Distributed SDN Control: Survey, Taxonomy and Challenges

Fetia Bannour, Sami Souihi and Abdelhamid Mellouk
LiSSi/TincNetwork Research Team
University of Paris-Est Créteil (UPEC), France
(fetia.bannour, sami.souihi, mellouk)@u-pec.fr

***Abstract*—As opposed to the decentralized control logic underpinning the devising of the Internet as a complex bundle of box-centric protocols and vertically-integrated solutions, the SDN paradigm advocates the separation of the control logic from hardware and its centralization in software-based controllers. These key tenets offer new opportunities to introduce innovative applications and incorporate automatic and adaptive control aspects, thereby easing network management and guaranteeing the user's QoE. Despite the excitement, SDN adoption raises many challenges including the scalability and reliability issues of centralized designs that can be addressed with the physical decentralization of the control plane. However, such physically distributed, but logically centralized systems bring an additional set of challenges. This paper presents a survey on SDN with a special focus on the distributed SDN control. Besides reviewing the SDN concept and studying the SDN architecture as compared to the classical one, the main contribution of this survey is a detailed analysis of state-of-the-art distributed SDN controller platforms which assesses their advantages and drawbacks and classifies them in novel ways (physical and logical classifications) in order to provide useful guidelines for SDN research and deployment initiatives. A thorough discussion on the major challenges of distributed SDN control is also provided along with some insights into emerging and future trends in that area.**

***Index Terms*—Software-Defined Networking (SDN), Distributed Control, Network Management, Quality of Experience (QoE), Adaptive and Automatic control approaches, Programmable Networks.**

I. Introduction

THE unprecedented growth in demands and data traffic, the emergence of network virtualization along with the ever-expanding use of mobile equipment in the modern network environment have highlighted major problems that are basically inherent to the Internet's conventional architecture. That made the task of managing and controlling the information coming from a growing number of connected devices increasingly complex and specialized.

Indeed, the traditional networking infrastructure is considered as highly rigid and static as it was initially conceived for a particular type of traffic, namely monotonous text-based contents, which makes it poorly suited to today's interactive and dynamic multimedia streams generated by increasingly-demanding users. Along with multimedia trends, the recent emergence of the Internet of Things (IoT) has allowed for the creation of new advanced services with more stringent communication requirements in order to support its innovative use cases. In particular, e-health is a typical IoT use case where the health-care services delivered to remote patients (e.g. diagnosis, surgery, medical records) are highly intolerant of delay, quality and privacy. Such sensitive data and life-critical traffic are hardly supported by traditional networks.

Furthermore, in the traditional architecture where the control logic is purely distributed and localized, solving a specific networking problem or adjusting a particular network policy requires acting separately on the affected devices and manually changing their configuration. In this context, the current growth in devices and data has exacerbated scalability concerns by making such human interventions and network operations harder and more error-prone.

Altogether, it has become particularly challenging for today's networks to deliver the required level of Quality of Service (QoS), let alone the Quality of Experience (QoE) that introduces additional user-centric requirements. To be more specific, relying solely on the traditional QoS that is based on technical performance parameters (e.g. bandwidth and latency) turns out to be insufficient for today's advanced and expanding networks. Additionally, meeting this growing number of performance metrics is a complex optimization task that can be treated as an NP-complete problem. Alternatively, network operators are increasingly realizing that the end-user's overall experience and subjective perception of the delivered services are as important as QoS-based mechanisms. As a result, current trends in network management are heading towards this new concept commonly referred to as the QoE to represent the overall quality of a network service from an end-user perspective.

That said, this huge gap between, on the one hand, the advances achieved in both computer and software technologies and on the other, the traditional non-evolving and *hard to manage* [1] underlying network infrastructure supporting these changes has stressed the need for an automated networking platform [2] that facilitates network operations and matches the IoT needs. In this context, several research strategies have been proposed to integrate automatic and adaptive approaches into the current infrastructure for the purpose of meeting the challenges of scalability, reliability and availability for real-time traffic, and therefore guaranteeing the user's QoE.

While radical alternatives argue that a brand-new network architecture should be built from scratch by breaking with the conventional network architecture and bringing fundamental changes to keep up with current and future requirements, other realistic alternatives are appreciated for introducing

slight changes tailored to specific needs and for making a gradual network architecture transition without causing costly disruptions to existing network operations.

In particular, the early Overlay Network alternative introduces an application layer overlay on the top of the conventional routing substrate to facilitate the implementation of new network control approaches. However, the obvious disadvantage of Overlay Networks is that they depend on several aspects (e.g. selected overlay nodes) to achieve the required performance. Besides, such networks can be criticized for compounding the complexity of existing networks due to the additional virtual layers.

On the other hand, the recent Software-Defined Networking (SDN) paradigm [3] offers the possibility to program the network and thus facilitates the introduction of automatic and adaptive control approaches by separating hardware (data plane) and software (control plane) enabling their independent evolution. SDN aims for the centralization of the network control, offering an improved visibility and a better flexibility to manage the network and optimize its performance. When compared to the Overlay Network alternative, SDN has the ability to control the entire network not only a selected set of nodes and to use a public network for transporting data. Besides, SDN spares network operators the tedious task of temporarily creating the appropriate overlay network for a specific use case. Instead, it provides an inherent programmatic framework for hosting control and security applications that are developed in a centralized way while taking into consideration the IoT requirements [4] to guarantee the user's QoE.

Along with the excitement, there have been several concerns and questions regarding the widespread adoption of SDN networks. For instance, research studies on the feasibility of the SDN deployment have revealed that the physical centralization of the control plane in a single programmable software component, called the controller, is constrained by several limitations in terms of scalability, availability, reliability, etc. Gradually, it became inevitable to think about the control plane as a distributed system [5], where several SDN controllers are in charge of handling the whole network, while maintaining a logically centralized network view.

In that respect, networking communities argued about the best way to implement distributed SDN architectures while taking into account the new challenges brought by such distributed systems. Consequently, several SDN solutions have been explored and many SDN projects have emerged. Each proposed SDN controller platform adopted a specific architectural design approach based on various factors such as the aspects of interest, the performance goals, the deployed SDN use case, and also the trade-offs involved in the presence of multiple conflicting challenges.

Despite that great interest in SDN, its deployment in the industrial context is still in its relative early stages. There might be indeed a long road ahead before technology matures and standardization efforts pay off so that the full potential of SDN can be achieved. At this point, we underline the importance of conducting a serious analysis of the proposed SDN solutions in envisioning the potential trends that may drive future research in this field.

Main contributions of this survey

Prior surveys [1, 6, 7, 8] have covered different aspects of the SDN paradigm. In particular, surveys published in IEEE CST over the last few years elaborated on various topics within the SDN scope such as the concept, benefits and historical roots [9, 10], the architecture elements and the design challenges [9, 10, 11, 12], the SDN programming languages [13], the virtualization of SDN networks using hypervisors [14], the security challenge in SDN [15], the fault management challenge in SDN [16] and the application of SDN in wireless networks [17]. Despite reviewing the distributed SDN control topic in some specific sections (e.g. the future perspective section), none of these surveys has, to the best of our knowledge, particularly focused on covering the various aspects of the decentralization problem in SDN.

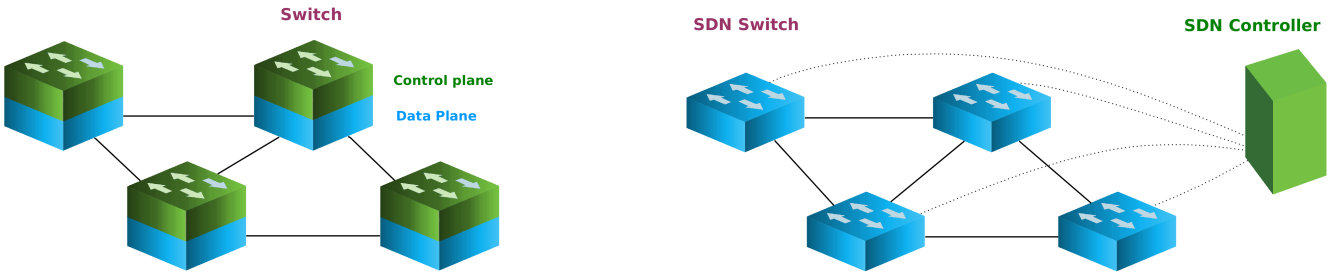
While the decentralized SDN control may be implemented using the existing distributed SDN controllers, their great number along with their particular pros and cons made the choice extremely difficult for those who attempted to adopt a distributed SDN architecture in the context of large-scale deployments. In order to assist and promote recent initiatives to put into practice the SDN paradigm, this survey proposes original classifications that make comparisons between the broad range of SDN controller platform solutions with respect to various scalability, reliability and performance criteria.

Outline

In this paper, we present a survey on distributed control in Software-Defined Networking. In Section II, we start by exposing the promises and solutions offered by SDN as compared to conventional networking. Then, we elaborate on the fundamental elements of the SDN architecture. In subsequent sections, we expand our knowledge of the different approaches to SDN by exploring the wide variety of existing SDN controller platforms. In doing so, we intend to place a special emphasis on distributed SDN solutions and classify them in two different ways: In Section III, we propose a physical classification of SDN control plane architectures into centralized and distributed (Flat or Hierarchical) in order to highlight the SDN performance, scalability and reliability challenges. In Section IV, we put forward a logical classification of distributed SDN control plane architectures into logically centralized and logically distributed while tackling the associated consistency and knowledge dissemination issues. Finally, Section V discusses the emerging challenges, opportunities and trends facing the distributed control in SDNs.

II. SDN architecture

Over the last few years, the need for a new approach to networking has been expressed to overcome the many issues associated with current networks. In particular, the main vision of the SDN approach is to simplify networking operations, optimize network management and introduce innovation and flexibility as compared to legacy networking architectures.
In this context and in line with the vision of Kim *et al.* [18], four key reasons for the problems encountered in the management of existing networks can be identified:



(a) Traditional architecture

(b) SDN architecture

Fig. 1: Conventional Networking Versus Software-Defined Networking

(i) *Complex and low-level Network configuration*
Network configuration is a complex distributed task where each device is typically configured in a low-level vendor-specific manner. Additionally, the rapid growth of the network together with the changing networking conditions have resulted in network operators constantly performing manual changes to network configurations, thereby compounding the complexity of the configuration process and introducing additional configuration errors.

(ii) *Dynamic Network State*
Networks are growing dramatically in size, complexity and consequently in dynamicity. Furthermore, with the rise of mobile computing trends as well as the advent of network virtualization [19] and cloud computing [20, 21], the networking environment becomes even more dynamic as hosts are continually moving, arriving and departing due to the flexibility offered by VM migration, and thus making traffic patterns and network conditions change in a more rapid and significant way.

(iii) *Exposed Complexity*
In today's large-scale networks, network management tasks are challenged by the high complexity exposed by distributed low-level network configuration interfaces. That complexity is mainly generated by the tight coupling between the management, control, and data planes, where many control and management features are implemented in hardware.

(iv) *Heterogeneous Network Devices*
Current networks are comprised of a large number of heterogeneous network devices including routers, switches and a wide variety of specialized middle-boxes. Each of these appliances has its own proprietary configuration tools and operates according to specific protocols, therefore increasing both complexity and inefficiency in network management.

All that said, network management is becoming more difficult and challenging given that the static and inflexible architecture of legacy networks is ill-suited to cope with today's increasingly dynamic networking trends, and to meet the QoE requirements of modern users. This fact has fueled the need for the enforcement of complex and high-level policies to adapt to current networking environments, and for the automation of network operations to reduce the tedious workload of low-level device configuration tasks.

In this sense, and to deliver the goals of easing network management in real networks, operators have considered running dynamic scripts as a way to automate network configuration settings before realizing the limitations of such approaches which led to misconfiguration issues. It is, however, worth noting, that recent approaches to scripting configurations and network automation are becoming relevant [22].

The SDN initiative led by the Open Networking Foundation (ONF) [23], on the other hand, proposes a new open architecture to address current networking challenges with the potential to facilitate the automation of network configurations , and better yet, fully program the network. Unlike the conventional distributed network architecture (Figure 1(a)) where network devices are closed and vertically-integrated bundling software with hardware, the SDN architecture (Figure 1(b)) raises the level of abstraction by separating the network data and control planes. That way, network devices become simple forwarding switches whereas all the control logic is centralized in software controllers providing a flexible programming framework for the development of specialized applications and for the deployment of new services.

Such aspects of SDN are believed to simplify and improve network management by offering the possibility to innovate, customize behaviors and control the network according to high-level policies expressed as centralized programs, therefore bypassing the complexity of low-level network details and overcoming the fundamental architectural problems raised in (i) and (iii). Added to these features is the ability of SDN to easily cope with the heterogeneity of the underlying infrastructure (outlined in (iv)) thanks to the SDN Southbound interface abstraction.

More detailed information on the SDN-based architecture which is split vertically into three layers (see Figure 2) is provided in the next subsections:

A. SDN Data plane

The data plane, also known as the forwarding plane, consists of a distributed set of forwarding network elements (mainly *switches*) in charge of forwarding packets. In the context of SDN, the control-to-data plane separation feature requires the data plane to be remotely accessible for software-based control via an open vendor-agnostic Southbound interface.

