Journal of Network and Innovative Computing.
ISSN 2160-2174 Volume 4 (2016) pp. 076–084

Software-Defined Networking Paradigm and OpenFlow Protocol: A Review

Safae Zerrik [1], **Mohamed Bakhouya** [2] **and Driss El Ouadghiri** [1]

[1]Faculty of science, Moulay Ismail University,
Meknes, Morocco
{*zerriksafae,dmelouad*}*@gmail.com*

[2]International University of Rabat, Parc Technopolis,
11 100 Sala el Jadida, Morocco
mohamed.bakhouya@uir.ac.ma

Abstract: **Software-defined networking is a rising technology in near future aiming to build programmable networks as a way to simplify the evolution of networks. It is a new networking paradigm with the main aim is to decouple forwarding hardware from control decisions. This article explores the capabilities of SDN and OpenFlow techniques, provides a state of the art review of existing works, and examines current and future research directions for developing SDN architectures.**
Keywords: Software-defined networking, OpenFlow, Scalability, SDN architectures, Controllers, Switch.

I. Introduction

Software Defined Networking (SDN) is a new approach for the design and network management by decoupling forwarding Plane from control decisions. It aims at standardization of the network configuration and control through open programmatic interfaces to individual network devices as well as the entire network. SDN will make networks more dynamic, flexible and cost-efficient, while significantly simplifying operational complexity. In addition, this solution provides several benefits like network and service customizability, configurability, increased performance and improved operation.

Switches, controllers and interfaces are main components of the SDN architecture (see Figure 1). Switches are often represented as basic transmission equipment accessible via an open interface. The controller provides features and a programmable interface to the network, where applications can be programmed to perform management. It is worth noting that SDN decouples the control plane and the data plane for more agility to handle the large growth rates of network devices and traffic demands. Data plane is also called forwarding plane or transport plane. Data plane and control plane cover layers 1-4 of the OSI model with additional functions to virtualize the network from end-to-end.
There are mainly three categories of interfaces: southbound, northbound, and East/West as depicted in Figure 1. Southbound interface defines the control communications between controller platform and data plane devices (including physical and virtual switches). This interface should allow several functionalities such as programmability and quick re-configurability, resource sharing, and traffic isolation. For example it supports flexibility in the control plane and makes it possible to easily adopt new control system on the network, facilitate the creation and the reconfiguration of virtual networks. The southbound interface should allow also the abstraction of physical resources characteristics in order to allow for the controller or the application accessing the capabilities of resources by using this interface. It also provides secure isolations among multiple virtual networks, such as performance and security. It can leverage other protocols in addition to Openflow (e.g., ForCES).

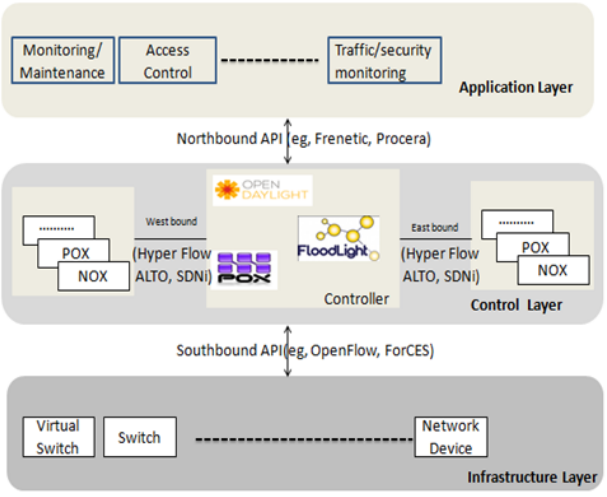

Figure. 1: SDN architecture[1]

The northbound interface refers to the software interfaces between modules of the controller platform and the SDN applications running on top of the network platform. Those applications can be routing (e.g., topology discovery, traffic engineering), management (e.g., monitoring, maintenance) and policy (e.g., access control, security). The East-West

interface is situated between controllers themselves in order to allow intra domains, inter domains, scalability and interoperability [2].

In order to allow access to the forwarding plane of a network device such as switches and routers, a communication protocol, called OpenFlow, is implemented between network infrastructure devices and an SDN controller (southbound API). This protocol uses the concept of flows to identify network traffic based on predefined match rules that are programmed by the SDN controller. Figure 2 illustrates the components of the Openflow system [3]. It is composed of three main components as illustrated in Figure 2. Flow tables consist of flow entries that determine how packets belonging to a flow will be processed and forwarded. Flow entries typically consist of: i) rules (or match fields), used to match incoming packets; ii) counters, used to collect statistics for the particular flow of bytes and duration of the flow; and iii) a set of actions, to be applied upon a packet.

The second component is the group table for handling specific forwarding like load-balancing or failover on different group of port or multicast flooding to all ports. The openFlow protocol to allow switch communicating with the controller and the controller manages the switch via the OpenFlow protocol. The OpenFlow channel is an interface that connects each OpenFlow switch to a controller, and, therefore OpenFlow-based controller can add, update, and delete flow entries in flow tables. For example when a packet arrives at an OpenFlow switch, the header fields are compared to flow table entries. If a match is found, the packet is either forwarded or dropped, depending on one or more actions stored in the flow table as illustrated in Figure 2.

