Software-Defined Networking: Management Requirements and Challenges

Juliano Araujo Wickboldt, Wanderson Paim de Jesus, Pedro Heleno Isolani, Cristiano Bonato Both, Juergen Rochol, and Lisandro Zambenedetti Granville

The authors are with Federal University of Rio Grande do Sul (UFRGS).

ABSTRACT

SDN is an emerging paradigm currently evidenced as a new driving force in the general area of computer networks. Many investigations have been carried out in the last few years about the benefits and drawbacks in adopting SDN. However, there are few discussions on how to manage networks based on this new paradigm. This article contributes to this discussion by identifying some of the main management requirements of SDN. Moreover, we describe current proposals and highlight major challenges that need to be addressed to allow wide adoption of the paradigm and related technology.

INTRODUCTION

Software-Defined Networking (SDN) is a network paradigm usually characterized by three fundamental aspects:
- A clear separation of network forwarding and control planes.
- The abstraction of the network logic from hardware implementation into software.
- The presence of a network controller that coordinates the forwarding decisions of network devices.

Given that software, in SDN, can be more easily coded, deployed, and executed, SDN turns out to be a very disruptive technology that better promotes network innovation. In addition, SDN has been grabbing the attention of both industry and academia, and has experienced strong support by major Internet players (e.g. Google, Cisco, NEC, Juniper) and standardization bodies (e.g. ONF and IETF). Today SDN is a driving force in the field of computer networks.

Much has been discussed about SDN and network management, mostly from the perspective of where SDN is taken as a management tool [1]. There are several benefits, from the traditional network management point-of-view, in adopting SDN because it simplifies or even solves critical management tasks. For example, because SDN devices need to be registered or discovered by the network controller in order to establish a communication path between control and forwarding planes, network discovery, a traditional network management task, is intrinsically solved.

In this article we take a different perspective on SDN and network management. Although SDN does solve some classical management problems, it also creates new ones. This fact turns out to be a traditional "meta-problem" in the area of network management: every time a new network technology is introduced, the importance of its management is generally underestimated. When the technology matures and is widely deployed, network management arises as a real need. However, because management did not evolve together with the newly introduced networking technology, frequently network management becomes just a technology patch. As such, addressing SDN management is imperative to avoid patching SDN later.

In this article we address SDN management requirements to foster the adoption and development of the topic. We investigate to what extent current proposals for SDN management suffice, review how research in the area has evolved, and finally highlight management challenges that arise from the current picture of SDN management. The remainder of this article is organized as follows. In the next section we present the concepts and evolution of SDN. We then define management requirements of SDN. After that, current proposals for SDN management are presented. Next, challenges on SDN management are discussed. Finally, we summarize the article, presenting our final remarks.

SDN: EVOLUTION AND CONCEPTS

This section presents an evolution timeline and a review of SDN concepts. It is important to emphasize that SDN concepts are still under formation. Nevertheless, from the interest and especially given the significant investments made by the industry, one can safely state that this new network paradigm has a lot of potential to succeed. From one perspective, it is

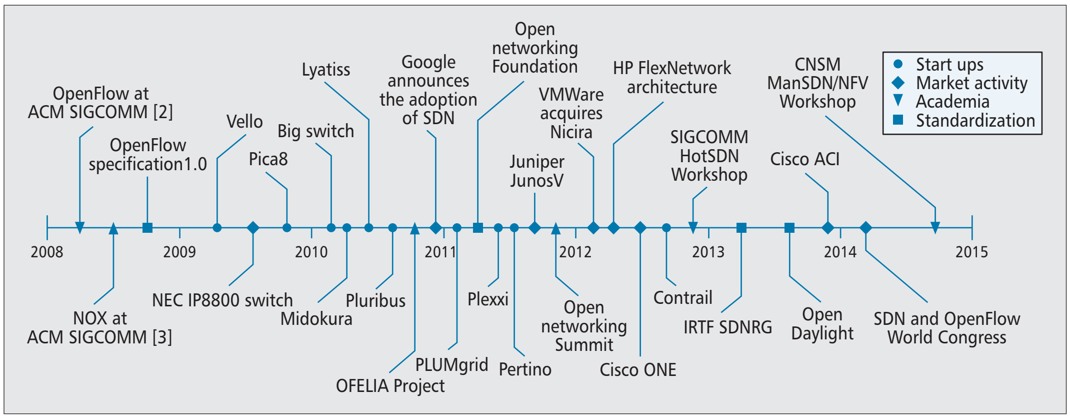

Figure 1. Major events on the SDN timeline over the last seven years.