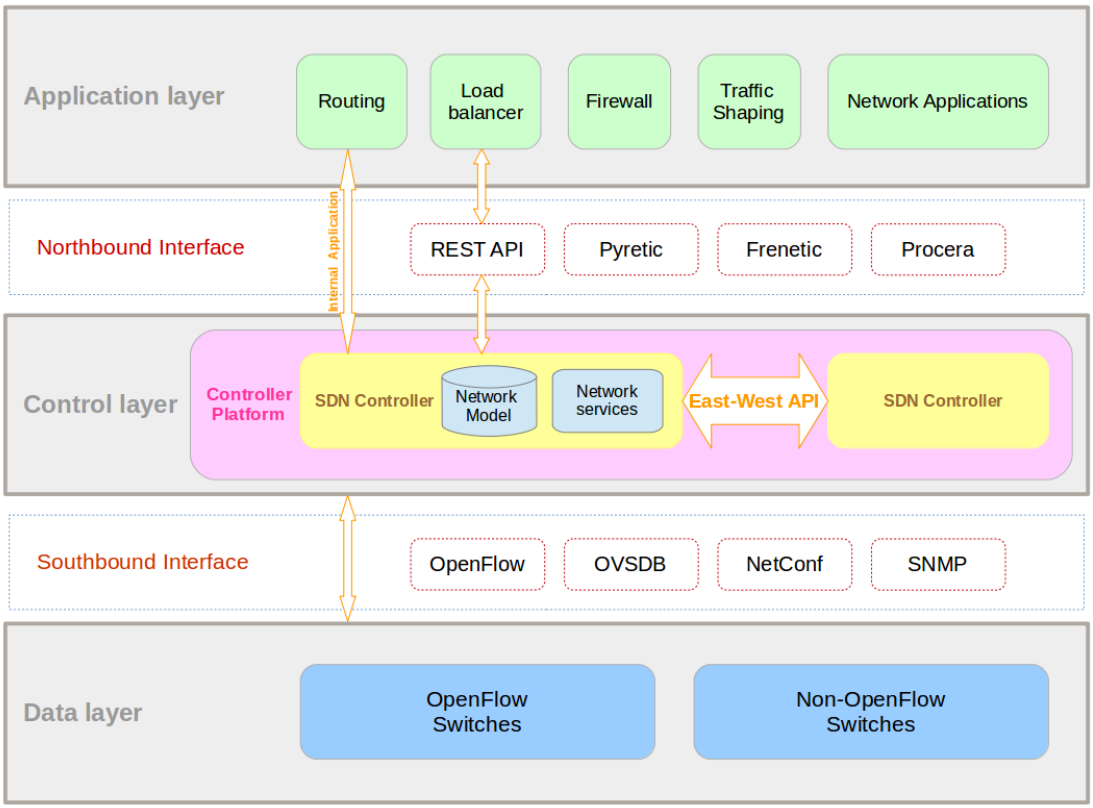


Fig. 2: A three-layer distributed SDN architecture

Both OpenFlow [24] and ForCES [25] are well-known candidate protocols for the Southbound interface. They both follow the basic principle of splitting the control plane and the forwarding plane in network elements and they both standardize the communication between the two planes. However, these solutions are different in many aspects, especially in terms of network architecture design.

Standardized by IETF, ForCES (Forwarding and Control Element Separation) [25] introduced the separation between the control plane and the forwarding plane. In doing so, ForCES defines two logic entities that are logically kept in the same physical device: the Control Element (CE) and the Forwarding Element (FE). However, despite being a mature standard solution, the ForCES alternative did not gain widespread adoption by major router vendors.

On the other hand, OpenFlow [24] received major attention in both the research community and the industry. Standardized by the ONF [23], it is considered as the first widely accepted communication protocol for the SDN Southbound interface. OpenFlow enables the control plane to specify in a centralized way the desired forwarding behavior of the data plane. Such traffic forwarding decisions reflect the specified network control policies and are translated by *controllers* into actual packet forwarding rules populated in the flow tables of OpenFlow *switches*.

In more specific terms, and according to the original version 1.0.0 of the standard defined in [26], an OpenFlow-enabled Switch consists of *a flow table* and an OpenFlow *secure channel* to an external OpenFlow controller. Typically, the forwarding table maintains a list of flow entries; Each flow entry comprises *match fields* containing header values to match packets against, *counters* to update when packets match for flow statistics collection purposes, and a set of *actions* to apply to matching packets.

Accordingly, all incoming packets processed by the switch are compared against the flow table where flow entries match packets based on a priority order specified by the controller. In case a matching entry is found, the flow counter is incremented and the actions associated with the specific flow entry are performed on the incoming packet belonging to that flow. According to the OpenFlow specification [26], these actions may include forwarding a packet out on a specific port, dropping the packet, removing or updating packet headers, etc. If no match is found in the flow table, then the unmatched packet is encapsulated and sent over the secure channel to the controller which decides on the way it should be processed. Among other possible actions, the controller may define a new flow for that packet by inserting new flow table entries.

Despite the advantages linked to the flexibility and innovation brought to network management, OpenFlow [24] suffers from scalability and performance issues that stem mainly from pushing all network intelligence and control logic to the centralized OpenFlow controller, thus restricting the task of OpenFlow switches to a dumb execution of forwarding actions.

To circumvent these limitations, several approaches [27, 28, 29, 30] suggest revisiting the delegation of control between

the controller and switches and introducing new SDN switch Southbound interfaces.

Notably, DevoFlow [28] claims to minimize switch-to-controller interactions by introducing new control mechanisms inside switches. That way, switches can make local control decisions when handling frequent events, without involving controllers whose primary tasks will be limited to keeping centralized control over far fewer significant events that require network-wide visibility. Despite introducing innovative ideas, the DevoFlow alternative has been mainly criticized for imposing major modifications to switch designs [31].

On the other hand, *stateful* approaches [29, 32, 33], as opposed to the original *stateless* OpenFlow abstraction, motivate the need to delegate some stateful control functions back to switches in order to offload the SDN controller. These approaches face the challenging dilemma of programming stateful devices (evolving the data plane) while retaining the simplicity, generality and vendor-agnostic features offered by the OpenFlow abstraction. In particular, the OpenState proposal [29] is a stateful platform-independent data plane extension of the current OpenFlow match/action abstraction supporting a finite-state machine (FSM) programming model called Mealy Machine in addition to the flow programming model adopted by OpenFlow. That model is implemented inside the OpenFlow switches using additional *state tables* in order to reduce the reliance on remote controllers for applications involving local states like MAC learning operations and port-knocking on a firewall.

Despite having the advantage of building on the adaptation activity of the OpenFlow standard and leveraging its evolution using the (stateful) extensions provided by recent versions (version 1.3 and 1.4), OpenState faces important challenges regarding the implementation of a stateful extension for programming the forwarding behaviour inside switches while following an OpenFlow-like implementation approach. The feasibility of the hardware implementation of OpenState has been addressed in [34]. Finally, the same authors extended their work into a more general and expressive abstraction of OpenState called OPP [35] which supports a full extended finite-state machine (XFSM) model, thereby enabling a broader range of applications and complex stateful flow processing operations.

In the same spirit, the approach presented in [36] explored delegating some parts of the controller functions involving packet generation tasks to OpenFlow switches in order to address both switch and controller scalability issues. The InSP API was introduced as a generic API that extends OpenFlow to allow for the programming of autonomous packet generation operations inside the switches such as ARP and ICMP handling. The proposed OpenFlow-like abstractions include an *InSP Instruction* for specifying the actions that the switch should apply to a packet being generated after a triggering event and a *Packet Template Table (PTE)* for storing the content of any packet generated by the switch.

According to [36], the InSP function, like any particular offloading function, faces the challenging issue of finding the relevant positioning with respect to the broad design space for delegation of control to SDN switches. In their opinion, a good approach to conceiving (eventually standardizing) a particular offloading function should involve a programming abstraction that achieves a fair compromise between viability and flexibility, far from extreme solutions that simply turn on well-known legacy protocol functions (e.g. MAC learning) or push a piece of code inside the switches [37, 38].

The authors of FOCUS [39] express the same challenges but, unlike the above proposals, they reject a performance-based design choice that requires adding new hardware primitives to OpenFlow switches in the development of the delegated control function. Instead, they promote a deployable software-based solution to be implemented in the switch's *software stack* to achieve a balanced trade-off between the flexibility and cost of the control function delegation process.

B. SDN Control plane

Regarded as the most fundamental building entity in SDN architecture, the control plane consists of a centralized software controller that is responsible for handling communications between network applications and devices through open interfaces. More specifically, SDN controllers translate the requirements of the application layer down to the underlying data plane elements and give relevant information up to SDN applications.
The SDN control layer is commonly referred to as the Network Operating System (NOS) as it supports the network control logic and provides the application layer with an abstracted view of the global network, which contains enough information to specify policies while hiding all implementation details.

Typically, the control plane is logically centralized and yet implemented as a physically distributed system for scalability and reliability reasons as discussed in Sections III and IV. In a distributed SDN control configuration, East-Westbound APIs [40] are required to enable multiple SDN controllers to communicate with each other and exchange network information. Despite the many attempts to standardize SDN protocols, there has been to date no standard for the East-West API which remains proprietary for each controller vendor. Although a number of East-Westbound communications happen only at the data-store level and do not require additional protocol specifics, it is becoming increasingly advisable to standardize that communication interface in order to provide wider interoperability between different controller technologies in different autonomous SDN networks.

On the other hand, an East-Westbound API standard requires advanced data distribution mechanisms and involves other special considerations. This brings about additional SDN challenges, some of which have been raised by the state-of-the art distributed controller platforms discussed in Sections III and IV, but have yet to be fully addressed.

C. SDN Application plane

The SDN application plane comprises SDN applications which are control programs designed to implement the network control logic and strategies. This higher-level plane interacts with the control plane via an open Northbound API. In doing so, SDN applications communicate their network

requirements to the SDN controller which translates them into Southbound-specific commands and forwarding rules dictating the behavior of the individual data plane devices. Routing, Traffic Engineering (TE), firewalls and load balancing are typical examples of common SDN applications running on top of existing controller platforms.

In the context of SDN, applications leverage the decoupling of the application logic from the network hardware along with the logical centralization of the network control, to directly express the desired goals and policies in a centralized high-level manner without being tied to the implementation and state-distribution details of the underlying networking infrastructure. Concurrently, SDN applications make use of the abstracted network view exposed through the Northbound interface to consume the network services and functions provided by the control plane according to their specific purposes.

That being said, the Northbound API implemented by SDN controllers can be regarded as a network abstraction interface to applications, easing network programmability, simplifying control and management tasks and allowing for innovation. In contrast to the Southbound API, the Northbound API is not supported by an accepted standard.
Despite the broad variety of Northbound APIs adopted by the SDN community (see Figure 2), we can classify them into two main categories:

- The first set involves simple and primitive APIs that are directly linked to the internal services of the controller platform. These implementations include:
  - Low-level ad-hoc APIs that are proprietary and tightly dependent on the controller platform. Such APIs are not considered as high-level abstractions as they allow developers to directly implement applications within the controller in a low-level manner. Deployed internally, these applications are tightly coupled with the controller and written in its native general-purpose language. NOX in C++ and POX in Python are typical examples of controllers that use their own basic sets of APIs.
  - APIs based on Web services such as the widely-used REST API. This group of programming interfaces enables independent external applications (*Clients*) to access the functions and services of the SDN controller (*Server*). These applications can be written in any programming language and are not run inside the bundle hosting the controller software. Floodlight is an example of an SDN controller that adopts an embedded Northbound API based on REST.
- The second category contains higher level APIs that rely on domain-specific programming languages such as Frenetic [41], Procera [42] and Pyretic [43] as an indirect way for applications to interact with the controller. These APIs are designed to raise the level of abstraction in order to allow for the flexible development of applications and for the specification of high-level network policies.

III. Physical classification of SDN control plane architectures

Despite the undeniable strengths of SDN, there have always been serious concerns about the ability to extend SDN to large-scale networks.

Some argue that these scalability limits are basically linked to the protocol standards being used for the implementation of SDN. OpenFlow [24] in particular, although recognized as a leading and widely-deployed SDN Southbound technology, is currently being rethought for potentially causing excessive overheads on switches (*switch bottleneck*). Scalable alternatives to the OpenFlow standard which propose to revisit the delegation of control between the controller and the switches with the aim of reducing the reliance on SDN the control plane, have been discussed in II-A.

Another entirely different approach to addressing the SDN scalability and reliability challenges, which is advocated by the present paper, is to physically distribute the SDN control plane. This has led to a first categorization of existing controller platforms into centralized and distributed architectures (see Figure 3). Please note that, in Figure 3 and Figure 4, controllers that present similar characteristics for the discussed comparison criteria are depicted in the same color.

A. Centralized SDN control

A physically-centralized control plane consisting of a single controller for the entire network is a theoretically perfect design choice in terms of simplicity. However, a single controller system may not keep up with the growth of the network. It is likely to become overwhelmed (*controller bottleneck*) while dealing with an increasing number of requests and concurrently struggling to achieve the same performance guarantees.

Obviously, a centralized SDN controller does not meet the different requirements of large-scale real-world network deployments. Data Centers and Service Provider Networks are typical examples of such large-scale networks presenting different requirements in terms of scalability and reliability.

More specifically, a *Data Center Network* involves tens of thousands of switching elements. Such a great number of forwarding elements which can grow at a fast pace is expected to generate a huge number of control events that are enough to overload a single centralized SDN controller [44, 45]. Studies conducted in [46] show important scalability implications (in terms of throughput) for centralized controller approaches. They demonstrate that multiple controllers should be used to scale the throughput of a centralized controller and meet the traffic characteristics within realistic data centers.

Unlike data centers, *Service Provider Networks* are characterized by a modest number of network nodes. However, these nodes are usually geographically distributed making the diameter of these networks very large [44]. This entails a different type of controller scalability issues for centralized controller approaches, more specifically, high latencies. In addition to latency requirements, service provider networks have large numbers of flows that may generate overhead and bandwidth issues.