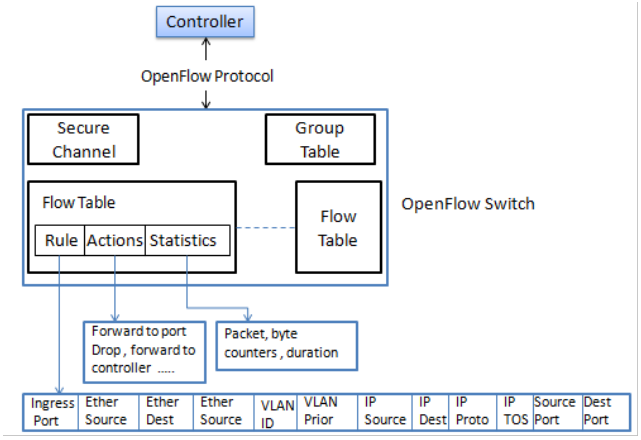

Figure. 2: Open Flow architecture (Table Fields (rules) specific to OpenFlow v1.0.0 [[4]])

In the past few years, several research work have been performed either at controller level or at switch level to address issues such security, interoperability, and scalability. The rest of this paper provides a state of the art review of existing works, and examines possible research directions for developing SDN architectures and applications. The remainder of this document is structured as follows. Section 2 provides a classification of exiting approaches. Section 3 introduces simulation tools and experimental platforms for evaluating SDN solutions. In Section 4, we present the different applications that require SDN capabilities. Section 5 presents some challenges of SDN and highlight some future research directions. Section 6 gives some conclusions and perspectives.

II. Related Work

Figure 3 illustrates a classification of existing approaches based on SDN levels: at controller level and at switch level. These approaches are described in the rest of this section.

A. Switch-based approaches

There are mainly two varieties of switches, Pure and Hybrid. Pure OpenFlow switches support only OpenFlow operations, while OpenFlow-hybrid switches support both OpenFlow and normal Ethernet switching operations. More precisely, pure switches are designed to perform efficient Openflow switching by developing either hardware or software solutions. For example in [5], to perform OpenFlow switching, authors have applied network processor based acceleration cards using different designs, and the result showed 20 per cent reduction on packet delay. A general comparison between Openflow-enabled switches from three vendors has been done by the authors in [6], and showed that control path delay and flow table design significantly impact latency and throughput.

Authors in [7] present the design and implementation of an architecture for accelerating a software OpenFlow implementation (Open vSwitch) using a programmable network processor. Results are reported and show that this implementation offers at least an 8-10x performance improvement over software only Open vSwitch. A study on data plane performance over Linux based OpenFlow switching is explained in [8]. Forwarding throughput, packet latency, OpenFlow switching, layer2 ethernet switching and layer 3 IP routing performance are compared and analyzed, employing multiple load conditions. Reported results show that openFlow has the ability to handle up to 128K exact matches and up to 100 wild-card matches, combined with good throughput, latency and fairness performance.

Hardware-based approaches focus on hardware implementations of switches. The implementation of an OpenFlow Switch on a NetFPGA platform is described in [9]. This implementation is managing all traffic and operating at line-rate is also evaluated according to three parameters: flow table size, forwarding rate, and new flow insertion rate. These parameters when implemented are able to fulfill full line-rate switching and simply accommodate active flows in a production. The size of the flow header entry can represent a real concern among other issues related to these three metrics.

The main aim of using hybrid switches is to connect a pure OpenFlow network with a non-OpenFlow network and an overlay OpenFlow on top of existing production networks. Hybrid OpenFlow model is an OpenFlow switch with two forwarding tables: OpenFlow controlled table dedicated for

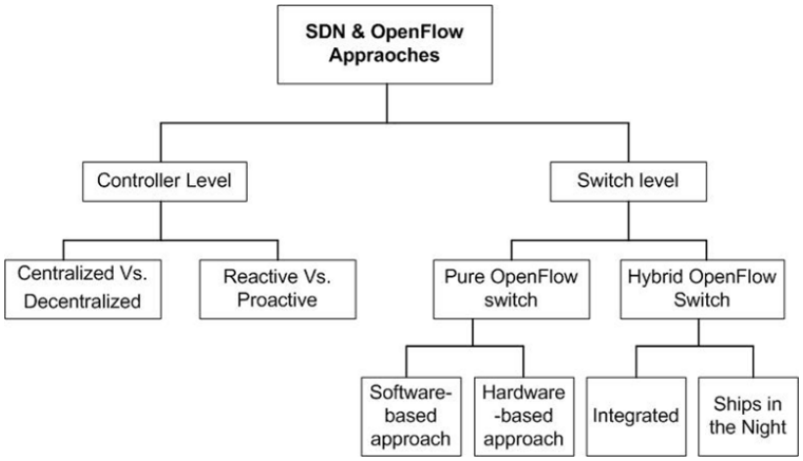

Figure. 3: Classification of SDN and OpenFlow approaches

Table 1: Implementation of some OpenFlow Switches.

