly in the case of SDN OpenFlow networks, where the monitoring is centralized and is done by pooling.
- A monitoring function can involve some locality constraints for instance if a high responsiveness is required.
- A monitoring function can involve a lot of ressources when a sophisticated analysis is required. It justifies the benefit of dynamically adjusting the use of computing resources.

B. *Use case description*

The general principle of the use case is as follows. The goal is to detect an elephant flow, which consumes too much bandwidth, at the expense of other flows; then to take corrective measures if necessary at the node that detects this flow, usually a network entry point.

In order to optimize CPU resources, we only integrate at first a basic monitoring function in the network nodes. This function is generic and has little impact on resources and is distributed on all nodes of the network in the same way (step 1 in figure 2). This basic monitoring function is only able of detecting, at wire speed, a large, increasing variation in traffic on one of the nodes via a local controller and a global counter shown in figure 3.

If an abnormal increase in traffic is detected by a controller, this latter sends an alert to the higher-level controller, which then decides to install, on-the-fly, a more elaborated monitoring function, taking into account other possible criteria such as the position of the node in the network graph. For example, if it is an ingress node, the more complex monitoring configuration would then be able to perform a fine-grained analysis and identify the abnormal flow at the origin of the anomaly (step 2 in figure 2). The identification is achieved through an enhanced monitoring service, which can identify IP sessions and calculate their average throughput. This computation can be done in particular through additional controllers, i.e. local flows controllers that will interact with the first global flow controller in figure 4.

When detecting an anomaly, a corrective action is implemented by the local controller (step 3 in figure 2). It may consist in activating and configuring a traffic shaper that is included in the enriched configuration and that will limit the rate of the elephant flow.

A threshold-based mechanism on the global traffic volume allows the return to the initial monitoring configuration without risks of oscillations.

We illustrate by this use case the following properties that cannot be realized in an Openflow context for the already mentioned reasons:

- upgrade, without interruptions, of a generic configuration,
- resources' optimization on nodes,
- intelligent processing and local analysis of flows,
- distribution of the analysis on several elementary controllers that interact and organize in a distributed or hierarchical way the control plane,
- differentiated distribution of functions according to nodes and needs,

- complexification or simplification of processing at a node level as required.

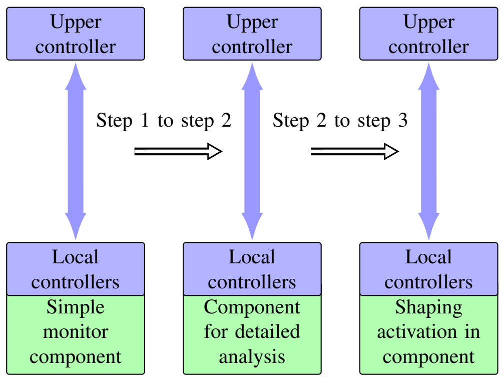

Fig. 2: Monitoring use case steps

V. Implementation of the use case

A. Click

The Click Modular Router [13] solution consists in launching a "Click" engine (program), in partial or total replacement of the standard linux kernel network of a physical or virtual machine. The engine provides a library of "elements" that can be assembled through a configuration script to compose a directed graph datapath, representing a particular service.

Packets travel along oriented edges between vertices (elements) in which specific processing is applied (classification, rewriting ...). The class of an element specifies a fixed or open number of input/output ports and its behavior (function).

The generic syntax is:
$(element_a)[output_x] \; - \; packets \; \rightarrow \; [input_y](element_b)$.
Packets are processed, following the functions coded in each element possibly parametrized through a configuration string. At the border, 'FromDevice' & 'ToDevice' elements achieve input/output from/to network interfaces. Besides, elements expose "handlers" (interfaces) for exchanging with each other or with a remote manager, at run time.

The Click engine itself exposes "global handlers" for controlling the overall operations. In particular, one of them allows a "hot-reconfiguration" meaning a fast reloading on the fly of a whole new graph.

At design time, the Click router is made from standard and customized sources (C++ classes), allowing to extend the library of elements. Click can be built for running either as a user application which is more convenient for testing/debugging experimental developments, or as a kernel module when better performances are expected.

A configuration script is given as an argument when launching the Click router. It comprises the definitions of the graph and initial parametrization of its elements. Basically, standard elements accomplish simple though efficient operations. Smart processing can be specified through classification rules for example, to dispatch different flows onto different edges.