Academia has played an important role in this evolution, early during SDN conception, during recent last years maturing the concepts, and regularly stimulating discussions through research projects, such as the development of the OFELIA testbed in Europe.

impressive how SDN is able to foster innovation in networks at a level that has not been seen in the last decade. On the other hand, companies are usually too concerned with time to market rather than providing conceptually elegant solutions. Particularly, the main issue we envision in this scenario is that the majority of efforts are focused on providing solutions and services over SDN networks, whereas network management is not given much attention. However, as widely recognized, management must not be an afterthought [2].

Evolution Timeline

In order to observe the evolution of SDN from the perspective of both industry and academia, we present a timeline containing some of the major events in this area (Fig. 1). Our investigation begins in the earliest phase of OpenFlow [3], arguably the most important SDN implementation today. We did not include pre-SDN proposals, such as ETHANE, because the concept was not then referred to as SDN yet. Moreover, that period has already been discussed [4]. OpenFlow was first introduced in a white paper in March 2008, from the network research group of Stanford University and collaborators. Given the importance of the work, the paper appeared as an editorial note in *ACM SIGCOMM Computer Communication Review* one month after that. The objective back then was simple: to enable researchers to innovate, and try out their proposals, within a campus network. The first time the concept of "network operating system" appeared in this context was through the proposal of the NOX controller [5]. NOX represents the first attempt to enable the development of software to control OpenFlow-based networks. In the context of SDN, such a development task may turn out to be a common responsibility among software developers and network administrators.

Right after launching the first specification of OpenFlow, with official vendor support,[1] at the end of 2009, there was much excitement in the networking market. NEC was the first to announce a commercial switch with native OpenFlow support, in April 2009. Also, start-up companies, such as Pica8, Big Switch, Plexxi, and Vello began to pop out offering SDN-ready solutions. In 2011 some Internet giants, such as Google, announced the adoption of SDN inside their data centers and backbones. In the same year a few standardization efforts also started. ONF is basically dedicated to promote and evolve the concept of SDN by standardizing the OpenFlow technology. Near 2012 other major players such as Cisco, Juniper, Hewlett-Packard, and VMWare rushed to put themselves in a leading position in that promising but not yet matured networking paradigm. In the beginning of 2013 the IRTF started the Software-Defined Networking Research Group (SDNRG), while the Linux Foundation launched the Open Daylight project. All these standardization efforts are supposed to be vendor independent, although most of the aforementioned companies actually embody the majority of the working groups and committees. Academia has played an important role in this evolution, early during SDN conception, in recent years maturing the concepts, and regularly stimulating discussions through research projects, such as the development of the OFELIA testbed in Europe, and conferences, such as the Open Networking Summit, the SIGCOMM HotSDN Workshop, and CNSM ManSDN/NFV Workshop.

Revisiting SDN Concepts

SDN and its properties have been defined by several authors in the last few years. SDN is commonly defined as a new networking paradigm in which the forwarding plane is decoupled from the control plane [5]. In addition, most of the "intelligence" of an SDN network is implemented inside a controlling entity, while network devices (switches) are only simple packet forwarding boxes. In fact, this definition is biased by the association of the concepts of SDN with its main enabling technology, that is, OpenFlow. Separating the forwarding and control planes and adding a logically centralized controlling entity are two characteristics that have often appeared in other types of networks.[2,3] This rather vague definition leaves the doors open for divergent SDN instantiations. As a consequence, many emergent network solutions propose a variety of concepts that have significant overlap with this definition, such as Software-Friendly Networks, Software-Driven Networks, Deeply Programmable Networks, and Network Functions Virtualization.

[1] First specification available online is 0.8.9 from December 2008, although the first for official vendor support was 1.0.0, released in the last day of 2009.

[2] IETF RFC 3654 — http://tools.ietf.org/html/rfc3654 3

[3] GPP TS 23.002 GSM Network architecture — http://www.3gpp.org/DynaReport/23002.htm.