Switch type	Model	impl/Marker	version
Pure SW-based	Open vSwitch[1]	C/Python	V1.0
Pure SW-based	Pica8[2]	C	V1.3
Pure HW-based	3290 and 3780	Pronto[3]	V1.0
Pure HW-based	RackSwitch G8264	IBM[4]	V1.0
Pure HW-based	Junos MX-series	Juniper[5]	V1.0
Hybrid Switch	PF5240 and PF5820	NEC[6]	V1.3 and V1.0

open flows, and source MAC address protocol information as regular table. Some ports of the switch use regular table and other ports use OpenFlow-controlled table. There are two forwarding models representing how OpenFlow and non-OpenFlow approaches might interact: Ships in the Night and Integrated. In the first model, the data plane traffic is segregated into OpenFlow and non-OpenFlow [10]. There are two common approaches to implement this model: approaches that split traffic using Ports and those that split traffic using VLAN. It is worth noting that Ships in the Night approaches are simple to implement, but present several limitations. For example, sharing of information between OpenFlow ports/vlans nodes and non-openflow ports/vlans is difficult. Thus, two separated networks should be built by the user: a regular network and a separate overlay openflow network.

The Integrated model attempts to capture how OpenFlow might be folded into an existing non-OpenFlow forwarding model. However, this model is more complicated to implement, but presents several advantages. For example, Services Migration to openflow should be done slowly, so other services traditionally implemented for the same traffic can be reused. Moreover, visibility is enhanced for end-user and more capability and flexibility is given to system administrator. Table 1 describes implementations of some Open Flow switches.

[1]http://openvswitch.org/

[2]http://www.pica8.com/

[3]www.iwnetworks.com/main/pronto switches

[4]www-03.ibm.com/systems/networking

[5]http://www.juniper.net

[6]http://www.nec.com/en/global/prod/pflow/

B. Controller-based approaches

The SDN controller is a new class of data networking product that will use a standard to communicate with the network devices. In fact, the controller is a platform on which many applications might run. Figure 4 illustrates the architecture of an SDN controller based on the OpenFlow protocol (Floodlight Controller[7]). It shows the separation between the controller and the application layers. Applications can be written in Java and can interact with the controller modules via a Java API. Other applications can be written in different languages and interact with the controller modules via the REST API. The controller manages forwarding devices, and it is responsible for configuring the forwarding state on those devices.

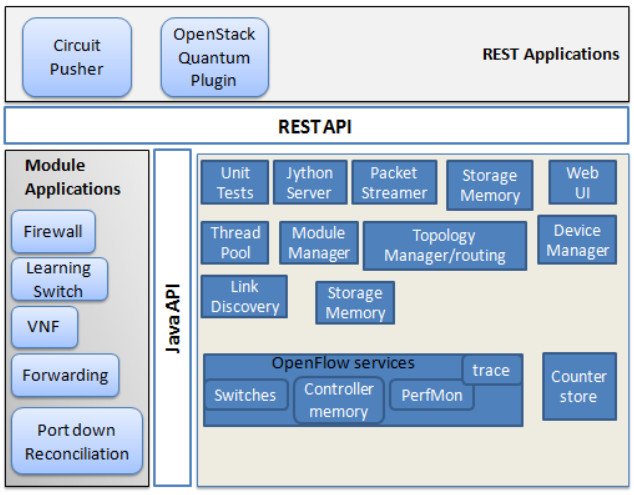

Figure. 4: the architecture of floodlight controller

There are two main approaches: based on how controllers are organized, centralized/decentralized architectures and reactive/proactive approaches. A centralized architecture, as illustrated in Figure 5-a, represents a single point of management for the entire network without proprietary forwarding. The route and data flow definitions are performed within that centralized controller. There are various

[7]http://www.projectfloodlight.org/

implementations of centralized architectures. For example, Ethane [11] is a centralized network architecture for the enterprise, it couples switches with a centralized controller that manages the permission and routing of flows. This central controller contains the global network policy that determines the destination of all packets; the forwarding elements must consult a controller each time a decision must be made. Ethane allows managers to define a single network wide fine-grain policy and then enforces it directly, but there are some scalability issues in larger networks, as the controller throughput must be able to handle the volume of new flow requests.

While centralized control planes offer the advantages of easy operations and management, it causes, however, problems concerning scalability and reliability. Unlike centralized controllers, distributed controllers (see Figure 5-b) have been proposed for SDN to address the scalability and reliability issues. There two main classes. The first class proposes to build a logically centralized control plane but physically distributed which can benefit from the scalability and reliability of the distributed architecture while preserving the simplicity of the centralized system. The main advantage is to have a consistent network wide view in all controllers while maintaining a logically centralized distribution. Examples of controller platforms are HyperFlow [12] and Onix [13].