B. A new controller element in Click to control the dataplane

For the needs of our demonstrator and to illustrate the proposed concepts, we extended the functionality of the Click controller agent introduced in [14]. This element offers a programmable automation mechanism where some actions can be performed, as a result of a logical/arithmetical combination of global or local element handlers, following a control expression written in an ad hoc language. This expression is evaluated periodically, upon packet event or on demand. If need be, actions are started by writing to arbitrary handlers. An External Call element exposes a 'command' handler informing about a job to be done when active. It can be polled by the upper controller which could in turn achieve a remote feedback. This logic is used, in our experiment to reconfigure on the fly the Click engine.

Beyond the basic logic functions of handlers composition, the main points of our extensions are:

- The ability to maintain variable values in memory. These variables make it possible, for example, to compute the Infinite Impulse Response (IIR) filter which allows us to smooth the traffic, to perform a traffic rate measurement.
- The possibility to read and write handlers of other controllers so as to be able to compose logically or arithmetically the results coming from several controllers attached to different Click components.

Our controller can be programmed to perform the required operations by simple configuration like any Click element. Configuring a controller consists in configuring it with a control expression that describes the operations to be performed such as reading or writing handlers of Click elements, composing the values of these handlers to produce new results to trigger actions either directly at the local level, or at a more global level through the external element (cf. figure 3 for more details).

C. Implementation

Figure 3 shows how the basic monitoring function is implemented in Click (i.e. first step of the process illustrated in figure 2). The incoming stream is retrieved by the Click From Device element and goes through a Click classifier element that allows to collect the data streams on which the monitoring applies, for example TCP or UDP streams. The flow of interest then pass through a counter element and then through the queue and output interface elements.

A local controller allows reading the global counter and smoothing the extracted values. If the average value exceeds a fixed threshold, the controller activates the "external element" which allows raising a flag. Once this flag detected, the central controller downloads via the communication channel a more sophisticated Click monitoring configuration allowing a detailed analysis on flows.

Figure 4 shows how the more sophisticated monitoring function is designed in Click, to allow a detailed flows analysis. The yellow elements are those that are downloaded and added to the previous configuration. We develop the

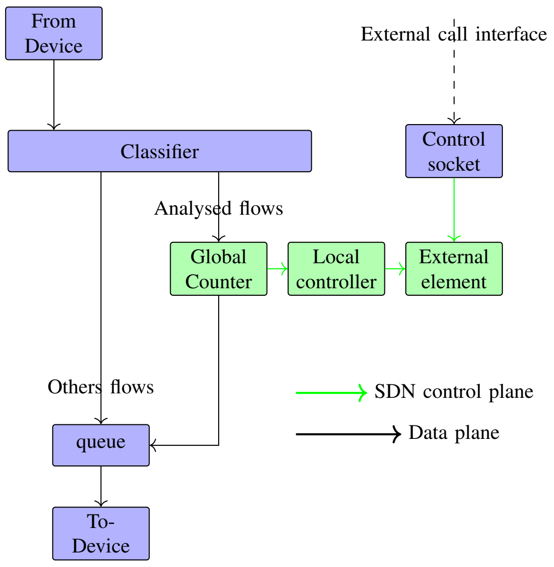

Fig. 3: Minimal monitoring Click component

Contrack element. It allows to identify and separate TCP or UDP sessions, on the different outputs. It includes variables allowing to build a finite state machine and the identification and the tracking of TCP or UDP sessions. On each output of the Contrack element, we find a chain of elements composed of a counting element, a queue element and a shaper. This chain is used for measuring the session flow bit rate on each output. The local flow controller of each visible chain in the diagram is used to measure the bit rate of the session flow and to compare it to the global flow measured by the global flow controller. Both controllers work in coordination to activate the shaper if needed by a simple configuration of the shaper element in step 3 of figure 2. The configuration of the shaper is therefore done locally.