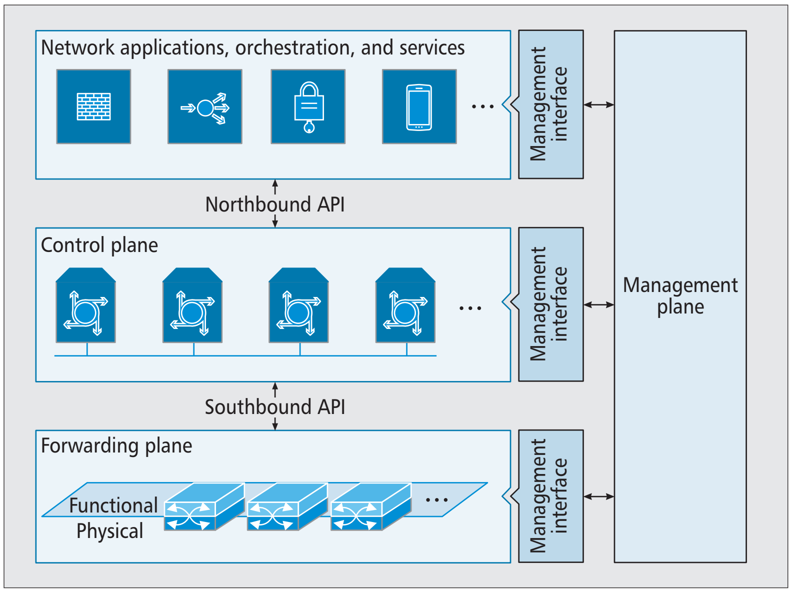

Figure 2. High-level conceptual architecture of SDN.

We look at SDN from a slightly different angle than many other authors [1, 3, 5]. We consider that SDN is essentially about abstracting the logic of traditional networks from within the fixed hardware implementation, thus raising it to a higher level defined by software. Separating the logic of the forwarding and control planes and laying down a network controller entity are also important concepts concerning architectural organization. However, most fundamentally, software within all planes needs to dictate how traffic is handled in the network.

Observing the state-of-the-art in SDN, we are able to depict a general architecture composed of four planes and three interfaces, as depicted in Fig. 2. Observing the architecture from a bottom-up approach, the forwarding plane is split into "functional" and "physical." The functional part will always be a collection of software functions to be executed in forwarding devices. This functional part is the reason why we call this the "forwarding plane" instead of "data plane," as many other authors do. From our point-of-view, the functional part does not only hold data, but it must allow the use of software to perform forwarding tasks, employing tables, trees, queues, or any other internal data structure. The remaining elements in this plane are strictly physical, such as I/O ports, memory, processor, and storage.

The control plane is originally meant to compile higher-level decisions and enforce the necessary configurations into forwarding devices. Elements in the control plane should at least execute requests coming from the plane above. Commonly, these elements also include internal logic to handle network events and recover from failures. Environment-specific demands might dictate how this plane should be designed, defining the best strategies for locating and distributing elements. On the top level of the SDN architecture, network applications, orchestration functions for business and network logic, as well as high-level network services are found, for example, load balancers and firewalls. We take a generic approach, including in the architecture a management plane. This plane rarely appeared in early SDN proposals, although it is being gradually considered in recent specifications.[4] We consider the management plane a fundamental part of the architecture to organize the implementation of operations, administration, and management (OAM) functions in SDN.

To allow information to flow between the management plane and other planes, management interfaces are in place. These interfaces allow settings to be enforced in network devices and information to be retrieved back into the management plane, for example, to be reported to the network administrator. In addition, two other important interfaces exist: a southbound API that allows the forwarding plane to communicate with the control plane, and a northbound API that abstracts control plane functions to network applications at the top level.

[4] SDN Architecture 1.0 (June, 2014) — https://www.opennetworking.org/sdn-resources/sdn-library/technical-papers.

Management Requirements of SDN

Given the current noticeable paradigm shift to SDN and its new concepts, the development of a management plane as presented in Fig. 2 encompasses satisfying new management requirements. In this section we discuss some of the most relevant requirements considering all planes of the SDN architecture. We also summarize such requirements in Table 1, providing a comparison with traditional networks and management activities examples.

Bootstrap and Configuration

In traditional networks the logic of the forwarding and control planes is confined inside each network device according to a well defined set of standardized protocols. With the separation of planes, as SDN promotes, the need to bootstrap the communication between the forwarding and control planes becomes a basic requirement. Configuring this communication can be particularly complex, considering that both planes can operate under protocols defined by software. Moreover, software changes in any plane may affect such communication directly. Ideally, new software-defined protocols should include management interfaces to enable proper bootstrap and configuration in these networks.