In HyperFlow, scalability is guaranteed by enabling network operators to deploy any number of controllers. The main aim is to tune the performance of the control plane based on their needs, while keeping the network control logically centralized. However, this approach can only support global visibility of rare events, such as link state changes, and not frequent events, such as flow arrivals. Onix [13] is a control platform that can be implemented as a distributed system. It provides flexible distribution primitives allowing application designers to implement control applications without re-inventing distribution mechanisms, and while retaining the flexibility to make performance/scalability trade-offs as dictated by applications requirements.

The second class of approaches proposes a hierarchical distribution of controllers, such as Kandoo [14]. More precisely, as defined in Kandoo, the control plane is hierarchically structured into two levels: local controller executes local applications, and root controller runs non-local control applications. This structure allows: i) reducing the load on the global controller by filtering the number of new flow requests, and ii) providing the datapath with faster responses for requests, which can be handled by a local control application. Experiments have been performed and results show that Kandoo scales better than a normal OpenFlow implementation, without modifying switches or using sampling techniques.

However, in distributed architectures, the mapping between a switch and a controller is statically configured; which may result in uneven load distribution among the controllers. To address this issue, dynamic approaches are required. For example, a distributed controller architecture, named ElastiCon, is proposed in [15]. The controller Pool is dynamically grown or shrunk according to traffic conditions and the load is dynamically shifted across controllers. Authors present a novel switch migration protocol for enabling such load shifting, which conforms to the Openflow standard. They showed that their preliminary evaluation results are very encouraging.

In reactive approaches, the first packet of flow received by the switch implies the controller to install the flow entries in each network switch as shown in Figure 5-c. This approach presents the most effective use of existing flow table memory, but causes a small additional installation time for each new flow, given the dependence on the controller, if the switch loses the connection it cannot forward the flow [16]. Ethane is an example of the reactive control model [11], in which the controller contains the global network policy that determines the destination of all packets. Switches must consult the central controller whenever a decision must be made. The controller decides whether the flow represented by this packet should be allowed. Switches forward packets in accordance with the direction of the controller. A slight performance delay when the first packet of each new flow is forwarded to the controller for decision. The first-packet delay is negligible in many cases, but if the controller is geographically faraway, or if most flows are short-lived, it may be a concern.In proactive approaches, the controller pre-populates the flow table in each forwarding device as illustrated in Figure 5-d. In these approaches, the forwarding rules are defined already so it has zero additional flow setup time. In the case where the switch loses the connection with the controller, it does not disrupt traffic.

However, a hard management is required for the network operation. An example of a proactive approach is presented in DIFANE[17], in which the rules are distributed to authority switches that handles all data traffic in the fast path. Authors evaluates a DIFANE prototype, with various networks, and large sets of wildcard rules, in their evaluation they show that it is scalable with networks with a large number of hosts, flows, and rules.

Its worth noting that both approaches present some advantages and drawbacks. Authors, in [16], compared and analyzed the performance of both approaches using different OpenFlow controllers and switches. They stated that reactive approaches make easy the network operation and management, but evaluation results demonstrated the shortcoming of reactive approaches. Proactive approaches show their effectiveness when compared to reactive approaches. Therefore, hybrid approaches have been introduced to tackle some issues of reactive/proactive approaches [16].
In these approaches, a new intelligent switch was incorporated taking into consideration the performance of proactive approaches and the facility of reactive approaches. The controller acts like a learning switch without management interference but can perform a proactive set up using a reinforcement learning mechanism. The results presented in [16] show an improvement in controller performance due to control messages reduction. Table 2 presents some OpenFlow