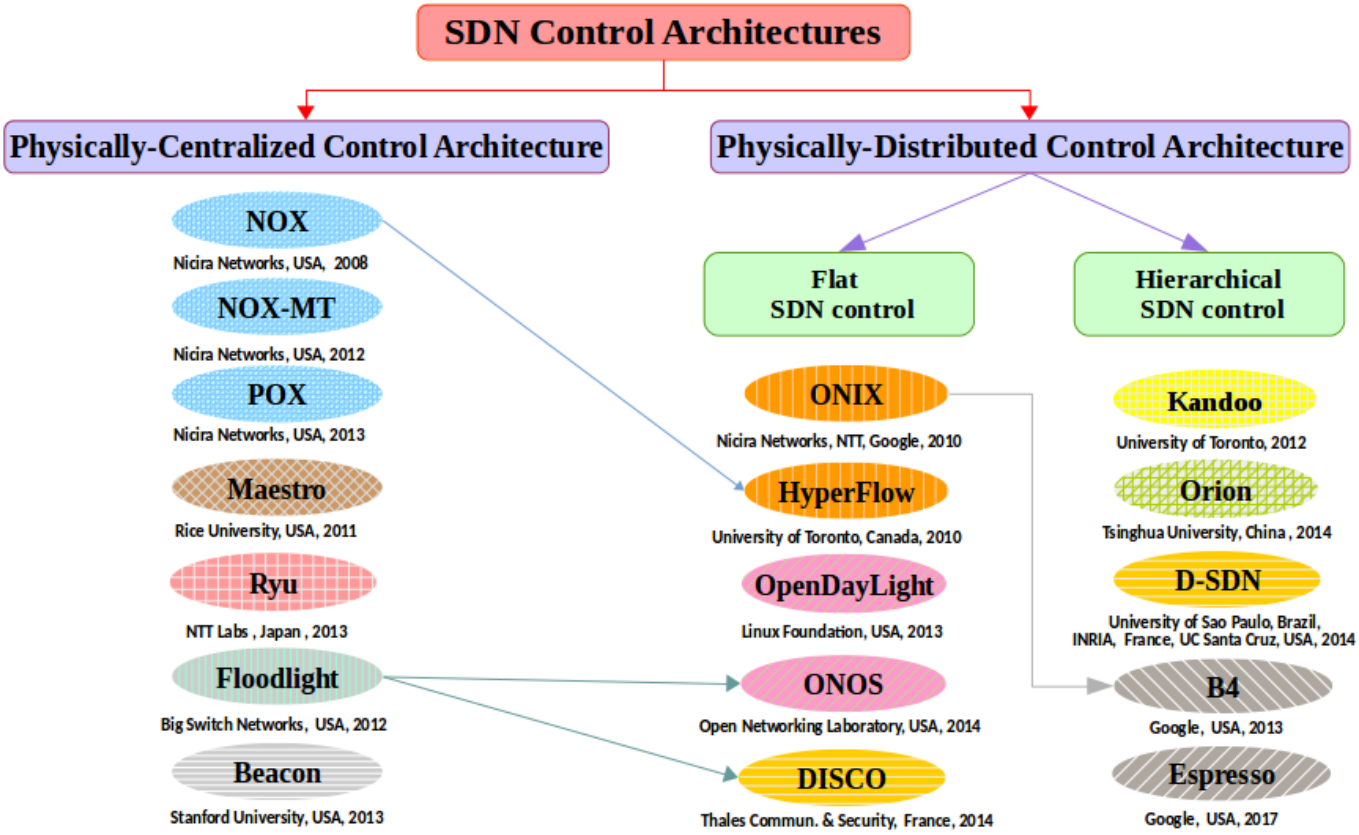


Fig. 3: Physical classification of SDN control plane architectures

In general, Wide Area Network (WAN) deployments typically impose strict resiliency requirements. In addition, they present higher propagation delays as compared to data center networks. Obviously, a centralized controller design in a SD-WAN cannot achieve the desired failure resiliency and scale-out behaviors [47]. Several studies have emphasized the need for a distributed control plane in a SD-WAN architecture: They indeed focused on placing multiple controllers on real WAN topologies to benefit both control plane latency and fault-tolerance [48, 49].

That said, the potential scalability, reliability and vulnerability concerns associated with centralized controller approaches have been further confirmed through studies [7, 50] on the behavior of state-of-the-art centralized SDN controllers such as NOX [51], Beacon [52] and Floodlight [53] in different networking environments.

In particular, NOX classic [51], the world's first-generation OpenFlow controller with an event-based programming model, is believed to be limited in terms of throughput. Indeed, it cannot handle a large number of flows, namely a rate of 30k flow initiation events per second [7, 54]. Such a flow setup throughput may sound sufficient for an enterprise network, but, it could be arguable for data-center deployments with high flow initiation rates [46]. Improved versions of NOX have been consequently developed by the same community (Nicira Networks) such as NOX-MT [55] for better performance and POX [56] for a more developer-friendly environment.

However, while none of these centralized designs is believed to meet the above scalability and reliability requirements of large-scale networks, they have gained greater prominence as they were widely used for research and educational purposes.

Additionally, Floodlight [53] which is a very popular Java-based OpenFlow controller from Big Switch Networks, suffers from serious security and resiliency issues. For instance, Dhawan *et al.* [57] have reported that the centralized SDN controller is inherently susceptible to Denial-of-Service (DoS) attacks. Another subsequent version of Floodlight, called SE-Floodlight, has therefore been released to overcome these problems by integrating security applications. However, despite the introduced security enhancements aimed at shielding the centralized controller, the latter remains a potential weakness compromising the whole network. In fact, the controller still maintains a single point of failure and bottlenecks even if its latest version is less vulnerable to malicious attacks.

On the other hand, given its obvious performance and functionality advantages, the open-source Floodlight has been extensively used to build other SDN controller platforms supporting distributed architectures such as ONOS [58] and DISCO [59].

B. Distributed SDN control

Alternatively, physically-distributed control plane architectures have received increased research attention in recent years since they appeared as a potential solution to mitigate the issues brought about by centralized SDN architectures (poor scalability, Single Point of Failure (SPOF), performance bottlenecks, etc). As a result, various SDN control plane designs have been proposed in recent literature. Yet, we discern two main categories of distributed SDN control architectures based on the physical organization of SDN controllers: A flat SDN control architecture and a hierarchical SDN control architecture (see Figure 3).

1) Flat SDN control:

The flat structure implies the horizontal partitioning of the network into multiple areas, each of which is handled by a single controller in charge of managing a subset of SDN switches. There are several advantages to organizing controllers in such a flat style, including reduced control latency and improved resiliency.

Onix [60], Hyperflow [61] and ONOS [58] are typical examples of flat physically-distributed controller platforms which are initially designed to improve control plane *scalability* through the use of multiple interconnected controllers sharing a global network-wide view and allowing for the development of centralized control applications. However, each of these contributions takes a different approach to distribute controller states and providing control plane scalability.

For example, Onix provides a good scalability through additional partitioning and aggregation mechanisms. To be more specific, Onix partitions the NIB (Network Information Base) giving each controller instance responsibility for a subset of the NIB and it aggregates by making applications reduce the fidelity of information before sharing it between other Onix instances within the cluster. Similar to Onix, each ONOS instance (composing the cluster) that is responsible for a subset of network devices holds a portion of the network view that is also represented in a graph. Different from Onix and ONOS, every controller in HyperFlow has the global network view, thus getting the illusion of control over the whole network. Yet, HyperFlow can be considered as a scalable option for specific policies in which a small number of network events affect the global network state. In that case, scalability is ensured by propagating these (less frequent) selected events through the event propagation system.

Furthermore, different mechanisms are put in place by these distributed controller platforms to meet *fault-tolerance* and *reliability* requirements in the event of failures or attacks.

Onix [60] uses different recovery mechanisms depending on the detected failures. Onix instance failure is most of the time handled by distributed coordination mechanisms among replicas whereas network element/link failures are under the full responsibility of applications developed atop Onix. Besides, Onix is assumed reliable when it comes to connectivity infrastructure failures as it can dedicate the failure recovery task to a separate management backbone that uses a multi-pathing protocol.

Likewise, Hyperflow [61] focuses on ensuring resiliency and fault tolerance as a means for achieving availability. When a controller failure is discovered by the failure detection mechanisms deployed by its publish/subscribe WheelFS [62] system, HyperFlow reconfigures the affected switches and redirects them to another nearby controller instance (from a neighbor's site). Alongside this ability to tackle component failures, HyperFlow is resilient to network partitioning thanks to the partition tolerance property of WheelFS. In fact, in the presence of a network partitioning, WheelFS partitions continue to operate independently, thus favoring availability.

Similarly, ONOS [58] considers fault-tolerance as a prerequisite for adopting SDN in Service Provider networks. ONOS's distributed control plane guards against controller instance failures by connecting, from the onset, each SDN switch to more than one SDN controller; its master controller and other backup controllers (from other domains) that may take over in the wake of master controller failures. Load balancing mechanisms are also provided to balance the mastership of switches among the controllers of the cluster for scalability purposes. Besides, ONOS incorporates additional recovery protocols, such as the Anti-Entropy protocol [63], for healing from lost updates due to such controller crashes.

Recent SDN controller platform solutions [64, 65, 66, 67, 68, 69] focused specifically on improving fault-tolerance in the distributed SDN control plane. Some of these works assumed a simplified flat design where the SDN control was centralized. However, since the main focus was placed at the fault-tolerance aspect, we believe that their ideas and their fault-tolerance approaches can be leveraged in the context of medium to large scale SDNs where the network control is physically distributed among multiple controllers.

In particular, Botelho et. al [70] developed a hybrid SDN controller architecture that combines both passive and active replication approaches for achieving control plane fault-tolerance. SMaRtLight adopts a simple Floodlight [53]-based multi-controller design following OpenFlow 1.3, where one main controller (the primary) manages all network switches, and other controller replicas monitor the primary controller and serve as backups in case it fails.

This variant of a traditional passive replication system relies on an external data store that is implemented using a modern active Replicated State Machine (RSM) built with a Paxos-like protocol (BFT-SMaRt [71]) to ensure fault-tolerance and strong consistency. This shared data store is used for storing the network and application state (the common global NIB) and also for coordinating fault detection and leader election operations between controller replicas that run a lease management algorithm.

In case of a failure of the primary controller, the elected backup controller starts reading the current state from the shared consistent data store in order to mitigate the cold-start (empty state) issue associated with traditional passive replication approaches, and thereby ensure a smoother transition to the new primary controller role.

The limited feasibility of the deployed controller fault-tolerance strategy is warranted by the limited scope of the SMaRtLight solution which is only intended for small to medium-sized SDN networks. On the other hand, in large-scale deployments, adopting a simplified Master-Slave approach, and more importantly, assuming a single main controller scheme where one controller replica must retrieve all the network state from the shared data store in failure scenarios, have major disadvantages in terms of increased latency and failover time.

Similarly, the Ravana controller platform proposal [66] addresses the issue of recovering from complete fail-stop controller crashes. It offers the abstraction of a fault-free centralized SDN controller to unmodified control applications which are relieved of the burden of handling controller failures. Accordingly, network programmers write application programs for a single main controller and the transparent master-slave Ravana protocol takes care of replicating, seamlessly and consistently, the control logic to other backup controllers for fault-tolerance.

The Ravana approach deploys enhanced Replicated State Machine (RSM) techniques that are extended with switch-side mechanisms to ensure that control messages are processed transactionally with ordered and exactly-once semantics even in the presence of failures. The three Ravana prototype components, namely the Ryu [72]-based controller runtime, the switch runtime, and the control channel interface, work cooperatively to guarantee the desired correctness and robustness properties of a fault-tolerant logically centralized SDN controller.

More specifically, when the master controller crashes, the Ravana protocol detects the failure within a short failover time and elects the standby slave controller to take over using Zookeeper [73]-like failure detection and leader election mechanisms. The new leader finishes processing any logged events in order to catch up with the failed master controller state. Then, it registers with the affected switches in the role of the new master before proceeding with normal controller operations.

2) Hierarchical SDN control:

The hierarchical SDN control architecture assumes that the network control plane is vertically partitioned into multiple levels (layers) depending on the required services. According to [74], a hierarchical organization of the control plane can improve SDN scalability and performance.

To improve *scalability*, Kandoo [31] assumes a hierarchical two-layer control structure that partitions control applications into local and global. Contrary to DevoFlow [28] and DIFANE [27], Kandoo proposes to reduce the overall stress on the control plane without the need to modify OpenFlow switches. Instead, it establishes a two-level hierarchical control plane, where frequent events occurring near the data path are handled by the bottom layer (local controllers with no interconnection running local applications) and non-local events requiring a network-wide view are handled by the top layer (a logically centralized root controller running non-local applications and managing local controllers).

Despite the obvious scalability advantages of such a control plane configuration where local controllers can scale linearly as they do not share information, Kandoo did not envision *fault-tolerance* and resiliency strategies to protect itself from potential failures and attacks in the data and control planes. Besides, from a developer perspective, Kandoo imposes some kandoo-specific conditions on the control applications developed on top of it, in such a way that makes them aware of its existence.

On the other hand, Google's B4 [75, 76], a private intra-domain software-defined WAN connecting their data centers across the planet, proposes a two-level hierarchical control framework for improving *scalability*. At the lower layer, each data-center site is handled by an Onix-based [60] SDN controller hosting local site-level control applications. These site controllers are managed by a global *SDN Gateway* that collects network information from multiple sites through site-level TE services and sends them to a logically centralized *TE server* which also operates at the upper layer of the control hierarchy. Based on an abstract topology, the latter enforces high-level TE policies that are mainly aimed at optimizing bandwidth allocation between competing applications across the different data-center sites. That being said, the TE server programs these forwarding rules at the different sites through the same gateway API. These TE entries will be installed into higher-priority switch forwarding tables alongside the standard shortest-path forwarding tables. In this context, it is worth mentioning that the *topology abstraction* which consists in abstracting each site into a *super-node* with an aggregated *super-trunk* to a remote site is key to improving the scalability of the B4 network. Indeed, this abstraction hides the details and complexity from the logically centralized TE controller, thereby allowing it to run protocols at a coarse granularity based on a global controller view and, more importantly preventing it from becoming a serious performance bottleneck.