Availability and Resilience

As in any other network, SDN is susceptible to faults in physical or logical elements, for example, link failures or network software bugs. Particularly, with the decoupling of planes comes the possibility of eventual disconnection between the forwarding and control planes. Many approaches to SDN even consider placing the whole control plane inside a single node, in which case unavailability and resilience issues become even more critical because of a single point of failure. Despite the fact that forwarding devices are usually able to react to this issue, it is still important to manage whether the connection between planes is active and in accordance with the policies of the network.

Management requirement	Traditional networks	Software-defined networking
Bootstrap and configuration	Set well known protocol parameters, track configuration changes	Configure customized and ever-changing software, setup forwarding and control plane connectivity
Availability and resilience	Configure alternative routes in case of link failure	Configure forwarding devices behavior in case of failure in the connection with control plane
Network programmability	Not required	Control versioning, coordinated deployment, and verification of network software
Performance and scalability	Bandwidth assignment and reservation, Quality of Service configuration and enforcement	Monitor performance of network applications, adjust connection quality between forwarding and control planes
Isolation and security	Control network access, prevent intrusion, spoofing and Denial of Service	Grant isolation to network applications, prevent eavesdropping and usurpation of control traffic
Flexibility and decoupling	Adjust the management of higher level protocols	Adapt management functions along with management interfaces, coordinate management information within planes or among management systems
Network planning	Assess capacity and performance needs, choose a network topology	Plan the disposition of controlling elements in relation to forwarding elements
Monitoring and visualization	Track resource utilization, identify outages and trigger alarms	Trace functional parameters of novel applications, visualize jurisdiction of network controllers

Table 1. Traditional versus software-defined networking management activities.

Network Programmability

In the daily life of SDN administrators, every new network software release or update must be consistently persisted over the forwarding and control plane implementations across the network. To provide proper support for network programmability management, tools to allow network administrators to control versioning, coordinated deployment, rollback, and verification of network software are required. Moreover, software to dictate network behavior can also be developed in a variety of abstraction levels (high or application level, mid or control plane level, and low or forwarding plane level). Tools and methods need to be designed to decompose these high/mid-level policies or commands down to low-level device configurations.

Performance and Scalability

In SDN, network hardware needs to be more generic to allow software to be written in high levels of abstraction. Moreover, the separation of planes itself implies extra communication requirements between planes, which may add delay to network traffic being forwarded. Part of the responsibility of performance assurance falls on the shoulders of software developers that need to design optimized software for networks. Another part resides on network administrator's effort to understand bottlenecks, tweak the correct parameters to optimize software-defined protocols, and choose the most efficient control and management models to be employed (e.g. centralized, distributed, or hierarchical).

Isolation and Security

In SDN not only the network traffic is shared among many different users and applications, but also the network logic itself is controlled remotely by custom software. New resource isolation and security management techniques must be investigated to guarantee isolation in the three SDN planes as well as in their communications. Specifically, one must ensure that control traffic (flowing between controllers and forwarding devices) is isolated, and that code written by one developer does not affect the code written by others (e.g. by defining conflicting rules or resource race conditions). As SDN becomes more popular, vulnerabilities will appear and soon enough people will start writing the first malicious code for these networks.

Flexibility and Decoupling

Interfaces are required to allow exchange of management information from/to forwarding, control, and application planes. Since the behavior of the network can now be defined by software, SDN management needs to be flexible enough to quickly adapt to new protocols written for all planes. For example, one could implement a novel protocol for Information Centric Networking (ICN), which routes information relying on a caching system inside network devices. Managing cache performance (e.g. miss rates, occupation, object sizes) is a function that should be present in the management plane to allow administrators to adjust this protocol-specific parameter to optimize the network.

Network Planning

Traditionally, administrators need to plan for both deployment and expansion of networks, which encompasses tasks such as defining capacity and performance needs and deciding whether and where in the topology the network will be segmented (on layers 2 and 3). These decisions will result into acquiring a set of network hard-

Positioning these elements across the network topology can influence directly the performance, resilience, and survivability. In addition, the management plane could assist the SDN administrator in making planning decisions through a set of planning tools deployed in the SDN management frontend.

ware boxes (e.g. routers, switches, and firewalls) that operate under well known standard protocols. Planning is not a requirement for the SDN management plane per se, but it impacts and influences the other requirements. In SDN, administrators now need to acquire a set of forwarding devices and possibly another set of controlling devices. On top of that, network topology and capacity planning can be influenced by the type of software one installs on the network, also requiring software acquisitions. Administrators are still able to choose from different vendors of network software, controlling devices, and forwarding devices. Positioning these elements across the network topology can directly influence the performance, resilience, and survivability. In addition, the management plane could assist the SDN administrator in making planning decisions through a set of planning tools deployed in the SDN management frontend.