VI. Implementation results & feedback

Figure 5 illustrates the proof of concept composed of three virtual machines running on one physical machine. The VM2 node carries the simple or complex monitoring function that applies to a route flow between two input/output interfaces of the virtual machine. A traffic source and a receiver on the VM1 and VM3 virtual machines allow traffic scenarios to be performed. The global controller is set up on the physical machine and is in charge of downloading the complex monitoring configuration. It is a simple Python script daemon. The traffic generator successively generates over time a sequence of UDP sessions of variable time durations and bit rates. When an elephant flow is generated, there is a large growth of traffic which triggers steps 2 and 3 of figure 2.
A test sequence is plotted in figure 6. It shows the progress of the test sequence that we performed with the incoming and outgoing flows. The sum of the incoming flows is drawn in blue, the outgoing flows in red. The incoming elephant flow is in green, the outcoming flow is in orange. It can be seen that the elephant flow is limited as soon as it appears in order to

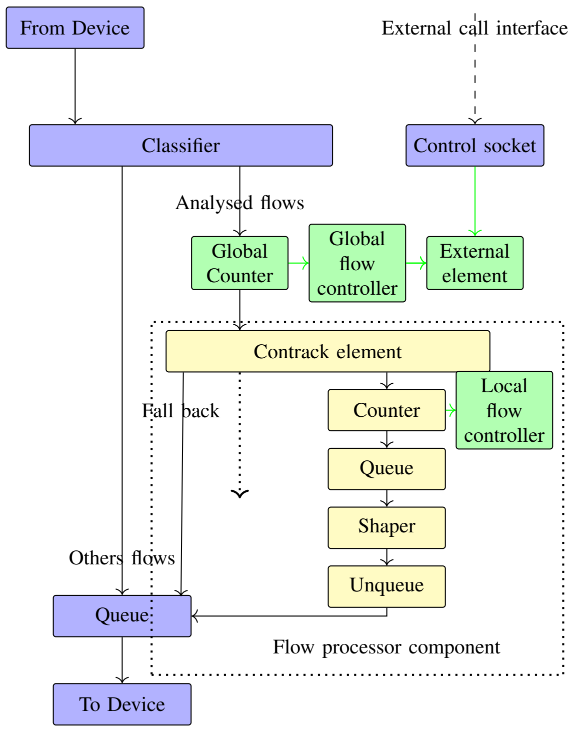

Fig. 4: Complex monitoring Click component

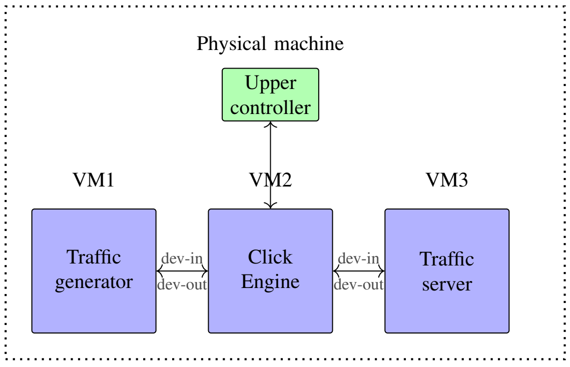

Fig. 5: Proof of concept

maintain the overall traffic in a planned gauge. The transition occurs at around $t = 50s$. We can see just at this time a short loss of packets of around 40 ms related to the upgrade on the fly of the monitoring component. It shows that the upgrade is not fully transparent and impacts the traffic. Indeed in Click the upgrade of Click components is done through a complete replacement of the component by the new component. It would be necessary to implement much more fine-tuned upgrade mechanisms acting separately on part of the Click component graphs to expect an update with limited impact. Nevertheless the loss of packets remains moderate and short.

VII. Conclusion

In this paper, we demonstrate the feasibility of an extended and flexible SDN control plane that overcomes the limitations

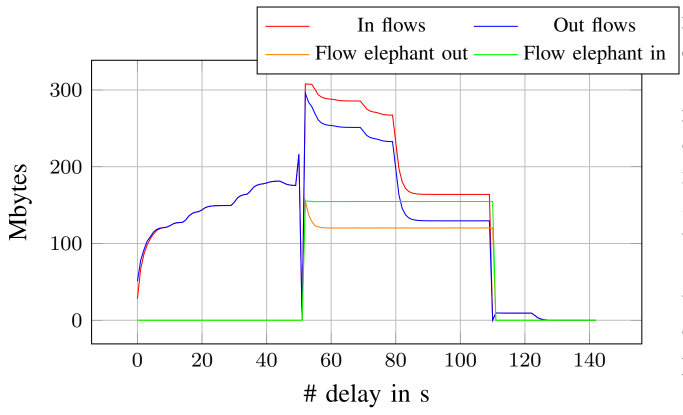

Fig. 6: Experimental results

of the openflow approach. Such limitations are as follows: Firstly, the low programmability of Openflow switches which prevents sophisticated local processing. Secondly, The strong centralization that is inherent in the Openflow architecture limits the distribution possibilities. We base this concept on several design principles:

- A white box approach of generic programmable switches.
- A disaggregation of network services into elementary components assembled via templates that model the service using a graph framework.
- A service execution engine hosted on all nodes.
- The introduction of a controller function that takes the form of an intelligent and programmable agent that can be flexibly distributed in network service components.