Unlike Kandoo [31], B4 [75] deploys robust *reliability* and *fault-tolerance* mechanisms at both levels of the control hierarchy in order to enhance the B4 system availability. These mechanisms have been especially enhanced after experiencing a large-scale B4 outage. In particular, Paxos [77] is used for detecting and handling the primary controller failure within each data-center site by electing a new leader controller among a set of reachable standby instances. On the other hand, network failures at the upper layer are addressed by the logically centralized TE controller which adapts to failed or unresponsive site controllers in the bandwidth allocation process. Additionally, B4 is resilient against other failure scenarios where the upper-level TE controller encounters major problems in reaching the lower-level site controllers (e.g. TE operation/session failures). Moreover, B4 guards against the failure of the logically centralized TE controller by geographically replicating TE servers across multiple WAN sites (one master TE server and four secondary hot standbys). Finally, another fault recovery mechanism is used in case the TE controller service itself faces serious problems. That mechanism stops the TE service and enables the standard shortest-path routing mechanism as an independent service.

In the same spirit, Espresso [78] is another interesting SDN contribution that represents the latest and more challenging pillar of Google's SDN strategy. Building on the previous three layers of that strategy (the *B4* WAN [75], the *Andromeda* NFV stack and the *Jupiter* data center interconnect), *Espresso* extends the SDN approach to the peering edge of Google's network where it connects to other networks worldwide. Considered as a large-scale SDN deployment for the public Internet, Espresso, which has been in production for more than two years, routes over 22% of Google's total traffic to the Internet. More specifically, the Espresso technology allows Google to dynamically choose from where to serve content for individual users based on real-time measurements of end-to-end network connections.

To deliver unprecedented *scale-out* and efficiency, Espresso assumes a hierarchical control plane design split between *Global controllers* and *Local controllers* that perform different functions. Besides, Espresso's software programmability design principle externalizes features into software thereby exploiting commodity servers for scale.

Moreover, Espresso achieves higher availability (*reliability*) when compared to existing router-centric Internet protocols. Indeed, it supports a fail static system, where the local data plane keeps the last known good state to allow for control plane unavailability without impacting data plane and BGP peering operations. Finally, another important feature of Espresso is that it provides full *interoperability* with the rest of the Internet and the traditional heterogeneous peers.

IV. Logical classification of distributed SDN control plane architectures

Apart from the physical classification, we can categorize distributed SDN control architectures according to the way knowledge is disseminated among controller instances (the *consistency* challenge) into logically centralized and logically distributed architectures (see Figure 4). This classification has been recently adopted by [79].

A. Logically centralized SDN control

Onix and SMaRtLight:

Both Onix [60] and SMaRtLight [70] are logically centralized controller platforms that achieve controller state redundancy through state replication. But the main difference is that Onix uses a distributed data store while SMartLight uses a centralized data store for replicating the shared network state. They also deploy different techniques for sharing knowledge and maintaining a consistent network state.

Onix is a distributed control platform for large-scale production networks that stands out from previous proposals by providing a simple general-purpose API, a central NIB abstraction and standard state distribution primitives for easing the implementation of network applications.

In more specific terms, Onix uses the NIB data structure to store the global network state (in the form of a network graph) that is distributed across running Onix instances and synchronized through Onix's built-in state distribution tools according to different levels of consistency as dictated by application requirements. In fact, besides interacting with the NIB at run-time, network applications on top of Onix initially configure their own data storage and dissemination mechanisms by choosing among two data-store options already implemented by Onix in the NIB: A replicated transactional database that guarantees strong consistency at the cost of good performance for persistent but slowly-changing data (state), and a high-performance memory-only distributed hash table (DHT) for volatile data that does not require strict consistency.

While the main advantage of Onix is its programmatic framework created for the flexible development of control applications with desired trade-offs between performance and state consistency (strong/eventual), it carries the limitations of eventually consistent systems which rely on application-specific logic to detect network state inconsistencies for the eventually-consistent data and provide conflict resolution methods for handling them.

As mentioned in Section III-B1, SMaRtLight is a fault-tolerant logically centralized Master-Slave SDN controller platform, where a single controller is in charge of all network decisions. This main controller is supported by backup controller replicas that should have a synchronized network view in order to take over the network control in case of the primary failure. All controller replicas are coordinated through a shared data store that is kept fault-tolerant and strongly consistent using an implementation of Replicated State Machine (RSM).

Consistency between the master and backup controllers is guaranteed by replicating each change in the network image (NIB) of the master into the shared data store before modifying the state of the network. However, such synchronization updates generate additional time overheads and have a drastic impact on the controller's performance. To address this issue, the controllers keep a local cache (maintained by one active primary controller at any time) to avoid accessing the shared data store for read operations. By keeping the local cache and the data store consistent even in the presence of controller failures, the authors claim that their simple Master-Slave structure achieves, in the context of small to medium-sized networks, a balance between consistency and fault-tolerance while keeping performance at an acceptable level.

HyperFlow and Ravana:

Both HyperFlow [61] and Ravana [66] are logically centralized controller platforms that achieve state redundancy through event replication. Despite their similarities in building the application state, one difference is that the Ravana protocol is completely transparent to control applications while HyperFlow requires minor modifications to applications. Besides, while HyperFlow is eventually consistent favoring availability, Ravana ensures strong consistency guarantees.

More specifically, Hyperflow [61] is an extension of NOX into a distributed event-based control plane where each NOX-based controller manages a subset of OpenFlow network switches. It uses an event-propagation publish/subscribe mechanism based on the distributed WheelFS [62] file system

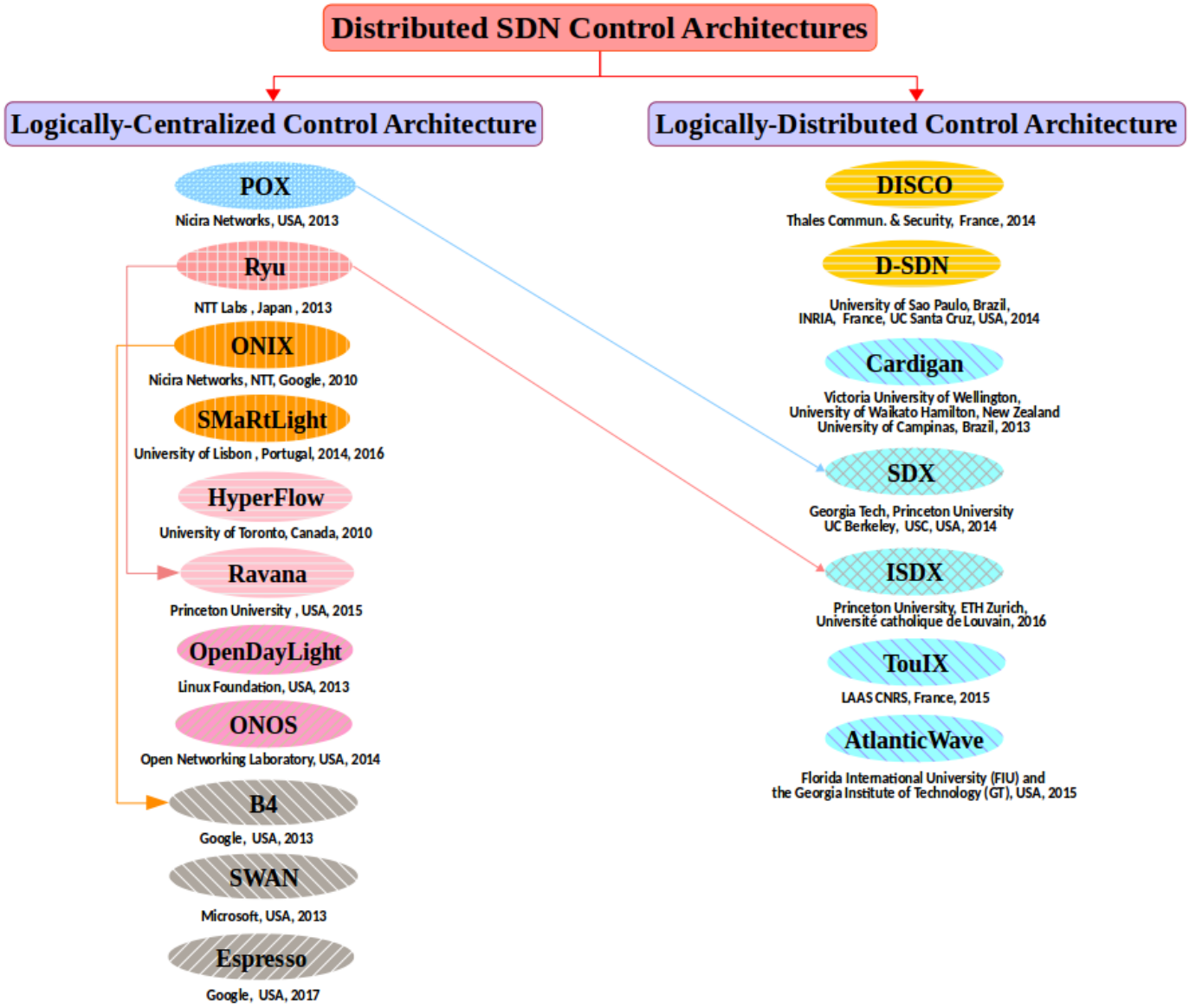


Fig. 4: Logical classification of distributed SDN control plane architectures

for propagating selected network events and maintaining the global network-wide view across controllers. Accordingly, the Hyperflow controller application instance running on top of an individual NOX controller selectively publishes relevant events that affect the network state and receives events on subscribed channels to other controllers. Then, other controllers locally replay all the published events in order to reconstruct the state and achieve the synchronization of the global view.

By this means, all controller instances make decisions locally and individually (without contacting remote controller instances): They indeed operate based on their synchronized eventually-consistent network-wide view as if they are in control of the entire network. Through this synchronization scheme, Hyperflow achieves the goal of minimizing flow setup times and also congestion, in other words, cross-site traffic required to synchronize the state among controllers. However, the potential downside of Hyperflow is related to the performance of the publish/subscribe system which can only deal with non-frequent events. Besides, HyperFlow does not guarantee a strict ordering of events and does not handle consistency problems. This makes the scope of HyperFlow restricted to applications that does not require a strict event ordering with strict consistency guarantees.

To correctly ensure the abstraction of a ”logically centralized SDN controller”, an elaborate fault-tolerant controller platform called Ravana [66] extended beyond the requirements for controller state consistency to include that for switch state consistency under controller failures.

Maintaining such strong levels of consistency in both controllers and switches in the presence of failures, requires handling the entire event-processing cycle as a transaction in accordance with the following properties: (i) events are

processed in the same total order at all controller replicas so that controller application instances would reach the same internal state, (ii) events are processed exactly-once across all the controller replicas, (iii) commands are executed exactly-once on the switches.

To achieve such design goals, Ravana follows a Replicated State Machine (RSM) approach, but extends its scope to deal with switch state consistency under failures. Indeed, while Ravana permits unmodified applications to run in a transparent fault-tolerant environment, it requires modifications to the OpenFlow protocol, and it makes changes to current switches instead of involving them in a complex consensus protocol.

To be more specific, Ravana uses a two-stage replication protocol that separates the reliable logging of the master's event delivery information (stage 1) from the logging of the master's event-processing transaction completion information (stage 2) in the shared in-memory log (using Viewstamped Replication [80]) in order to guarantee consistency under joint switch and controller failures. Besides, it adds explicit acknowledgement messages to the OpenFlow 1.3 protocol and implements buffers on existing switches for event retransmission and command filtering. The main objective of the addition of these extensions and mechanisms is to guarantee the exactly-once execution of any event transaction on the switches during controller failures.

Such strong correctness guarantees for a logically centralized controller under Ravana come at the cost of generating additional throughput and latency overheads that can be reduced to a quite reasonable extent with specific performance optimizations. Since the Ravana runtime is completely transparent and oblivious to control applications, achieving relaxed consistency requirements for the sake of improved availability as required by some specific applications, entails considering new mechanisms that consider relaxing some of the correctness constraints on Ravana's design goals.

A similar approach to Ravana [66] was adopted by Mantas et. al [81] to achieve a consistent and fault-tolerant SDN controller platform. In their ongoing work, the authors claim to retain the same requirements expressed by Ravana, namely the transparency, reliability, consistency and performance guarantees, but without requiring changes to the OpenFlow protocol or to existing switches.

Likewise, Kandoo [31] falls in this category of logically centralized controllers that distribute the control state by propagating network events. Indeed, Kandoo assumes, at the top layer of its hierarchical design, a logically centralized root controller for handling global and rare network events. Since the main aim was to preserve scalability without changing the OpenFlow devices, Kandoo did not focus on knowledge distribution mechanisms for achieving network state consistency.

ONOS and OpenDayLight:

ONOS and OpenDayLight [82] represent another category of logically centralized SDN solutions that set themselves apart from state-of-the-art distributed SDN controller platforms by offering community-driven open-source frameworks as well as providing the full functionalities of Network Operating Systems. Despite their obvious similarities, these prominent Java-based projects present major differences in terms of structure, target customers, focus areas and inspirations.

Dissimilar to OpenDayLight [83] which is applicable to different domains, ONOS [58] from ON.LAB is specifically targeted towards service providers and is thus architected to meet their carrier-grade requirements in terms of scalability, high-availability and performance. In addition to the high-level Northbound abstraction (a global network view and an application intent framework) and the pluggable Southbound abstraction (supporting multiple protocols), ONOS, in the same way as Onix and Hyperflow, offers state dissemination mechanisms [84] to achieve a consistent network state across the distributed cluster of ONOS controllers, a required or highly desirable condition for network applications to run correctly.

More specifically, ONOS's distributed core eases the state management and cluster coordination tasks for application developers by providing them with an available set of core building blocks for dealing with different types of distributed control plane state, including a *ConsistentMap* primitive for state requiring strong consistency and an *EventuallyConsistentMap* for state tolerating weak consistency.