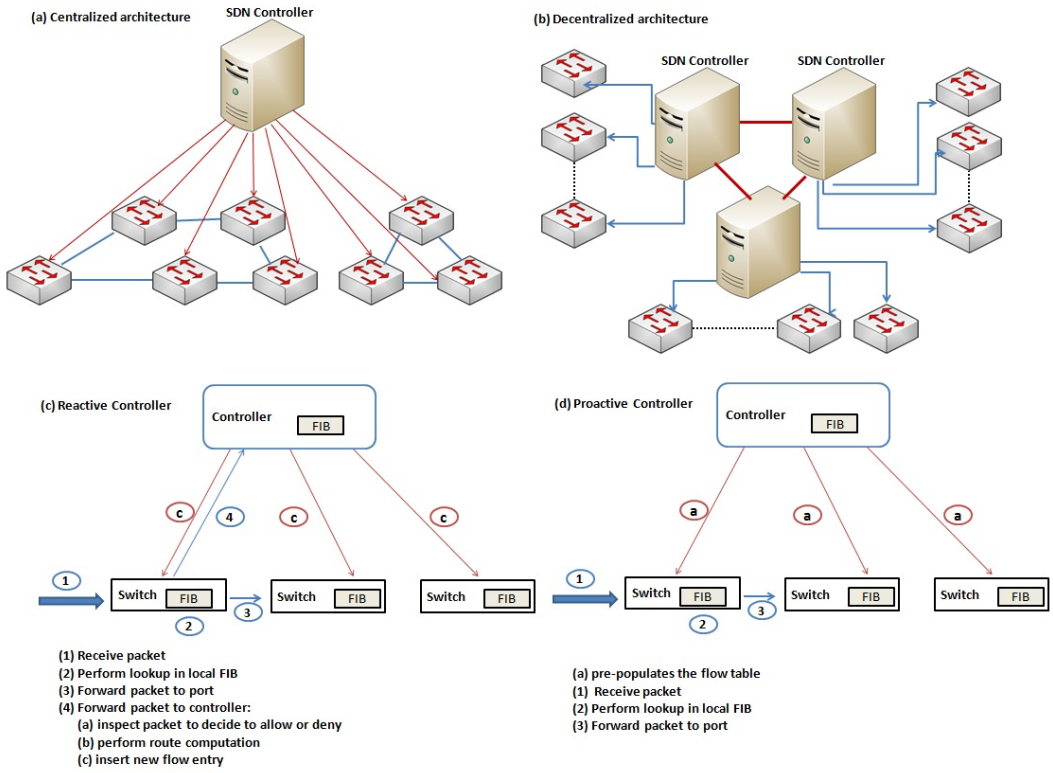

Figure. 5: a) Centralized architecture, b) decentralized architecture, c) reactive controller, d) proactive controller

Table 2: Comparing the results for the function optimization problems.

Model	implementation	Platform	Developer	Specification
NOX[19]	C++/Python	Linux	Nicira	The first OpenFlow controller
POX[20]	Python	Linux	Nicira	includ a queriable topology graph,
MUL[8]	C	Linux	Kulcloud	Supports a multi-level north bound interface
Beacon[21]	Java	Win, Mac, Linux, Android	David Erickson	Supports both event-based and threaded operation.
Trema[9]	C/Ruby	Linux	NEC	Framework for developing controllers in Ruby and C.
Floodlight[10]	Java	Win, Mac, Linux,	BigSwitch	Java-based OpenFlow Controller.

Controller implementations.

III. SDN Evaluation Tools and Platforms

Several tools and platforms have been recently developed to evaluate SDN architectures based on openflow protocol. These architectures can be either evaluated by experiments or by simulations. There are four main tools for evaluating SDN architectures based on OpenFlow protocol as follows. Mininet [18] is an example of network emulator tools that emulates any type of forwarding element, in terms of function and performance. It allows an entire OpenFlow network to be emulated on a single machine. New services, applications and protocols can first be developed and tested on an emulation of the anticipated deployment environment before moving to the actual hardware. By default Mininet supports OpenFlow v1.0. Ns-3 is a discrete-event network simulator that supports OpenFlow switches within its environment. It allows performing extensive simulations of large scale networks.

Platforms using SDN and Openflow features are Geni[11] and GANT[12] OpenFlow facilities. These platforms provide students and researchers to set-up experiments in a real-world environment. The aim is to test and set-up experiments, such as routing protocols, security models, addressing schemes, and to provide feedbacks based on experiments. For example, the GANT is the pan-European research and education network that interconnects Europes National Research and Education Networks. The OpenFlow facility of this backbone network is designed to support software defined networking experiments and prototyping. It is located in five GANT Points of Presence (PoPs) in the following cities: London, Frankfurt, Vienna, Zagreb, and Amsterdam. The OpenFlow switches are implemented by Open vSwitch. The host components of the testbed run as virtual machines on top of dedicated hypervisors applying XEN paravirtualization technique and connect to nodes via virtual links. Alike

[8] http://www.openmul.org/

[9] https://trema.github.io/trema/

[10] http://www.projectfloodlight.org/floodlight/

[11] http://geni.net

[12] http://www.geant.net

GEANT in Europe, GENI (Global Environment for Network Innovations) provides a virtual laboratory for networking and distributed systems research and education in USA. OpenFlow resources are available through GENI such as Switches based OpenFlow protocol.
Recently, in order to facilitate the adoption of SDN, industry recognizes the benefits of creating an open framework for programmability and control through an open source SDN and NFV (Network Functions Virtualization) solutions. Establishing an open source project in this way is designed to help and accelerate the development of technology, and enable prevalent adoption of SDN and create a solid foundation for NFV. Examples of these projects are Opendaylight[13] and Opencontrail[14]. Opendaylight is an open source project with a modular, pluggable, and flexible controller platform at its core, hosted by the Linux Foundation for network programmability to enable and accelerate adoption of SDN, and create a solid foundation for NFV. The opendaylight controller is implemented strictly in software. It can be deployed on any hardware and operating system platform that supports Java. The controller exposes open northbound APIs, which are used by applications. Opencontrail is a project that is built using standards-based protocols and provides all the necessary components for network virtualization SDN controller, virtual router, analytics engine, and published northbound APIs. It has an extensive REST API to configure and gather operational and analytics data from the system. OpenContrail can act as a fundamental network platform for cloud infrastructure.