Monitoring and Visualization

For the physical part, monitoring and visualization requirements remain similar to traditional networks. The logical part is far more complex assuming that forwarding and control protocols can be completely redesigned. For example, in current IP networks it is easy to draw a reachability map by analyzing information of routing tables, even before traffic starts flowing. This analysis is only possible because forwarding behavior of IP routers is predictable. When the internal implementation of a protocol is not known in advance, or defined by software, that is no longer possible.

Current Management Proposals for SDN

In this section we briefly review recent proposals that deal with some of the requirements presented in the previous section. We check whether these requirements are addressed fully, partially, or not addressed at all by current efforts. The proposals are grouped according to the SDN architectural part, being described starting by the forwarding plane, followed by the control plane, application plane, and finally the communication between planes. Since most proposals do not include explicitly a management plane in their conceptual architectures, we left this out of the organization of this section.

Given the need to deploy and set up novel forwarding devices into a traditional network, administrators must configure and bootstrap these devices in order to have them operational. In the most basic and common scenario, the administrator would interact with a command line interface of the device and have it configured by performing many intricate proprietary commands. In the case of SDN forwarding devices, there are still a few parameters to be configured, such as the communication policy with the control plane (e.g. drop every packet when the control plane is unavailable or follow the last valid set of rules). In response to these necessities, ONF proposed the OpenFlow Management and Configuration Protocol (OF-Config).[5] This protocol is based on NETCONF and allows operational staff to assign controller(s) to switches, set ports up/down, configure queues, assign certificates for the communication with the control plane, set up tunnels, handle versioning, and retrieve device capabilities. OF-Config is already a significant step toward tackling forwarding plane-related management requirements, such as *bootstrap and configuration* and *availability and resilience*. On the other hand, this protocol is targeted specifically to OpenFlow networks; therefore, it is tied to the limited view of SDN employed by this technology (e.g. fixed forwarding plane and logically centralized control plane).

The loose coupling between the forwarding and control planes in SDN allows greater flexibility in relation to the design and organization of elements in the control plane. Thus, the SDN control plane can be engineered with the strategy that best addresses the demands of users, such as availability, performance, and security demands. Proposals such as NOX [5], ONIX [6], HyperFlow [7], and Kandoo [8] are examples of different control plane implementations. While NOX adopts a simpler centralized strategy, ONIX, HyperFlow, and Kandoo employ distributed or hierarchical approaches where multiple controllers cooperate to control the underlying forwarding devices. These approaches are directly aligned with the *availability and resilience* requirement, and marginally with *performance and optimization*, and not aligned with *flexibility and decoupling*. To fit this last requirement, proposals should implement management interfaces as depicted in Fig. 2.

Network protocols and services, traditionally placed inside each network device, are now centralized in a loosely coupled way into the applications plane. Thus, tools previously employed to manage network services are incompatible with this new environment. Such tools must be redesigned under different requirements, such as *network programmability*, *isolation and security*, *monitoring and visualization*, and *flexibility and decoupling*. An attempt to address such requirements was led by RouteFlow [9], which allows traditional network protocols to run into virtualized network environments, although the routing decisions are still enforced in the underlying network using OpenFlow forwarding rules. Besides grouping network applications coherently, as proposed in RouteFlow, it is important to enable network administrators to manage applications.

SDN applications require high-level network abstractions, ideally presented through standardized interfaces, to communicate with elements in the control plane. A step toward meeting this requirement was given by SDN programming languages, such as Frenetic, Nettle, NetCore, Procera, and Pyretic [10]. These languages are predominantly declarative, but vary in objective, which includes efficiently expressing packet-forwarding policies, handling overlapping or conflicting rules, and improving horizontal scalability. From the operational point of view, SDN programming languages contribute to streamline the software development process, but to achieve *flexibility and decoupling*, developers must provide external interfaces to allow the execution of management tasks.

[5] OpenFlow Management and Configuration Protocol 1.2 (OF-Config 1.2) https://www.opennetworking.org/sdnresources/onf-specifications/openflow-config.