In particular, applications that favor performance over consistency store their state in the shared eventually-consistent data structure that uses optimistic replication assisted by the gossip-based Anti-Entropy protocol [63]. For example, the global network topology state which should be accessible to applications with minimal delays is managed by the *Network Topology store* according to this eventual consistency model. Recent releases of ONOS treat the network topology view as an in-memory state machine graph. The latter is built and updated in each SDN controller by applying local topology events and replicating them to other controller instances in the cluster in an order-aware fashion based on the events' logical timestamps. Potential conflicts and loss of updates due to failure scenarios are resolved by the anti-entropy approach [63] where each controller periodically compares its topology view with that of another randomly-selected controller in order to reconcile possible differences and recover from stale information.

On the other hand, state imposing strong consistency guarantees is managed by the second data structure primitive built using RAFT [85], a protocol that achieves consensus via an elected leader controller in charge of replicating the received log updates to follower controllers and then committing these updates upon receipt of confirmation from the majority. The mapping between controllers and switches which is handled by ONOS's *Mastership store* is an example of a network state that is maintained in a strongly consistent manner.

Administered by the Linux Foundation and backed by the industry, OpenDayLight (ODL) [83] is a generic and general-purpose controller framework which, unlike ONOS, was conceived to accommodate a wide variety of applications and use cases concerning different domains (e.g. Data Center, Service Provider and Enterprise). One important architectural feature of ODL is its YANG-based Model-Driven Service Abstraction Layer (MD-SAL) that allows for the easy and

flexible incorporation of network services requested by the higher layers via the Northbound Interface (OSGi framework and the bidirectional RESTful Interfaces) irrespective of the multiple Southbound protocols used between the controller and the heterogeneous network devices.

The main focus of ODL was to accelerate the integration of SDN in legacy network environments by automating the configuration of traditional network devices and enabling their communication with OpenFlow devices. As a result, the project was perceived as adopting vendor-driven solutions that mainly aim at preserving the brands of legacy hardware. This represents a broad divergence from ONOS which envisions a carrier-grade SDN platform with enhanced performance capabilities to explore the full potential of SDN and demonstrate its real value.

The latest releases of ODL provided a distributed SDN controller architecture referred to as ODL clustering. Differently from ONOS, ODL did not offer various consistency models for different types of network data. All the data shared across the distributed cluster of ODL controllers for maintaining the logically centralized network view is handled in a strongly-consistent manner using the RAFT consensus algorithm [85] and the Akka framework [86].

B4 and SWAN:

Google's B4 [75] network leverages the logical centralization enabled by the SDN paradigm to deploy centralized TE in coexistence with the standard shortest-path routing for the purpose of increasing the utilization of the inter-data-center links (near 100%) as compared to conventional networks and thereby enhancing network efficiency and performance. As previously explained in Section III-B2, the logically centralized *TE server* uses the network information collected by the centralized *SDN Gateway* to control and coordinate the behavior of site-level SDN controllers based on an abstracted topology view. The main task of the TE server is indeed to optimize the allocation of bandwidth among competing applications (based on their priority) across the geographically-distributed data-center sites.

That being said, we implicitly assume the presence of a specific consistency model used by the centralized SDN Gateway for handling the distributed network state across the data-center site controllers and ensuring that the centralized TE application runs correctly based on a consistent network-wide view. However, there has been very little information provided on the level of consistency adopted by the B4 system. As a matter of fact, one potential downside of the SDN approach followed by Google could be the fact that it is too customized and tailored to their specific network requirements as no general control model has been proposed for future use by other SDN projects.

Similarly, Microsoft has presented SWAN [87] as an intra-domain software-driven WAN deployment that takes advantage of the logically-centralized SDN control using a global TE solution to significantly improve the efficiency, reliability and fairness of their inter-DC WAN. In the same way as Google, Microsoft did not provide much information about the control plane state consistency updates.

B. Logically distributed SDN control

The potential of the SDN paradigm has been properly explored within single administrative domains like data centers, enterprise networks, campus networks and even WANs as discussed in Section IV-A. Indeed, the main pillars of SDN – the decoupling between the control and data planes together with the consequent ability to program the network in a logically centralized manner – have unleashed productive innovation and novel capabilities in the management of such intra-domain networks. These benefits include the effective deployment of new domain-specific services as well as the improvement of standard control functions following the SDN principles like intra-domain routing and TE. RCP [88] and RouteFlow [89] are practical examples of successful intra-AS platforms that use OpenFlow to provide conventional IP routing services in a centralized manner.

However, that main feature of logically-centralized control which has been leveraged by most SDN solutions to improve network management at the intra-domain level, cannot be fully exploited for controlling heterogeneous networks involving multiple Autonomous Systems (ASes) under different administrative authorities (e.g. the Internet). In this context, recent works have considered extending the SDN scheme to such inter-domain networks while remaining compatible with their distributed architecture. In this section, we shed light on these SDN solutions which adopted a logically distributed architecture in accordance with legacy networks. For that reason, we place them in the category of logically distributed SDN platforms as opposed to the logically centralized ones mainly used for intra-domain scenarios.

DISCO and D-SDN:

For instance, the DISCO project [59] suggests a logically distributed control plane architecture that operates in such multi-domain heterogeneous environments, more precisely WANs and overlay networks. Built on top of Floodlight [53], each DISCO controller administers its own SDN network domain and interacts with other controllers to provide end-to-end network services. This inter-AS communication is ensured by a unique lightweight control channel to share summary network-wide information.

The most obvious contribution of DISCO lies in the separation between intra-domain and inter-domain features of the control plane, while each type of features is performed by a separate part of the DISCO architecture. The intra-domain modules are responsible for ensuring the main functions of the controller such as monitoring the network and reacting to network issues, and the inter-domain modules (Messenger, Agents) are designed to enable a message-oriented communication between neighbor domain controllers. Indeed, the AMQP-based Messenger [90] offers a distributed publish/subscribe communication channel used by agents which operate at the inter-domain level by exchanging aggregated information with intra-domain modules. DISCO was assessed on an emulated environment according to three use cases: inter-domain topology disruption, end-to end service priority request and Virtual Machine Migration.

The main advantage of the DISCO solution is the possibility to adapt it to large-scale networks with different ASes such as the Internet [79]. However, we believe that there are also several drawbacks associated with such a solution including the static non-evolving decomposition of the network into several independent entities, which is in contrast to emerging theories such as David D. Clark's theory [91] about the network being manageable by an additional high-level entity known as the Knowledge Plane. Besides, following the DISCO architecture, network performance optimization becomes a local task dedicated to local entities with different policies, each of which acts in its own best interest at the expense of the general interest. This leads to local optima rather than the global optimum that achieves the global network performance. Additionally, from the DISCO perspective, one SDN controller is responsible for one independent domain. However, an AS is usually too large to be handled by a single controller. Finally, DISCO did not provide appropriate reliability strategies suited to its geographically-distributed architecture. In fact, in the event of a controller failure, one might infer that a remote controller instance will be in charge of the subset of affected switches, thereby resulting in a significant increase in the control plane latency. In our opinion, a better reliability strategy would involve local per-domain redundancy; Local controller replicas should indeed take over and serve as backups in case the local primary controller fails.

In the same spirit, INRIA's D-SDN [92] enables a logical distribution of the SDN control plane based on a hierarchy of *Main Controllers* and *Secondary Controllers*, matching the organizational and administrative structure of current and future Internet. In addition to dealing with levels of control hierarchy, another advantage of D-SDN over DISCO is related to its enhanced security and fault tolerance features.

SDX-based Controllers:

Different from DISCO which proposes per-domain SDN controllers with inter-domain functions for allowing autonomous end-to-end flow management across SDN domains, recent trends have considered deploying SDN at Internet eXchange Points (IXPs) thus, giving rise to the concept of Software-Defined eXchanges (SDXes). These SDXes are used to interconnect participants of different domains via a shared software-based platform. That platform is usually aimed at bringing innovation to traditional peering, easing the implementation of customized peering policies and enhancing the control over inter-domain traffic management.

Prominent projects adopting that vision of software-defined IXPs and implementing it in real production networks include Google's Cardigan in New Zealand [93], SDX at Princeton [94], CNRS's French TouIX [95] (European ENDEAVOUR [96]) and the AtlanticWave-SDX [97]. Here we chose to focus on the SDX project at Princeton since we believe in its potential for demonstrating the capabilities of SDN to innovate IXPs and for bringing answers to deploying SDX in practice.

The SDX project [94] takes advantage of SDN-enabled IXPs to fundamentally improve wide-area traffic delivery and enhance conventional inter-domain routing protocols that lack the required flexibility for achieving various TE tasks. Today's BGP is indeed limited to destination-based routing, it has a local forwarding influence restricted to immediate neighbors, and it deploys indirect mechanisms for controlling path selection. To overcome these limitations, SDX relies on SDN features to ensure fine-grained, flexible and direct expression of inter-domain control policies, thereby enabling a wider range of valuable end-to-end services such as Inbound TE, application-specific peering, server load balancing, and traffic redirection through middle-boxes.

The SDX architecture consists of a smart SDX controller handling both SDX policies (*Policy compiler*) and BGP routes (*Route Server*), conventional Edge routers, and an OpenFlow-enabled switching fabric. The main idea behind this implementation is to allow participant ASes to compose their own policies in a high-level (using Pyretic) and independent manner (through the virtual switch abstraction), and then send them to the SDX controller. The latter is in charge of compiling these policies to SDN forwarding rules while taking into account BGP information.

Besides offering this high-level softwarized framework that is easily integrated into the existing infrastructure while maintaining good interoperability with its routing protocol, SDX also stands out from similar solutions like Cardigan [93] by the efficient mechanisms used for optimizing control and data plane operations. In particular, the scalability challenges faced by SDX under realistic scenarios have been further investigated by iSDX [98], an enhanced Ryu [72]-based version of SDX intended to operate at the scale of large industrial IXPs.

However, one major drawback of the SDX contribution is that it is limited to the participant ASes being connected via the software-based IXP, implying that non-peering ASes would not benefit from the routing opportunities offered by SDX. Besides, while solutions built on SDX use central TE policies for augmenting BGP and promote a logical centralization of the routing control plane at the IXP level, SDX controllers are still logically decentralized at the inter-domain level since no information is shared between them about their respective interconnected ASes. This brings us back to the same problem we pointed out for DISCO [59] about end-to-end traffic optimization being a local task for each part of the network.

To remedy this issue, some recent works [99] have considered centralizing the whole inter-domain routing control plane to improve BGP convergence by outsourcing the control logic to a multi-AS routing controller that has a "Bird's-eye view" over multiple ASes.

It is also worth mentioning that SDX-based controllers face several limitations in terms of both security and reliability.

Because the SDX controller is the central element in the SDX architecture, security strategies must focus on securing the SDX infrastructure by protecting the SDX controller against cyber attacks and by authenticating any access to it. In particular, Chung *et al.* [100] argue that SDX-based controllers are subjected to the potential vulnerabilities introduced by SDN in addition to the common vulnerabilities associated with classical protocols. In that respect, they distinguish three types of current SDX architectures and discuss the involved security concerns. In their opinion, Layer-3 SDX [93, 94] will inherit all BGP vulnerabilities, Layer-2 SDX [101] will get the vul-

nerabilities of a shared Ethernet network, and SDN SDX [40] will also bring controller vulnerabilities like DDoS attacks, comprised controllers and malicious controller applications. Moreover, the same authors of [100] point out that SDX-based controllers require security considerations with respect to Policy isolation between different SDX participants.

Finally, since the SDX controller becomes a potential single point of failure, fault-tolerance and resiliency measures should be taken into account when building an SDX architecture. While the distributed peer-to-peer SDN SDX architecture [102] is inherently resilient, centralized SDX approaches should incorporate fault-tolerance mechanisms like that discussed in Section V-B and should also leverage the existing fault-tolerant distributed SDN controller platforms [58].

V. Summary and Future Perspectives

While offering a promising potential to transform and improve current networks, the SDN initiative is still in the early stages of addressing the wide variety of challenges involving different disciplines. In particular, the distributed control of SDNs faces a series of pressing challenges that require our special consideration. These include the issues of (1) Scalability (2) Reliability (3) Consistency (4) Interoperability, (5) Monitoring and (6) Security).

In this paper, we surveyed the most prominent state-of-the art distributed SDN controller platforms and more importantly we discussed the different approaches adopted in tackling the above challenges and proposing potential solutions. Table I gives a brief summary of the main features and KPIs of the discussed SDN controllers. Physically-centralized controllers such as NOX [51], POX [56] and FloodLight [53] suffer from scalability and reliability issues. Solutions like DevoFlow [28] and DIFANE [27] attempted to solve these scalability issues by rethinking the OpenFlow protocol whereas most SDN groups geared their focus towards distributing the control plane. While some of the distributed SDN proposals such as Kandoo [31] promoted a hierarchical organization of the control plane to further improve scalability, other alternatives opted for a flat organization for increased reliability and performance (latency). On the other hand, distributed platforms like Onix [60], HyperFlow [60], ONOS [58] and OpenDaylight [83], focused on building consistency models for their logically centralized control plane designs. In particular, Onix [60] chose DHT and transactional databases for network state distribution over the Publish/Subscribe system used by HyperFlow [61]. Another different class of solutions has been recently introduced by DISCO which promoted a logically distributed control plane based on existing ASs within the Internet.

In previous sections, we classified these existing controllers according to the physical organization of the control plane (*Physical classification*) and, alternatively, according to the way knowledge is shared in distributed control plane designs (*Logical classification*). Furthermore, within each of these classifications, we performed another internal classification based on the similarities between competing SDN controllers (*The color classification* shown in Figure 3 and Figure 4).