IV. SDN Use Cases

SDN is a major trend in the industry and it promises to change how networks are built and managed. One of primary use case of SDN is Network Virtualization. Network virtualization is defined as a mechanism to isolate certain classes of traffic from other classes for regulatory conformity, administrative, or security reasons. SDN can provide a very flexible way of achieving network virtualization while simplifying operations, which promises to solve networking challenges around agility, resource utilization and on-demand cloud deployments in Enterprise data center and cloud providers.

An SDN approach to network virtualization can provide many benefits. For example SDN enables slicing network amongst universities to create logical isolated networks, and allow them to be partitioned using a technique, called slicing (as illustrated in Figure 6). As universities experience increasingly large numbers, such as students, faculty, researchers and staff, network slicing needs to be developed. University research projects typically communicate in isolated networks to perform their research and experiments within the campus backbone. However, the need for collaborating on a wider scale requires that isolated networks must grow within themselves and be able to communicate with other university research networks. Example of projects related to develop platforms using SDN and Openflow features as mentioned previously are GANT OpenFlow and Geni facilities. Universities can include proper production network management and control while networks expand to meet growing and varied requirements through higher automation and dynamic configuration.

Furthermore, SDN makes traffic isolation criteria flexible based on context of a flow (user, device, location, time, application, and potentially other external factors). For example in the case of virtual Multi-Tenant Data Center: Data centers are adopting virtualization technologies and providing diverse user services. Multi-tenant cloud data centers serve tenants who could belong to different organizations, companies, or institutions. Each tenant requires security separating their resources from those of other tenants and isolating the traffic of one virtual network instance from traffic of another supports multi-tenancy.

SDN centralizes, automates control, and provisions models that support the convergence of data, voice, and video. It can help tackling most challenges that an enterprise might suffer from when using current models. For example, using SDN paradigm can reduce the configuration complexity and can be much easier to manage, because the network is defined at software level, and makes easy to use policy to define traffic flows on the network. Similarly since SDN supports automated provisioning and management of network resources (determined by individual user profiles and application requirements) network provisioning becomes simpler and efficient. For example, Ethane [11] project focuses on using a centralized controller to manage policy and security at enterprise level.

Another interesting application of using SDN-based feature is to support a private or hybrid cloud environment. Hence, SDN allows network resources to be allocated in a highly elastic way, enabling rapid provisioning of cloud services and more flexible hand-off to external cloud providers. More precisely, SDN offers scalability and automation techniques required to implement a utility computing model for IT-as-a-Service. However, to provide cloud services via an SDN approach, one must pay close attention to cloud network orchestration, management, and automation. The orchestration system is responsible for provisioning all resources of a cloud data center, including computing, networking, and storage.

V. Key Challenges and Research Directions

There is numerous research issues that need to be tackled. For example, in[1], authors present a discussion of a number of challenges in the area of security, interoperability, and scalability. The main aim is to increase the performance of the network. It is worth noting that the performance of the network depends on the switch capabilities, hardware resources (e.g., CPU, memory), and software resources (e.g., operating system). The security is the main concern that is associated with SDN. Questions have been raised around authentication and authorization mechanisms to enable multiple organizations to access network resources while providing the appropriate protection of these resources. In the case of using SDN in Heterogeneous Networked

[13] http://www.opendaylight.org

[14] http://www.opencontrail.org

Environments as presented in [22]. Authors explore how SDN can be utilized to support both infrastructure-based wireless and infrastructure-less networks, as challenge involved with adapting the SDN model to these environments is security, i) in an infrastructure-less network end devices acting as forwarding nodes, so with independently-owned end devices, it may be difficult to establish trust and ensure a secure channel from end-to-end, ii) forwarding nodes need to be able to trust that the discovered controller is not malicious before accepting control, at the same the controller must be able to trust that forwarding nodes that have accepted control are correctly following instructions.

The interoperability is another issue that can be seen from different sides: i) how SDN will interoperate with traditional networks such as hybrid SDN and overlay network solutions, ii) communication between different controllers platforms need standard for communication, iii) how to orchestrate SDNs managed by different controller. Furthermore, the scalability has been a major concern since the introduction of SDN, either at switch level or at controller level. The scalability has been studied by researchers from several points of view, most of research work to date focus on controller scalability. For example, as stated in [23], three specific challenges are identified. The first is the latency introduced by exchanging network information between multiple nodes and a single controller. The second is how SDN controllers communicate with other controllers. The third challenge is the size and operation of the controller back-end database.

Several solutions to controller scalability have been proposed. For example, Onix platform [13] has been developed to facilitate the implementation of distributed control planes. It provides control applications with a set of general APIs to facilitate access to network state, which is distributed over Onix instances. HyperFlow [12] is another solution that aims to synchronize network state among multiple controller instances, giving the control applications (running on every controller instance) an illusion of control over the whole network. This keeps the simplicity of developing the control plane on a central controller while alleviating a class of scalability issues associated with a centralized controller. Kandoo [17] is another example that takes a different approach for distributing the control plane. A root controller takes charge of applications that require network-wide state, and also acts as a mediator for any coordination required between local controllers.