As discussed within the *availability and resilience* requirement, it is of fundamental importance to ensure the correct communication between the forwarding and control planes. FlowVisor [11] and its variations AdVisor [12] and VeRTIGO [13] are examples of tools that operate as proxies for delivering control information to different SDN controllers according to predefined rules. The employment of such tools allows network administrators to split the control logic of a network into several independent controlling devices. AdVisor, specifically, proposes the use of a management interface to control the rules used to forward control messages. Because of insufficiency of management interfaces and restricted management target, that is, only applied to communication between forwarding and control planes, these approaches are considered just partially aligned with the requirements of *flexibility and decoupling* and *monitoring and visualization*.

It is important to highlight that the aforementioned proposals have not been designed considering our list of management requirements, thus the mapping of the scope of each proposal to each requirement is not always direct. Still, one can now better observe how the state-of-the-art covers the listed management requirements.

Key Research Challenges in SDN Management

In this section we identify and discuss a non-exhaustive set of research challenges for SDN management. Instead of listing challenges that would cover the FCAPS (fault, configuration, accounting, performance, security) model, widely employed in network management in general, these challenges are a result of our analysis of the management requirements, and the coverage of proposals, discussed previously. Therefore, we highlight the main gaps found between requirements and proposals for SDN management.

From High-Level Rules to Network Configuration

Lower-level SDN planes introduce their own software abstractions and expose APIs to handle these abstractions to the upper-level planes. Every time the level of abstraction is raised, there is loss of information because not all low-level details are accessible through the defined APIs. When very high-level commands or rules come from the topmost planes of the SDN architecture, the lost information needs to be reconstructed or translated into a set of lower-level actions. To properly tackle the *bootstrap and configuration* requirement, processes for translating high-level rules into low-level configurations become a challenge. Although the vast literature on policy-based network management (PBNM) presents a solid base in this regard, further and SDN-focused research is required to tackle this challenge. We envision as a challenge, for the management plane of SDN, dealing with issues, such as: How is this translation of high-level rules (e.g. reduce latency, increase link redundancy, or save energy) realized in SDN-specific setups? How is the per-level rule refinement performed considering that the target infrastructure operates using SDN? Do SDN peculiarities generate conflicts that should be handled in high-level rules, and how should these conflicts be solved taking into account the low-level SDN details?

Lower level SDN planes introduce their own software abstractions and expose APIs to handle these abstractions to the upper level planes. Every time the level of abstraction is raised, there is loss of information because not all low-level details are accessible through the defined APIs.

Autonomic and In-Network Management

Ideally, a management plane should concentrate all network management functions of an SDN. Nevertheless, since SDN allows software to dictate the behavior of network devices, it should be possible to place code inside these devices to allow them to react to network conditions and perform self-management functionalities. Autonomic and in-network management approaches offer the capability to migrate management functions, usually deployed in the management plane, to software running on devices of both forwarding or control planes. Autonomic and in-network are also important because they help addressing the requirement of *availability and resilience* in SDN. These approaches are generally considered in scenarios where the amount of devices or network connectivity conditions are unsuitable to maintain a frequent communication between devices and the management plane. In these scenarios, for instance, the implementation of fast failover capabilities (e.g. self-healing) directly in network devices becomes essential. Therefore, one important question to be answered is: What are the criteria to define where in the network or in which plane should autonomic loops be deployed and controlled?

Flexible Management Through Interfaces

In the SDN architecture, the management plane interacts with other planes through the use of interfaces. Adequate management interfaces are also required, for example, to enable a more integrated view of network resources with the system running on top of them, such as in cloud computing environments. To cope with the *flexibility and decoupling* requirement, the challenge of finding a systematic way of defining, using, and evolving management interfaces arises. Defining which management information flows where, when, and how encompasses answering a number of questions, such as: Who is the person responsible for writing the required interfaces, that is, the developer, the administrators, or both? Is it possible to make interfaces generic enough to be used or reused when a new software is developed and installed in networks? Can we take advantage of standards for interfaces already proposed or do we need new standards? Is there the need for an intermediary element to host these interfaces, such as a gateway?

Smart Network Planning

In SDN, administrators must consider three main aspects when planning a network infrastructure:

1. The physical separation of planes.
2. The software abstractions that drive network logic.
3. The existence of one or more controlling entities.

In these times of "networking democracy", a crucial opportunity presents itself to addressing management requirements, and to avoid the recurrent mistake of patching management solutions after other concepts are already mature.