We then illustrate this concept on an adaptive monitoring use case with the objective of detecting and limiting elephant flows. We are therefore relying on the Click architecture for which we have developed the controller function inside the framework of a Click element. This implementation allows us to demonstrate the feasibility of the proposed concept. It also allows us to highlight, limitations inherent to the Click environment:
Firstly, regarding current Click implementation, actual data links are statically defined. Therefore, only a reasonable number of flows can be considered. In contrast, the logic inside a given element has ”no limit” in terms of complexity since this is pure software. For example, it can implement an open list of session/flow contexts, each with a state-machine. We developed such an element: Contrack implementing the basis of a statefull firewall. Therefore, two axes appear for implementing a complex logic with the Click technology:

- Gather all the logic in rich and specific multi-function elements, which can be inconsistent with the intent of the Click approach.
- Distribute the algorithm onto several more generic elements which require to find a way to keep the contextual knowledge between them, using meta-data for example.

Secondly, the limitations of the hot upgrade mechanism of Click components that causes packet loss. It would probably require a differentiated and fine-tuned mechanism for updating components in order to reduce losses.

However, the proposed architecture allows flexible and reconfigurable network services to be implemented, distributed or centralized according to requirements. It can also be implemented to create the primitive OpenFlow scenario as a reference. In the following part of this work, we want to complete the notion of controller that we presented by specifying in detail the scope of the functionalities necessary to ensure a satisfactory programmability and modularity in order to be able to centralize or distribute the SDN control plane according to the requirements of network services.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: Enabling innovation in campus networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[2] S. Schmid and J. Suomela, “Exploiting locality in distributed sdn control,” in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN ’13. New York, NY, USA: ACM, 2013, pp. 121–126.

[3] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, “Towards an elastic distributed sdn controller,” *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 7–12, Aug. 2013.

[4] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O’Connor, P. Radoslavov, W. Snow, and G. Parulkar, “Onos: Towards an open, distributed sdn os,” in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN ’14. New York, NY, USA: ACM, 2014, pp. 1–6.

[5] S. Hassas Yeganeh and Y. Ganjali, “Kandoo: A framework for efficient and scalable offloading of control applications,” in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN ’12. New York, NY, USA: ACM, 2012, pp. 19–24.

[6] Y. Fu, J. Bi, K. Gao, Z. Chen, J. Wu, and B. Hao, “Orion: A hybrid hierarchical control plane of software-defined networking for large-scale networks,” in *2014 IEEE 22nd International Conference on Network Protocols*, Oct 2014, pp. 569–576.

[7] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, “Openstate: Programming platform-independent stateful openflow applications inside the switch,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 44–51, Apr. 2014.

[8] Intel, “Fd.io - vector packet processing white paper,” INTEL, Tech. Rep., 2017.

[9] S. Choi, X. Long, M. Shahbaz, S. Booth, A. Keep, J. Marshall, and C. Kim, “Pvpp: A programmable vector packet processor,” in *Proceedings of the Symposium on SDN Research*, ser. SOSR ’17. New York, NY, USA: ACM, 2017, pp. 197–198.

[10] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, “P4: Programming protocol-independent packet processors,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[11] S. Jouet and D. P. Pezaros, “Bpfabric: Data plane programmability for software defined networks,” in *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, May 2017, pp. 38–48.

[12] J. M. Sanner, M. Ouzzif, and Y. Hadjadj-Aoul, “Dices: A dynamic adaptive service-driven sdn architecture,” in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, April 2015, pp. 1–5.

[13] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, “The click modular router,” *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, Aug. 2000.

[14] M. Yassin, K. Guillouard, M. Ouzzif, R. Picard, and D. Aluze, “A programmable controller for unified management of virtualized network infrastructures,” in *2017 IEEE Symposium on Computers and Communications (ISCC)*, July 2017, pp. 1344–1351.