Authors in [23] studied scalability issue by mainly focusing on the increased load on the controller, flow initiation overhead, and resiliency to failures. They argue that the scalability is not only related to SDN, but also to the traditional networks and still a main challenge in emerging networking communication paradigms, such as cloud computing and the Internet-of-Things. Other researchers focused on providing techniques to improve the controller performance. As mentioned above there two main approaches: proactive and reactive. The proactive approaches improve the controller performance, since roles are already defined. In fact, switches do not need to send messages to the controller and then fewer messages will be received by the controller. However, proactive approaches are not adaptive to dynamic changes of the environment, and then require that a controller know the traffic flows in advance in order to configure the paths. In reactive approaches, roles are defined in real time. These approaches require less configuration effort and more adaptable to dynamic changes, however they require a longer decision time that reduces the controllers performance. Authors in [16] compared reactive and proactive approaches and reported results showed that proactive approaches outperform reactive approaches in terms of scalability.

Almost all approaches proposed so far addresses the scalability issue at the controller level, but in the best of our knowledge no research work has been done at the switch level. For example, the performance and scalability of the switch could be improved by optimizing the OpenFlow hardware design. Other issues related to the scalability of the SDN and need also to be tackled is for example, when given a network topology; the issue is to find the number of controllers and their emplacement. In [24], authors present an initial analysis of this problem, and they report that the answer to where and how many controllers to deploy depends on desired reaction bounds, metric choice(s), and the network topology itself. Other research work related to programming languages for SDNs, design of switches and controllers are also introduced and we refer readers to an addition reading list[15] for more details.

In our work we are addressing scalability and adaptability issue at controller level. In [25] we have introduced a hierarchical SDN control plane architecture that consists of three levels as network devices, local controllers that serve local requests and applications, and global controllers to perform control for no local requests. Controllers are organized into a hierarchical structure in order to distribute responsibility among them. Local controllers serve frequent events and the inter-domain control traffic is served by others. Each con-

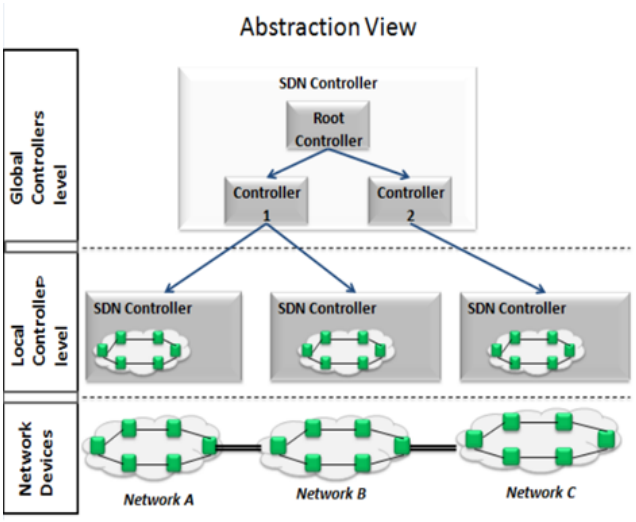

Figure. 6: The hierarchical control Plane architecture [25]

[15]http://www.nec-labs.com/ lume/sdn-reading-list.html

troller will be able to configure one another based on their positions in the hierarchy. The architecture as depicted in figure 7 use simple example: i) if a switch receives flow entries that are not present in its flow table, it sends request to local Controller, ii)Local Controller communicates with Global Controller to get rules to install for non-local flow, iii) Global Controller delegates the requests to the local controllers (Network A and Network B) to install flow-entries on switches, iv) Local Controllers install flow-entries in Switches:

- For local domain the rules in Switches are pushed proactively, while no local flows will be rerouted adaptively.
- If the Global Controller needs to install flow-entries on Switches in local domain, it delegates the requests to the respective local controller.
- For high availability, the superior controller in the hierarchy can register itself as the slave controller for a specific switch in case of failure. We have conducted experiments with Mininet, and preliminary results show that the proposed architecture reduce setup time of flow initiation time.

VI. Conclusion and Perspectives

In this paper, we provided an overview of approaches related to SDN and OpenFlow architectures and operations. We first introduced the SDN architecture and OpenFlow with a classification of exiting approaches, and different applications that require SDN capabilities and features. Almost in all research studies, hybrid approaches provide better performance. However, some key challenges and research directions are also presented. The scalability is one of most important issues that need to be tackled, either at the switch level or at the controller level. Recent solutions are promising, but the scalability still a remaining challenge for future emerging networking communication paradigms and related applications.

References

[1] B. Fraser, D. Lake, C. Systems, J. Finnegan, N. Viljoen, and S. O. E. N. Etworking, Are We Ready for SDN? Implementation Challenges for Software-Defined Networks, IEEE communication Magazine, pp. 3643, 2013.

[2] H.-J. K. Myung-Ki Shin, Ki-Hug Nam, Software-Defined Networking(SDN): A Reference Architecture and Open APIs, EEE International Conference on ICT Convergence (ICTC) pp. 360361, 2012.

[3] K. Bakshi, Considerations for Software Defined Networking (SDN): Approaches and Use Cases, IEEE Aerospace Conference pp. 19, 2013.