In light of the above, it is obvious that there are various approaches to building a distributed SDN architecture; Some of these approaches met some performance criteria better than others but failed in some other aspects. Clearly, none of the proposed SDN controller platforms met all the discussed challenges and fulfilled all the KPIs required for an optimal deployment of SDN. At this stage, and building on these previous efforts, we communicate our vision of a distributed SDN control model by going through these open challenges, identifying the best ways of solving them, and envisioning future opportunities:

A. Scalability

Scalability concerns in SDN may stem from the decoupling between the control and data planes [103] and the centralization of the control logic in a software-based controller. In fact, as the network grows in size (e.g. switches, hosts, etc.), the centralized SDN controller becomes highly solicited (in terms of events/requests) and thus overloaded (in terms of bandwidth, processing power and memory). Furthermore, when the network scales up in terms of both size and diameter, communication delays between the SDN controller and the network switches may become high, thus affecting flow-setup latencies. This may also cause congestion in both the control and data planes and may generate longer failover times [7].

That said, since control plane scalability in SDN is commonly assessed in terms of both *throughput* (the number of flow requests handled per second) and *flow setup latency* (the delay to respond flow requests) metrics [7], a single physically-centralized SDN controller may not particularly fulfill the performance requirements (with respect to these metrics) of large-scale networks as compared to small or medium scale networks (see Section III-A).

One way to alleviate some of these scalability concerns is to extend the responsibilities of the data plane in order to relieve the load on the controller (see Section II-A). The main drawback of that method is that it imposes some modifications to the design of OpenFlow switches.

The second way, which we believe to be more effective, is to model the control plane in a way that mitigates scalability limitations. In a physically-centralized control model, a single SDN controller is in charge of handling all requests coming from SDN switches. As the network grows, the latter is likely to become a serious bottleneck in terms of scalability and performance [104]. On the other hand, a physically-distributed control model uses multiple controllers that maintain a logically centralized network view. This solution is appreciated for handling the controller bottleneck, hence ensuring a better scale of the network control plane while decreasing control-plane latencies.

Even though the distributed control model is considered as a scalable option when compared to the centralized control model, achieving network scalability while preserving good performance requires a relevant control distribution scheme that takes into account both the organization of the SDN control plane and the physical placement of the SDN controllers. In this context, we recommend a hierarchical organization of the control plane over a flat organization for increased scalability and improved performance. We also believe that

	Control Plane Architecture	Control Plane Design	Programming language	Scalability	Reliability	Consistency
NOX [51]	Physically Centralized	–	C++	Very Limited	Limited	Strong
POX [56]	Physically Centralized	–	Python	Very Limited	Limited	Strong
Floodlight [53]	Physically Centralized	–	Java	Very Limited	Limited	Strong
SMaRtLight [70]	Physically Centralized	–	Java	Very Limited	Very Good	Strong
Ravana [66]	Physically Centralized	–	Python	Limited	Very Good	Strong
ONIX [60]	Physically Distributed Logically Centralized	Flat	Python C	Very Good	Good	Weak Strong
HyperFlow [61]	Physically Distributed Logically Centralized	Flat	C++	Good	Good	Eventual
ONOS [58]	Physically Distributed Logically Centralized	Flat	Java	Very Good	Good	Weak Strong
OpenDayLight [83]	Physically Distributed Logically Centralized	Flat	Java	Very Good	Good	Strong
B4 [75]	Physically Distributed Logically Centralized	Hierarchical	Python C	Good	Good	N/A
Kandoo [31]	Physically Distributed Logically Centralized	Hierarchical	C C++ Python	Very Good	Limited	N/A
DISCO [59]	Physically Distributed Logically Distributed	Flat	Java	Good	Limited	Strong (inter-domain)
SDX [94]	Physically Distributed Logically Distributed	Flat	Python	Limited	N/A	Strong
DevoFlow [28]	Physically Distributed Logically Centralized	N/A	Java	Good	N/A	N/A
DIFANE [27]	Physically Distributed Logically Centralized	N/A	–	Good	N/A	N/A

TABLE I: Main Characteristics of the discussed SDN controllers

the placement of controllers should be further investigated and treated as an optimization problem that depends on specific performance metrics [48].

Finally, by physically distributing the SDN control plane for scalability (and reliability V-B) purposes, it is worth mentioning that new kinds of challenges may arise. In particular, to maintain the logically centralized view, a strongly-consistent model can be used to meet certain application requirements. However, as discussed in Section V-C, a strongly consistent model may introduce new scalability issues. In fact, retaining strong consistency when propagating frequent state updates might block the state progress and cause the network to become unavailable, thus increasing switch-to-controller latencies.

B. Reliability

Concerns about reliability have been considered as serious in SDN. The data-to-control plane decoupling has indeed a significant impact on the reliability of the SDN control plane. In a centralized SDN-based network, the failure of the central controller may collapse the overall network. In contrast, the use of multiple controllers in a physically distributed (but logically centralized) controller architecture alleviates the issue of a single point of failure.

Despite not providing information on how a distributed SDN controller architecture should be implemented, the OpenFlow standard gives (since version 1.2) the ability for a switch to simultaneously connect to multiple controllers. That OpenFlow option allows each controller to operate in one of three roles (*master*, *slave*, *equal*) with respect to an active connection to

the switch. Leveraging on these OpenFlow roles which refer to the importance of controller replication in achieving a highly available SDN control plane, various resiliency strategies have been adopted by different fault-tolerant controller architectures. Among the main challenges faced by these architectures are control state redundancy and controller failover.

Controller redundancy can be achieved by adopting different approaches for processing network updates. In the Active replication approach [69], also known as State Machine Replication, multiple controllers process the commands issued by the connected clients in a coordinated and deterministic way in order to concurrently update the replicated network state. The main challenge of that method lies in enforcing a strict ordering of events to guarantee strong consistency among controller replicas. That approach for replication has the advantage of offering high resilience with an insignificant downtime, making it a suitable option for delay-intolerant scenarios. On the other hand, in passive replication, referred to as primary/backup replication, one controller (the *primary*) processes the requests, updates the replicated state, and periodically informs the other controller replicas (the *backups*) about state changes. Despite offering simplicity and lower overhead, the passive replication scheme may create (controller and switch) state inconsistencies and generate additional delay in case the primary controller fails.

Additional concerns that should be explored are related to the kind of information to be replicated across controllers. Existing controller platform solutions follow three approaches for achieving controller state redundancy [16]: state replication [60, 70], event replication [61, 66] and traffic replication [105].

Moreover, control distribution is a central challenge when designing a fault tolerant controller platform. The centralized control approach that follows the simple Master/Slave concept [66, 70] relies on a single controller (the *master*) that keeps the entire network state and takes all decisions based on a global network view. Backup controllers (the *slaves*) are used for fault-tolerance purposes. The centralized alternative is usually considered in small to medium-sized networks. On the other hand, in the distributed control approach [58, 60], the network state is partitioned across many controllers that simultaneously take control of the network while exchanging information to maintain the logically centralized network view. In that model, controller coordination strategies should be applied to reach agreements and solve the issues of concurrent updates and state consistency. Mostly effective in large-scale networks, the distributed alternative provides fault tolerance by redistributing the network load among the remaining active controllers.

Finally, the implementation aspect is another important challenge in designing a replication strategy [**?**]. While some approaches opted for replicating controllers that store their network state locally and communicate through a specific group coordination framework [106], other approaches went for replicating the network state by delegating state storage, replication and management to external data stores [58, 60, 61] like distributed data structures and distributed file systems.

Apart from controller redundancy, other works focused on failure detection and controller recovery mechanisms. Some of these works considered reliability criteria from the outset in the placement of distributed SDN controllers. Both the number and locations of controllers were determined in a reliability-aware manner while preserving good performance. Reliability was indeed introduced in the form of controller placement metrics (switch-to-controller delay, controller load) to prevent worst-case switch-to-controller re-assignment scenarios in the event of failures. Other works elaborated on efficient controller failover strategies that consider the same reliability criteria. Strategies for recovering from controller failures can be split into redundant controller strategies (with backups) and non-redundant controller strategies (without backups) [107].

The redundant controller strategy assumes more than one controller per controller domain; One primary controller actively controls the network domain and the remaining controllers (backups) automatically take over the domain in case it fails. Despite providing a fast failover technique, this strategy depends on the associated standby methods (*cold*, *warm* or *hot*) which have different advantages and drawbacks [108]. For instance, the cold standby method imposes a full initialization process on the standby controller given the complete loss of the state upon the primary controller failure. This makes it an adequate alternative for stateless applications. In contrast, the hot standby method is effective in ensuring a minimum recovery time with no controller state loss, but it imposes a high communication overhead due to the full state synchronization requirements between primary and standby controllers. The warm standby method reduces that communication overhead at the cost of a partial state loss.

On the other hand, the non-redundant controller strategy requires only one controller per controller domain. In case it fails, controllers from other domains extend their domains to adopt orphan switches, thereby reducing the network overhead. Two well-known strategies for non-redundant controllers are the greedy failover and the pre-partitioning failover [109]. While the former strategy relies on neighbor controllers to adopt orphan switches at the edge of their domains and from which they can receive messages, the latter relies on controllers to proactively exchange information about the list of switches to take over in controller failure scenarios.

All things considered, a number of challenges and key design choices based on a set of requirements are involved when adopting a specific controller replication and failover strategy. In addition to reliability and fault-tolerance considerations, scalability, consistency and performance requirements should be properly taken into account when designing a fault-tolerant SDN controller architecture.

C. Consistency

Contrary to physically centralized SDN designs, distributed SDN controller platforms face major consistency challenges [110, 111]. Clearly, physically distributed SDN controllers must exchange network information and handle the consistency of the network state being distributed across them and stored in their shared data structures in order to maintain a logically centralized network-wide view that eases the development of control applications. However, achieving a convenient level of consistency while keeping good performance

in software-defined networks facing network partitions is a complex task. As claimed by the CAP theorem applied to networks [112], it is generally impossible for SDN networks to simultaneously achieve all three of Consistency (C), high Availability (A) and Partition tolerance (P). In the presence of network partitions, a weak level of consistency in exchange for high availability (AP) results in state staleness causing an incorrect behavior of applications whereas a strong level of consistency serving the correct enforcement of network policies (CP) comes at the cost of network availability.

The *Strong Consistency* model used in distributed file systems implies that only one consistent state is observed by ensuring that any read operation on a data item returns the value of the latest write operation that occurred on that data item. However, such consistency guarantees are achieved at the penalty of increased data store access latencies. In SDNs, the strong consistency model guarantees that all controller replicas in the cluster have the most updated network information, albeit at the cost of increased synchronization and communication overhead. In fact, if certain data occurring in different controllers are not updated to all of them, then these data are not allowed to be read, thereby impacting network availability and scalability.

Strong consistency is crucial for implementing a wide range of SDN applications that require the latest network information and that are intolerant of network state inconsistencies. Among the distributed data store designs that provide strong consistency properties are the traditional SQL-based relational databases like Oracle [113] and MySQL [114].

On the other hand, as opposed to the strong consistency model, the *Eventual Consistency* model (sometimes referred to as a *Weak Consistency* model) implies that concurrent reads of a data item may return values that are different from the actual updated value for a transient time period. This model takes a more relaxed approach to consistency by assuming that the system will eventually (after some period) become consistent in order to gain in network availability. Accordingly, in a distributed SDN scenario, reads of some data occurring in different SDN controller replicas may return different values for some time before eventually converging to the same global state. As a result, SDN controllers may temporarily have an inconsistent network view and thus cause an incorrect application behavior.

Eventually-consistent models have also been extensively used by SDN designers for developing inconsistency-tolerant applications that require high scalability and availability. These control models provide simplicity and efficiency of implementation but they push the complexity of resolving state inconsistencies and conflicts to the application logic and the consensus algorithms being put in place by the controller platform. Cassandra [115], Riak [116] and Dynamo [117] are popular examples of NoSQL databases that have adopted the eventual consistency model.

All things considered, maintaining state consistency across logically centralized SDN controllers is a significant SDN design challenge that involves trade-offs between policy enforcement and network performance [118]. The issue is that achieving strong consistency in an SDN environment that is prone to network failures is almost impossible without compromising availability and without adding complexity to network state management. Panda et. al [112] proposed new ways to circumvent these impossibility results but their approaches can be regarded as specific to particular cases.

In a more general context, SDN designers need to leverage the flexibility offered by SDN to select the appropriate consistency models for developing applications with various degrees of state consistency requirements and with different policies. In particular, adopting a single consistency model for handling different types of shared states may not be the best approach to coping with such a heterogeneous SDN environment. As a matter of fact, recent works on SDN have stressed the need for achieving consistency at different levels. So far, two levels of consistency models have been applied to SDNs and adopted by most distributed SDN controller platforms: strong consistency and eventual consistency.

In our opinion, a hybrid approach that merges various consistency levels should be considered to find the optimal trade-off between consistency and performance. Unlike the previously-mentioned approaches which are based on static consistency requirements where SDN designers decide which consistency level should be applied for each knowledge upon application development, we argue that an SDN application should be able to assign a priority for each knowledge and, depending on the network context (.i.e. instantaneous constraints, network load, etc), select the appropriate consistency level that should be enforced.

In that sense, recent approaches [119] introduced the concept of adaptive consistency in the context of distributed SDN controllers, where adaptively-consistent controllers can tune their consistency level to reach the desired level of performance based on specific metrics. That alternative has the advantage of sparing application developers the tedious task of selecting the appropriate consistency level and implementing multiple application-specific consistency models. Furthermore, that approach can be efficient in handling the issues associated with eventual consistency models [120].

Finally, in the same way as scalability and reliability, we believe that consistency should be considered while investigating the optimal placement of controllers. In fact, minimizing inter-controller latencies (distances) which are critical for system performance facilitates controller communications and enhances network state consistency.

D. Interoperability

Alongside the concerns about the SDN network interoperability with legacy networks, there is the challenge of ensuring interoperability between disparate distributed SDN controllers belonging to different SDN domains and using different controller technologies in order to foster the development and adoption of SDN.