Regarding aspects 1 and 3, it is fundamental to consider that the number of elements in every plane can be arbitrary, as presented in Fig. 2. Moreover, considering aspect 2, choosing the right applications for networks becomes an essential and non-trivial task. In addition to being a requirement in SDN, planning is a challenge also because when virtualization is employed in SDN to support different networks on top of a single substrate, the amount of information the network administrator needs to handle is much larger in comparison with a non-virtualized network. Smart planning solutions become necessary to assist the network administrators with questions that were uncommon before the inception of SDN. Therefore, interesting questions need to be answered by SDN planning management solutions, such as: How many forwarding and controlling devices should be acquired to deploy the network? Where exactly should controlling devices be placed physically in relation to forwarding ones in the topology? How are control and management responsibilities organized (e.g. centralized, distributed, hierarchical)? How do software choices affect the physical topology? Are there compatibility issues among different software that need to coexist to form the deployed SDN?

Situational Management

In SDN the network conditions are much more dynamic because of facilitated, frequent software installation and changes. As a consequence, this is also more likely to create situational issues, which are not predicted in the design of management systems. To deal with unexpected and temporary conditions, such as debugging a newly installed software protocol, tools need to be available for fast creation of on-demand management applications taking advantage of information available from SDN planes and interfaces. Adequate situation management also encompasses fulfilling the requirement of *monitoring and visualization* of SDN [14]. Enabling situational management in SDN includes finding proper answers to: Which technologies are most suitable for creation of temporary management applications? Which techniques can be employed to combine information from multiple levels of the SDN architecture into on-demand management applications?

Conclusion

Despite being a relatively recent networking paradigm, the importance of SDN is already evidenced by the emergence of many start-up companies and foundations, the interest of researchers, and the support of major Internet players. Much has been recently discussed on taking SDN as a tool to simplify some classical network management issues. However, in this article we took a different perspective by analyzing the management necessities that did not exist before the inception of SDN.

Initially we revisited the discussions about the definition of SDN regarding concepts such as separation of planes and the actual implementation of a control plane. We approached SDN from a slightly different angle than many other authors, in which we emphasize the fact that SDN is essentially about abstracting network logic from hardware implementation to software. We also provided evidence that SDN currently overlaps with other emerging related concepts, such as Network Functions Virtualization and Software-Friendly Networks. Furthermore, we included in the SDN architecture a conceptual plane dedicated to implementing management functions, either in a centralized or distributed manner.

Aiming to encourage forthcoming proposals to take into account management concerns at the design phase, we discussed a non-exhaustive set of SDN management requirements. We kept our requirements irrespective to any technology and generic enough to remain valid over different SDN deployment designs. Moreover, we also discussed some of the most important investigations toward management of SDN already proposed and to which extent these investigations consider, directly or indirectly, one or more management requirements.

Finally, we established a set of challenges that we consider a fundamental contribution to encourage future investigations regarding SDN management. We envision mainly the resurgence of traditional network management concepts, such as autonomic/self-management and policy-based network management. Moreover, we believe in the empowerment of situational management, for example, based on mashup-oriented technologies. Most importantly, we understand that SDN represents a landmark: for the first time in decades we are witnessing computer network development happening outside private industry boundaries. In these times of "networking democracy" a crucial opportunity presents itself to address management requirements, and to avoid the recurrent mistake of patching management solutions after other concepts are already mature.

References

[1] H. Kim and N. Feamster, "Improving Network Management with Software Defined Networking," *IEEE Commun. Mag.*, vol. 51, no. 2, Feb. 2013, pp. 114–19.

[2] J. Schonwalder *et al.*, "Future Internet = Content + Services + Management," *IEEE Commun. Mag.*, vol. 47, no. 7, July 2009, pp. 27–33.

[3] N. McKeown *et al.*, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, April 2008, pp. 69–74.

[4] 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, Apr. 2014, pp. 87–98.

[5] N. Gude *et al.*, "NOX: Towards an Operating System for Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, July 2008, pp. 105–10.

[6] T. Koponen *et al.*, "ONIX: A Distributed Control Platform for Large-Scale Production Networks," *Proc. 9th USENIX Conference on Operating Systems Design and Implementation*, 2010, pp. 1–6.

[7] A. Tootoonchian and Y. Ganjali, "HyperFlow: A Distributed Control Plane for OpenFlow," *Proc. Internet Network Management Conference on Research on Enterprise Networking*, 2010, pp. 3–3.

[8] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications," *Proc. 1st workshop on Hot Topics in Software Defined Networks*, 2012, pp. 19–24.

[9] M. R. Nascimento *et al.*, "Virtual Routers as a Service: The Routeflow Approach Leveraging Software-Defined Networks," *Proc. 6th Int'l. Conf. Future Internet Technologies, ser. CFI '11*, New York, NY, USA: ACM, 2011, pp. 34–37.