[4] B. Heller, OpenFlow Switch Specification, ONF pp. 136, 2009.

[5] Y. Luo, P. Cascon, E. Murray, and J. Ortega, Accelerating OpenFlow Switching with Network Processors, pp. 7071, 2009.

[6] D. Y. Huang, K. Yocum, and A. C. Snoeren, High-Fidelity Switch Models for Software-Defined Network Emulation, ACM workshop on Hot topics in software defined networking, pp. 43-48, 2013.

[7] R. N. Netronome, Selective and transparent acceleration of OpenFlow switches.

[8] A. Bianco, R. Birke, L. Giraudo, M. Palacin, D. Elettronica, and P. Torino, OpenFlow Switching: Data Plane Performance, pp.1-5, IEEE international conferences on (ICC) 2010.

[9] J. Naous, D. Erickson, and G. A. Covington, Implementing an OpenFlow Switch on the NetFPGA platform, ACM/IEEE Symposium on Architectures for Networking and Communications Systems, pp.1-9, 2008.

[10] O. N. Foundation, Outcomes of the Hybrid Working Group, March, 2013.

[11] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Mckeown, and S. Shenker, Ethane: Taking Control of the Enterprise, ACM conference on Applications, technologies, architectures, and protocols for computer communications, pp.1-12, 2007.

[12] A. Tootoonchian, HyperFlow: A Distributed Control Plane for OpenFlow,, ACM internet network management conference on Research on enterprise networking, 2010.

[13] T. Koponen, et al., Onix: A Distributed Control Platform for Large-scale Production Networks, USINEX conference on Operating systems design and implementation, ACM 2010.

[14] S. H. Yeganeh, Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications, ACM workshop on Hot topics in software defined networks 2012, pp. 1924, 2012.

[15] A. Dixit, F. Hao, S. Mukherjee, T. V Lakshman, and R. Kompella, Towards an Elastic Distributed SDN Controller, ACM SIGCOMM workshop on Hot topics in software defined networking, pp. 712,2013.

[16] M. P. Fernandez, Comparing OpenFlow Controller Paradigms Scalability: Reactive and Proactive, International Conference on Advanced Information Networking and Applications, pp. 1009 - 1016, 2013.

[17] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, Scalable flow-based networking with DIFANE, ACM SIGCOMM Comput. Commun, pp 351-362 2010.

[18] J. Sommers and P. Barford, Fast , Accurate Simulation for SDN Prototyping, in proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, pp 31-36, 2013.

[19] 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,, pp. 105110, 2008.

[20] S. Kaur, J. Singh, and N. S. Ghumman, Network Programmability Using POX Controller, Int. Conf. Commun. Comput. Syst., p. 5, 2014.

[21] D. Erickson, The Beacon OpenFlow Controller, 2013.

[22] M. Mendonca and S. Antipolis, The Case for Software Defined Networking in Heterogeneous Networked Environments, , ACM conference on CoNEXT student workshop, pp: 59-60, 2012.

[23] S. Yeganeh, On Scalability of SoftwareDefined Networking, IEEE Commun. Magazine, pp. 136141, 2013.

[24] B. Heller, R. Sherwood, and N. Mckeown, The Controller Placement Problem, ACM communication, pp. 473-478, 2012.

[25] Z. Safae, D. El Ouadghiri, and M. Bakhouya, Design and Evaluation of a Distributed SDN Control Plane Architecture, mediterranean telecommunication journal, pp. 9397, 2015.

Author Biographies

Safae Zerrik received her telecommunication and network engineering degree from ENSA Fez, Morocco. Currently, is a PhD student at LIA Laboratory, TAR Team, Faculty of science University Moulay Ismail, Meknes, Morocco. Her research interests include Networking protocols and virtualization

Mohamed Bakhouya is an associate professor-HDR at the International University of Rabat. He received his HDR (2013) from University of High Alsace and the Ph.D. (2005) degree in computer science and computer engineering from the UTBM, France. His research interests include various aspects on the design, validation, implementation, performance evaluation and analysis of distributed systems, architectures, protocols and services. He is IEEE and ACM member.

Driss el Ouadghiri is an associate professor at Science Faculty, Moulay Ismail University, Meknes, Morocco, since September 1994. He got his "License" in applied mathematics and his "Doctorat de Spcialit de Troisime Cycle" in computer networks, respectively, in 1992 and 1997 from Mohamed V University, Rabat, Morocco. In 2000 he got his PhD in performance evaluation in wide area networks from Moulay Ismail University, Meknes, Morocco. He is a founding member, in 2007, of a research group e-NGN (e-Next Generation Networks) for Africa and Middle East. His research interests focus on performance evaluation in networks (modelling and simulation), DiffServ architecture (mechanisms based active queue management). Since 2012, Mr. EL OUADGHIRI is focussed on the security and performance study of Vehicular Ad Hoc Networks (VANET) and Delay Tolerant Networks (DTN). He has About 40 publications in international journals and in proceedings of international conferences.