In today's multi-vendor environments, the limited interoperability between SDN controller platforms is mainly due to a lack of open standards for inter-controller communications. Apart from the standardization of the Southbound interface—OpenFlow being the most popular Southbound standard, there

is to date no open standard for the Northbound and East-Westbound interfaces to provide compatibility between OpenFlow implementations.

Despite the emerging standardization efforts underway by SDN organizations, we argue that there are many barriers to effective and rapid standardization of the SDN East-Westbound interfaces, including the heterogeneity of the data models being used by SDN controller vendors. Accordingly, we emphasize the need for common data models to achieve interoperability and facilitate the tasks of standardization in SDNs. In this context, YANG [121] has emerged as a solid data modeling language used to model configuration and state data for standard representation. This NETCONF-based contribution from IETF is intended to be extended in the future and it is, more importantly, expected to pave the way for the emergence of standard data models driving interoperability in SDN networks.

Among the recent initiatives taken in that direction, we can mention OpenConfig's [122] effort on building a vendor-neutral data model written in YANG for configuration and management operations. Also worth mentioning is ONF's OF-Config protocol [123] which implements a YANG-based data model referred to as the Core Data Model. That protocol was introduced to enable remote configuration of OpenFlow-capable equipments.

E. Other Challenges

An efficient network monitoring is required for the development of control and management applications in distributed SDN-based networks. However, collecting the appropriate data and statistics without impacting the network performance is a challenging task. In fact, the continuous monitoring of network data and statistics may generate excessive overheads and thus affect the network performance whereas the lack of monitoring may cause an incorrect behavior of management applications. Current network monitoring proposals have developed different techniques to find the appropriate trade-offs between data accuracy and monitoring overhead. In particular, IETF's NETCONF Southbound protocol provides some effective monitoring mechanisms for collecting statistics and configuring network devices. In the near future, we expect the OpenFlow specification to be extended to incorporate new monitoring tools and functions.

Like network monitoring, network security is another crucial challenge that should be studied. The decentralization of the SDN control reduces the risk associated with a single point of failure and attacks (e.g. the risk of a DDoS attack). However, the integrity of data flows between the SDN controllers and switches is still not safe. For instance, we can imagine that an attacker can corrupt the network by acting as an SDN controller. In this context, new solutions and strategies (e.g. based on TLS/SSL) have been introduced with the aim of guaranteeing security in SDN environments.
Another aspect related to SDN security is the isolation of flows and networks through network virtualization. In the case of an underlying physical SDN network, this could be implemented using an SDN network hypervisor that creates multiple logically-isolated virtual network slices (called vSDNs), each is managed by its own vSDN controller [14]. At this point, care should be taken to design and secure the SDN hypervisor as an essential part of the SDN network.

VI. Conclusion

Software-Defined Networking has increasingly gained traction over the last few years in both academia and research. The SDN paradigm builds its promises on the separation of concerns between the network control logic and the forwarding devices, as well as the logical centralization of the network intelligence in software components. Thanks to these key attributes, SDN is believed to work with network virtualization to fundamentally change the networking landscape towards more flexible, agile, adaptable and highly automated Next Generation Networks.

Despite all the hype, SDN entails many concerns and questions regarding its implementation and deployment. For instance, current SDN deployments based on physically-centralized control architectures raised several issues of scalability and reliability. As a result, distributed SDN control architectures were proposed as a suitable solution for overcoming these problems. However, there are still ongoing community debates about the best and most optimal approach to decentralizing the network control plane in order to harness the full potential of SDN.

The novel aspect of this survey is the special focus placed on studying the wide variety of existing SDN controller platforms. These platforms are categorized in two ways: based on a physical classification or a logical classification. Our thorough analysis of these proposals allowed us to achieve an extensive understanding of their advantages and drawbacks and to develop a critical awareness of the challenges facing the distributed control in SDNs.

The scalability, reliability, consistency, and interoperability of the SDN control plane are among the key competing challenges faced in designing an efficient and robust high-performance distributed SDN controller platform. Although considered as the main limitations of fully centralized SDN control designs, scalability and reliability are also major concerns that are expressed in the case of distributed SDN architectures as they are highly impacted by the structure of the distributed control plane (e.g. flat, hierarchical or hybrid organization) as well as the number and placement of the multiple controllers within the SDN network. Achieving such performance and availability requirements usually comes at the cost of guaranteeing a consistent centralized network view that is required for the design and correct behavior of SDN applications. Consistency considerations should therefore be explored among the trade-offs involved in the design process of an SDN controller platform. Last but not least, the interoperability between different SDN controller platforms of multiple vendors is another crucial operational challenge surrounding the development, maturity and commercial adoption of SDN. Overcoming that challenge calls for major standardization efforts at various levels of inter-controller communications (e.g Data models, Northbound and East-Westbound interfaces).

Furthermore, such interoperability guarantees with respect to different SDN technology solutions represent an important step towards easing the widespread interoperability of these SDN platforms with legacy networks and, effectively ensuring the gradual transition towards softwarized network environments.

Giving that rich variety of promising SDN controller platforms with their broad range of significant challenges, we argue that developing a brand-new one may not be the best solution. However, it is essential to leverage the existing platforms by aggregating, merging and improving their proposed ideas in order to get as close as possible to a common standard that could emerge in the upcoming years. That distributed SDN controller platform should meet the emerging challenges associated with next generation networks (e.g. IoT [124] and Fog Computing [125]).

With these considerations in mind, we intend, as part of our future work, to shed more light on the complex problem of distributed SDN control. We propose to split that problem into two manageable challenges which are correlated: The controller placement problem and the knowledge sharing problem. The first problem investigates the required number of controllers along with their appropriate locations with respect to the desired performance and reliability objectives and depending on the existing constraints. The second one is related to the type and amount of information to be shared among the controller instances given a desired level of consistency.

REFERENCES

[1] D. Kreutz, F. M. V. Ramos, P. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, p. 63, 2015.

[2] N. Samaan and A. Karmouch, "Towards autonomic network management: an analysis of current and future research directions." *IEEE Communications Surveys and Tutorials*, vol. 11, no. 3, pp. 22–36, 2009.

[3] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: An intellectual history of programmable networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, Apr. 2014.

[4] Y. Li, X. Su, J. Riekki, T. Kanter, and R. Rahmani, "A sdn-based architecture for horizontal internet of things services," in *2016 IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–7.

[5] M. Canini, D. D. Cicco, P. Kuznetsov, D. Levin, S. Schmid, and S. Vissicchio, "Stn: A robust and distributed sdn control plane," Open Networking Summit (ONS) Research track, March 2014.

[6] W. Braun and M. Menth, "Software-defined networking using openflow: Protocols, applications and architectural design choices." *Future Internet*, vol. 6, no. 2, pp. 302–336, 2014.

[7] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in software-defined networking (sdn)," *Computer Networks*, vol. 112, pp. 279 – 293, 2017.

[8] W. Li, W. Meng, and L. F. Kwok, "A survey on openflow-based software defined networks: Security challenges and countermeasures," *Journal of Network and Computer Applications*, vol. 68, pp. 126 – 139, 2016.

[9] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software-defined networking," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 1955–1980, Fourthquarter 2014.

[10] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys Tutorials*, vol. 16, no. 3, pp. 1617–1634, Third 2014.

[11] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and openflow: From concept to implementation," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 2181–2206, Fourthquarter 2014.

[12] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys Tutorials*, vol. 17, no. 1, pp. 27–51, Firstquarter 2015.

[13] C. Trois, M. D. D. Fabro, L. C. E. de Bona, and M. Martinello, "A survey on sdn programming languages: Toward a taxonomy," *IEEE Communications Surveys Tutorials*, vol. 18, no. 4, pp. 2687–2712, Fourthquarter 2016.

[14] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for software defined networking," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 655–685, Firstquarter 2016.

[15] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 623–654, Firstquarter 2016.

[16] P. Fonseca and E. Mota, "A survey on fault management in software-defined networks," *IEEE Communications Surveys Tutorials*, vol. PP, no. 99, pp. 1–1, 2017.

[17] I. T. Haque and N. Abu-Ghazaleh, "Wireless software defined networking: A survey and taxonomy," *IEEE Communications Surveys Tutorials*, vol. 18, no. 4, pp. 2713–2737, Fourthquarter 2016.

[18] H. Kim and N. Feamster, "Improving network management with software defined networking." *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[19] M. F. Bari, R. Boutaba, R. P. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, "Data center network virtualization: A survey." *IEEE Communications Surveys and Tutorials*, vol. 15, no. 2, pp. 909–928, 2013.

[20] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges." *J. Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[21] M. A. Sharkh, M. Jammal, A. Shami, and A. Ouda, "Resource allocation in a network-based cloud computing environment: design challenges," *IEEE Communications Magazine*, vol. 51, no. 11, pp. 46–52, November 2013.

[22] Ansible. [Online]. Available: https://www.ansible.com/

[23] ONF, "Open networking foundation," 2014. [Online].

Available: https://www.opennetworking.org/

[24] 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.

[25] A. Doria, J. H. Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, R. Gopal, and J. Halpern, "Forwarding and Control Element Separation (ForCES) Protocol Specification," Internet Engineering Task Force, Mar. 2010.

[26] ONF, "OpenFlow switch specification," Open Networking Foundation, Tech. Rep., December 2009. [Online]. Available: https://www.opennetworking.org/

[27] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 351–362.

[28] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 254–265.

[29] 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.

[30] 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.

[31] 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.

[32] M. Moshref, A. Bhargava, A. Gupta, M. Yu, and R. Govindan, "Flow-level state transition as a new switch primitive for sdn," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14, 2014, pp. 377–378.

[33] M. T. Arashloo, Y. Koral, M. Greenberg, J. Rexford, and D. Walker, "Snap: Stateful network-wide abstractions for packet processing," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16, 2016, pp. 29–43.

[34] S. Pontarelli, M. Bonola, G. Bianchi, A. Capone, and C. Cascone, "Stateful openflow: Hardware proof of concept," in *2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR)*, July 2015, pp. 1–8.

[35] G. Bianchi, M. Bonola, S. Pontarelli, D. Sanvito, A. Capone, and C. Cascone, "Open packet processor: a programmable architecture for wire speed platform-independent stateful in-network processing," *CoRR*, vol. abs/1605.01977, 2016.

[36] R. Bifulco, J. Boite, M. Bouet, and F. Schneider, "Improving sdn with inspired switches," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '16, 2016, pp. 11:1–11:12.

[37] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 5, pp. 81–94, Oct. 2007.

[38] V. Jeyakumar, M. Alizadeh, C. Kim, and D. Mazières, "Tiny packet programs for low-latency network control and monitoring," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, ser. HotNets-XII, 2013, pp. 8:1–8:7.

[39] J. Yang, X. Yang, Z. Zhou, X. Wu, T. Benson, and C. Hu, "Focus: Function offloading from a controller to utilize switch power," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2016, pp. 199–205.

[40] P. Lin, J. Bi, S. Wolff, Y. Wang, A. Xu, Z. Chen, H. Hu, and Y. Lin, "A west-east bridge based sdn inter-domain testbed." *IEEE Communications Magazine*, vol. 53, no. 2, pp. 190–197, 2015.

[41] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," in *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, ser. ICFP '11. New York, NY, USA: ACM, 2011, pp. 279–291.

[42] A. Voellmy, H. Kim, and N. Feamster, "Procera: A language for high-level reactive network control," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 43–48.

[43] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing software-defined networks," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, ser. nsdi'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 1–14.

[44] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136–141, February 2013.

[45] S. Azodolmolky, P. Wieder, and R. Yahyapour, "Performance evaluation of a scalable software-defined networking deployment," in *2013 Second European Workshop on Software Defined Networks*, Oct 2013, pp. 68–74.

[46] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 267–280.

[47] O. Michel and E. Keller, "Sdn in wide-area networks: A survey," in *2017 Fourth International Conference on Software Defined Systems (SDS)*, May 2017, pp. 37–42.

[48] B. Heller, R. Sherwood, and N. McKeown, "The con-

troller placement problem," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 7–12.

[49] M. T. I. ul Huque, W. Si, G. Jourjon, and V. Gramoli, "Large-scale dynamic controller placement," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 63–76, March 2017.

[50] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of sdn/openflow controllers," in *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, ser. CEE-SECR '13. New York, NY, USA: ACM, 2013, pp. 1:1–1:6.

[51] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: Towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, Jul. 2008.

[52] D. Erickson, "The beacon openflow controller," 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. 13–18.

[53] Floodlight Project. [Online]. Available: http://www.projectfloodlight.org/

[54] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying nox to the datacenter," in *Proc. of workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.

[55] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, ser. Hot-ICE'12, Berkeley, CA, USA, 2012, pp. 10–10.

[56] POX. [Online]. Available: http://www.noxrepo.org/pox/about-pox/

[57] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "Sphinx: Detecting security attacks in software-defined networks." in *NDSS*. The Internet Society, 2015.

[58] 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.

[59] K. Phemius, M. Bouet, and J. Leguay, "DISCO: distributed multi-domain SDN controllers," *CoRR*, vol. abs/1308.6138, 2013.

[60] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6.

[61] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, ser. INM/WREN'10, Berkeley, CA, USA, 2010, pp. 3–3.

[62] J. Stribling, Y. Sovran, I. Zhang, X. Pretzer, J. Li, M. F. Kaashoek, and R. Morris, "Flexible, wide-area storage for distributed systems with wheelfs," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2009, April 22-24, 2009, Boston, MA, USA*, 2009, pp. 43–58.

[63] A. Bianco, P. Giaccone, S. D. Domenico, and T. Zhang, "The role of inter-controller traffic for placement of distributed SDN controllers," *CoRR*, vol. abs/1605.09268, 2016.

[64] B. Chandrasekaran and T. Benson, "Tolerating sdn application failures with legosdn," in *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XIII, 2014, pp. 22:1–22:7.

[65] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B. B. Kang, "Rosemary: A robust, secure, and high-performance network operating system," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14, 2014, pp. 78–89.