[10] C. E. Rothenberg *et al.*, "When Open Source Meets Network Control Planes," *IEEE Computer Mag.*, vol. 47, no. 11, Nov. 2014, pp. 46–54.
[11] R. Sherwood *et al.*, "FlowVisor: A Network Virtualization Layer," *OpenFlow Switch Consortium, Tech. Rep.*, 2009.
[12] E. Salvadori *et al.*, "Demonstrating Generalized Virtual Topologies in an OpenFlow Network," *ACM SIGCOMM Computer Commun. Rev.*, vol. 41, no. 4, Aug. 2011, pp. 458–59.
[13] R. Corin *et al.*, "VeRTIGO: Network Virtualization and Beyond," Proc. European Workshop on Software Defined Networking, 2012, pp. 24–29.
[14] O. M. C. Rendon *et al.*, "Monitoring Virtual Nodes using Mashups," *Elsevier Computer Networks*, vol. 64, May 2014, pp. 55–70.

BIOGRAPHIES

JULIANO ARAUJO WICKBOLDT (jwickboldt@inf.ufrgs.br) is a Ph.D. student at the Federal University of Rio Grande do Sul (UFRGS) in Brazil. He achieved his B.Sc. degree in computer science at Pontifical Catholic University of Rio Grande do Sul in 2006. He also holds an M.Sc. degree from UFRGS conducted in a joint project with HP Labs Bristol and Palo Alto. Juliano was an intern at NEC Labs Europe in Heidelberg, Germany for one year between 2011 and 2012. Between 2013 and 2014 Juliano was a substitute professor at UFRGS. His current research interests include cloud resource management and software-defined networking.

WANDERSON PAIM DE JESUS (wpjesus@inf.ufrgs.br) coordinates research and development projects at the National Education and Research Network (RNP) in Brazil. He is also an MBA student in strategic management of information technology at Fundao Getulho Vargas (FGV). Wanderson achieved his M.Sc. degree at the Federal University of Rio Grande do Sul (UFRGS) in 2013, and his B.Sc. at the Federal University of Gois (UFG) in 2010, both in computer science. He is currently committed to foster industry-academia partnerships around information and communication technology. His research interests include software-defined networking and cloud computing.

PEDRO HELENO ISOLANI (phisolani@inf.ufrgs.br) is an M.Sc. student in computer networks at the Federal University of Rio Grande do Sul (UFRGS) in Brazil. He achieved his bachelor's degree in information systems at the State University of Santa Catarina (UDESC) in 2012. During the undergraduate years he worked on scientific initiation research in the management of learning objects development process. Currently, his research interests are related to the management of software-defined networking, more specifically monitoring, visualization, and configuration management activities.

CRISTIANO BONATO BOTH (cbboth@inf.ufrgs.br) is an associate professor at the University of Santa Cruz do Sul, Brazil. He received his Ph.D. degree in computer science from UFRGS in 2011. He received his M.Sc. degree in computer science from the Pontifical Catholic University of Rio Grande do Sul, Brazil, in 2003. His research interests include wireless networks, next generation networks, and traffic control on broadband computer networks.

JUERGEN ROCHOL (juergen@inf.ufrgs.br) is an emeritus professor at the Institute of Informatics of the Federal University of Rio Grande do Sul (UFRGS), Brazil. He received his M.Sc. degree in physics and his Ph.D. degree in computer science, both from UFRGS, in 1972 and 2001, respectively. His research interests include wireless networks, next generation networks, optical networks, and traffic control on broadband computer networks.

LISANDRO ZAMBENEDETTI GRANVILLE (granville@inf.ufrgs.br) is an associate professor at the Institute of Informatics of the Federal University of Rio Grande do Sul (UFRGS), Brazil. He received his M.Sc. and Ph.D. degrees, both in computer science, from UFRGS in 1998 and 2001, respectively. Lisandro has served as a TPC co-chair of IFIP/IEEE DSOM 2007, IFIP/IEEE NOMS 2010, TPC vice-chair of CNSM 2010, and general co-chair of CNSM 2014. He is also chair of the IEEE Communications Society Committee on Network Operations and Management (CNOM), and co-chair of the Network Management Research Group (NMRG) of the Internet Research Task Force (IRTF). His areas of interest include management of virtualization for the Future Internet, Software-Defined Networking (SDN), and Network Functions Virtualization (NFV), and P2P-based services and applications.