[66] N. Katta, H. Zhang, M. Freedman, and J. Rexford, "Ravana: Controller fault-tolerance in software-defined networking," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, ser. SOSR '15, 2015, pp. 4:1–4:12.

[67] S. H. Yeganeh and Y. Ganjali, "Beehive: Simple distributed programming in software-defined networks," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '16, 2016, pp. 4:1–4:12.

[68] B. Chandrasekaran, B. Tschaen, and T. Benson, "Isolating and tolerating sdn application failures with legosdn," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '16, 2016, pp. 7:1–7:12.

[69] E. S. Spalla, D. R. Mafioletti, A. B. Liberato, G. Ewald, C. E. Rothenberg, L. Camargos, R. S. Villaca, and M. Martinello, "Ar2c2: Actively replicated controllers for sdn resilient control plane," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, April 2016, pp. 189–196.

[70] F. Botelho, A. Bessani, F. M. V. Ramos, and P. Ferreira, "On the design of practical fault-tolerant sdn controllers," in *2014 Third European Workshop on Software Defined Networks*, Sept 2014, pp. 73–78.

[71] A. Bessani, J. a. Sousa, and E. E. P. Alchieri, "State machine replication for the masses with bft-smart," in *Proceedings of the 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, ser. DSN '14, 2014, pp. 355–362.

[72] Ryu SDN Framework. [Online]. Available: https://osrg.github.io/ryu/

[73] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems," in *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, ser.

USENIXATC'10, 2010, pp. 11–11.

[74] Y. Liu, A. Hecker, R. Guerzoni, Z. Despotovic, and S. Beker, "On optimal hierarchical sdn," in *2015 IEEE International Conference on Communications (ICC)*, June 2015, pp. 5374–5379.

[75] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 3–14.

[76] A. Kumar, S. Jain, U. Naik, A. Raghuraman, N. Kasinadhuni, E. C. Zermeno, C. S. Gunn, J. Ai, B. Carlin, M. Amarandei-Stavila, M. Robin, A. Siganporia, S. Stuart, and A. Vahdat, "Bwe: Flexible, hierarchical bandwidth allocation for WAN distributed computing," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015, London, United Kingdom, August 17-21, 2015*, 2015, pp. 1–14.

[77] T. D. Chandra, R. Griesemer, and J. Redstone, "Paxos made live: An engineering perspective," in *Proceedings of the Twenty-sixth Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '07. New York, NY, USA: ACM, 2007, pp. 398–407.

[78] K.-K. Yap, M. Motiwala, J. Rahe, S. Padgett, M. Holliman, G. Baldus, M. Hines, T. Kim, A. Narayanan, A. Jain, V. Lin, C. Rice, B. Rogan, A. Singh, B. Tanaka, M. Verma, P. Sood, M. Tariq, M. Tierney, D. Trumic, V. Valancius, C. Ying, M. Kallahalla, B. Koley, and A. Vahdat, "Taking the edge off with espresso: Scale, reliability and programmability for global internet peering," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: ACM, 2017, pp. 432–445.

[79] R. B. Fouad Benamrane, Mouad Ben mamoun, "Performances of openflow-based softwaredefined networks: An overview," *Journal of Networks*, vol. 10, no. 6, pp. 329–337, 2015.

[80] B. Oki and B. Liskov, "Viewstamped replication: A new primary copy method to support highly-available distributed systems," in *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, Aug. 1988.

[81] A. Mantas and F. M. V. Ramos, "Consistent and fault-tolerant SDN with unmodified switches," *CoRR*, vol. abs/1602.04211, 2016.

[82] A. Bondkovskii, J. Keeney, S. van der Meer, and S. Weber, "Qualitative comparison of open-source sdn controllers," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, April 2016, pp. 889–894.

[83] OpenDayLight Project. [Online]. Available: http://www.opendaylight.org/

[84] A. S. Muqaddas, A. Bianco, P. Giaccone, and G. Maier, "Inter-controller traffic in onos clusters for sdn networks," in *2016 IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–6.

[85] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIX ATC'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 305–320.

[86] Akka Framework. [Online]. Available: http://akka.io/

[87] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 15–26.

[88] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and implementation of a routing control platform," in *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 15–28.

[89] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. Cunha de Lucena, and R. Raszuk, "Revisiting routing control platforms with the eyes and muscles of software-defined networking," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 13–18.

[90] AMQP. [Online]. Available: http://www.amqp.org/

[91] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the internet," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '03. New York, NY, USA: ACM, 2003, pp. 3–10.

[92] M. A. S. Santos, B. A. A. Nunes, K. Obraczka, T. Turletti, B. T. de Oliveira, and C. B. Margi, "Decentralizing sdn's control plane," in *IEEE 39th Conference on Local Computer Networks, LCN 2014, Edmonton, AB, Canada, 8-11 September, 2014*, 2014, pp. 402–405.

[93] J. P. Stringer, D. Pemberton, Q. Fu, C. Lorier, R. Nelson, J. Bailey, C. N. A. Corrêa, and C. E. Rothenberg, "Cardigan: SDN distributed routing fabric going live at an internet exchange," in *IEEE Symposium on Computers and Communications, ISCC 2014, Funchal, Madeira, Portugal, June 23-26, 2014*, 2014, pp. 1–7.

[94] A. Gupta, M. Shahbaz, L. Vanbever, H. Kim, R. Clark, N. Feamster, J. Rexford, and S. Shenker, "Sdx: A software defined internet exchange," *ACM SIGCOMM*, 2014.

[95] R. Lapeyrade, M. Bruyere, and P. Owezarski, "Openflow-based migration and management of the TouIX IXP," in *2016 IEEE/IFIP Network Operations and Management Symposium, NOMS 2016, Istanbul, Turkey, April 25-29, 2016*, 2016, pp. 1131–1136.

[96] ENDEAVOUR Project. [Online]. Available: https://www.h2020-endeavour.eu/

[97] Heidi Morgan, "Atlanticwave-sdx: A distributed in-

tercontinental experimental software defined exchange for research and education networking," Press Release, April 2015.

[98] A. Gupta, R. MacDavid, R. Birkner, M. Canini, N. Feamster, J. Rexford, and L. Vanbever, "An industrial-scale software defined internet exchange point," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. Santa Clara, CA: USENIX Association, Mar. 2016, pp. 1–14.

[99] V. Kotronis, A. Gämperli, and X. Dimitropoulos, "Routing centralization across domains via sdn," *Comput. Netw.*, vol. 92, no. P2, pp. 227–239, Dec. 2015.

[100] J. Chung, J. Cox, J. Ibarra, J. Bezerra, H. Morgan, R. Clark, and H. Owen, "Atlanticwave-sdx: An international sdx to support science data applications," in *Software Defined Networking (SDN) for Scientific Networking Workshop*, Austin, Texas, 11 2015.

[101] Internet2, "Advanced layer 2 system." [Online]. Available: https://www.internet2.edu/products-services/advanced-networking/layer-2-services/

[102] J. Chung, H. Owen, and R. Clark, "Sdx architectures: A qualitative analysis," in *SoutheastCon 2016*, March 2016, pp. 1–8.

[103] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software defined networking: State of the art and research challenges," *Computer Networks*, vol. 72, pp. 74 – 98, 2014.

[104] K. Qiu, S. Huang, Q. Xu, J. Zhao, X. Wang, and S. Secci, "Paracon: A parallel control plane for scaling up path computation in sdn," *IEEE Transactions on Network and Service Management*, vol. PP, no. 99, pp. 1–1, Oct 2017.

[105] P. Fonseca, R. Bennesby, E. Mota, and A. Passito, "Resilience of sdns based on active and passive replication mechanisms," in *2013 IEEE Global Communications Conference (GLOBECOM)*, Dec 2013, pp. 2188–2193.

[106] V. Yazici, M. O. Sunay, and A. O. Ercan, "Controlling a software-defined network via distributed controllers," *CoRR*, vol. abs/1401.7651, 2014.

[107] N. Kong, "Design concept for a failover mechanism in distributed sdn controllers," in *Master's Project*, 2017. [Online]. Available: http://scholarworks.sjsu.edu/etd_projects/548/

[108] V. Pashkov, A. Shalimov, and R. Smeliansky, "Controller failover for sdn enterprise networks," in *2014 International Science and Technology Conference (Modern Networking Technologies) (MoNeTeC)*, Oct 2014, pp. 1–6.

[109] M. Obadia, M. Bouet, J. Leguay, K. Phemius, and L. Iannone, "Failover mechanisms for distributed sdn controllers," in *2014 International Conference and Workshop on the Network of the Future (NOF)*, vol. Workshop, Dec 2014, pp. 1–6.

[110] L. Schiff, S. Schmid, and P. Kuznetsov, "In-band synchronization for distributed sdn control planes," *SIGCOMM Comput. Commun. Rev.*, vol. 46, no. 1, pp. 37–43, Jan. 2016.

[111] F. Botelho, T. A. Ribeiro, P. Ferreira, F. M. V. Ramos, A. Bessani, undefined, undefined, undefined, and undefined, "Design and implementation of a consistent data store for a distributed sdn control plane," *2016 12th European Dependable Computing Conference (EDCC)*, vol. 00, pp. 169–180, 2016.

[112] A. Panda, C. Scott, A. Ghodsi, T. Koponen, and S. Shenker, "Cap for networks," 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. 91–96.

[113] Oracle. [Online]. Available: https://www.oracle.com

[114] MySQL. [Online]. Available: http://www.mysql.fr/

[115] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010.

[116] R. Klophaus, "Riak core: Building distributed applications without shared state," in *ACM SIGPLAN Commercial Users of Functional Programming*, ser. CUFP '10. New York, NY, USA: ACM, 2010, pp. 14:1–14:1.

[117] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," in *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP '07, 2007, pp. 205–220.

[118] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: state distribution trade-offs in software defined networks," in *Proceedings of the first workshop on Hot topics in software defined networks*, ser. HotSDN '12, 2012.

[119] M. Aslan and A. Matrawy, "Adaptive consistency for distributed sdn controllers," in *2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks)*, Sept 2016, pp. 150–157.

[120] E. Sakic, F. Sardis, J. W. Guck, and W. Kellerer, "Towards adaptive state consistency in distributed sdn control plane," in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–7.

[121] S. Wallin and C. Wikström, "Automating network and service configuration using netconf and yang," in *Proceedings of the 25th International Conference on Large Installation System Administration*, ser. LISA'11, Berkeley, CA, USA, 2011, pp. 22–22.

[122] OpenConfig. [Online]. Available: http://www.openconfig.net/

[123] "OF-CONFIG 1.2: Openflow Management and Configuration Protocol," Open Networking Foundation, Tech. Rep., 2014.

[124] M. Ojo, D. Adami, and S. Giordano, "A sdn-iot architecture with nfv implementation," in *2016 IEEE Globecom Workshops (GC Wkshps)*, Dec 2016, pp. 1–6.

[125] K. Liang, L. Zhao, X. Chu, and H. H. Chen, "An integrated architecture for software defined and virtualized radio access networks with fog computing," *IEEE Network*, vol. 31, no. 1, pp. 80–87, January 2017.

Fetia Bannour received her B.Eng degree in multidisciplinary engineering in 2014 from the Tunisia Polytechnic School of the University of Carthage, Tunisia. She is currently a PHD student in Computer Science in the LiSSi Laboratory, at the Networks and Telecommunications (N&T) Department of the University of Paris-Est Cretéil (UPEC, ex. Paris 12 University), France. Her research interests include computer networks, software-defined networks, distributed control, Internet of Things, adaptive and autonomous control systems, and network virtualization. She also served as a local organizer for SaCCoNeT 2016 and as a Student volunteer for IEEE ICC 2017.

Sami Souihi Associate Professor of Computer Science at the Networks and Telecommunications (N&T) Department of Paris-Est University, and the LiSSi Laboratory, France. He received his MSc degree from the University of Paris 6, France in 2010, and his PhD degree from the University of Paris-Est Cretéil (UPEC, ex. Paris 12 University) in 2013. His research work focuses on adaptive mechanisms in large-scale dynamic networks. These mechanisms are based on context-enhanced knowledge, Network Functions Virtualization (NFV) and also on Software-Defined Networking (SDN). He also served as a Technical Program Committee (TPC) member for international conferences (e.g. IEEE SACONET, IEEE ICC, IEEE Globecom and WWIC) and as a TPC Co-Chair of the IEEE ICC 2017 Symposium on Communications QoS, Reliability and Modeling. In addition, he is a reviewer for several IEEE conferences and journals, including IEEE ICC, IEEE WCNC, IEEE Globecom, IEEE Transactions on Networking, and IEEE Transactions on Communications.

Abdelhamid Mellouk (UPEC Full Professor), received the engineer degree in computer network engineering, the D.E.A. and Ph.D. degrees in computer science from the University of Paris Sud (ex. Paris 11 University), Orsay, France, and the D.Sc. (Habilitation) diploma from the University of Paris-Est Cretéil (UPEC, ex. Paris 12 University), France. He is currently a full professor at UPEC, at the Networks and Telecommunications (N&T) Department and the LiSSi Laboratory, France. Founder of the Network Control Research activity with extensive international academic and industrial collaborations, his general area of research focuses on computer networks, including adaptive real-time bio-inspired control mechanisms for high-speed dynamic wired/wireless networking with the objective to maintain acceptable quality of service/quality of experience for added value services dedicated to several domains such as ICT-Health. Prof. Mellouk has held several national and international offices, including leadership positions in the IEEE Communications Society Technical Committees. He has published/coordinated eight books, three lecture notes and several refereed international publications in journals, conferences, and books, in addition to numerous keynotes and plenary talks in flagship venues. He serves on the Editorial Boards or as an Associate Editor for several journals, and he is chairing or has chaired (or co-chaired) some of the top international conferences and symposia (e.g. ICC, GlobeCom and VTC).

View publication stats