3

Software Defined Networking Concepts

Xenofon Foukas,[1,2] Mahesh K. Marina,[1] and Kimon Kontovasilis[2]
[1] *The University of Edinburgh, Edinburgh, UK*
[2] *NCSR "Demokritos", Athens, Greece*

3.1 Introduction

Software defined networking (SDN) is an idea that has recently reignited the interest of network researchers for programmable networks and shifted the attention of the networking community to this topic by promising to make the process of designing and managing networks more innovative and simplified compared to the well-established but inflexible current approach.

Designing and managing computer networks can become a very daunting task due to the high level of complexity involved. The tight coupling between a network's control plane (where the decisions of handling traffic are made) and data plane (where the actual forwarding of traffic takes place) gives rise to various challenges related to its management and evolution. Network operators need to manually transform high-level policies into low-level configuration commands, a process that for complex networks can be really challenging and error prone. Introducing new functionality to the network, like intrusion detection systems (IDS) and load balancers, usually requires tampering with the network's infrastructure and has a direct impact on its logic, while deploying new protocols can be a slow process demanding years of standardization and testing to ensure interoperability among the implementations provided by various vendors.

The idea of programmable networks has been proposed as a means to remedy this situation by promoting innovation in network management and the deployment of network services through programmability of the underlying network entities using some sort of an open network API. This leads to flexible networks able to operate according to the user's needs in a direct analogy to how programming languages are being used to reprogram computers in order to perform a number of tasks without the need for continuous modification of the underlying hardware platform.

Software Defined Mobile Networks (SDMN): Beyond LTE Network Architecture, First Edition.
Edited by Madhusanka Liyanage, Andrei Gurtov, and Mika Ylianttila.

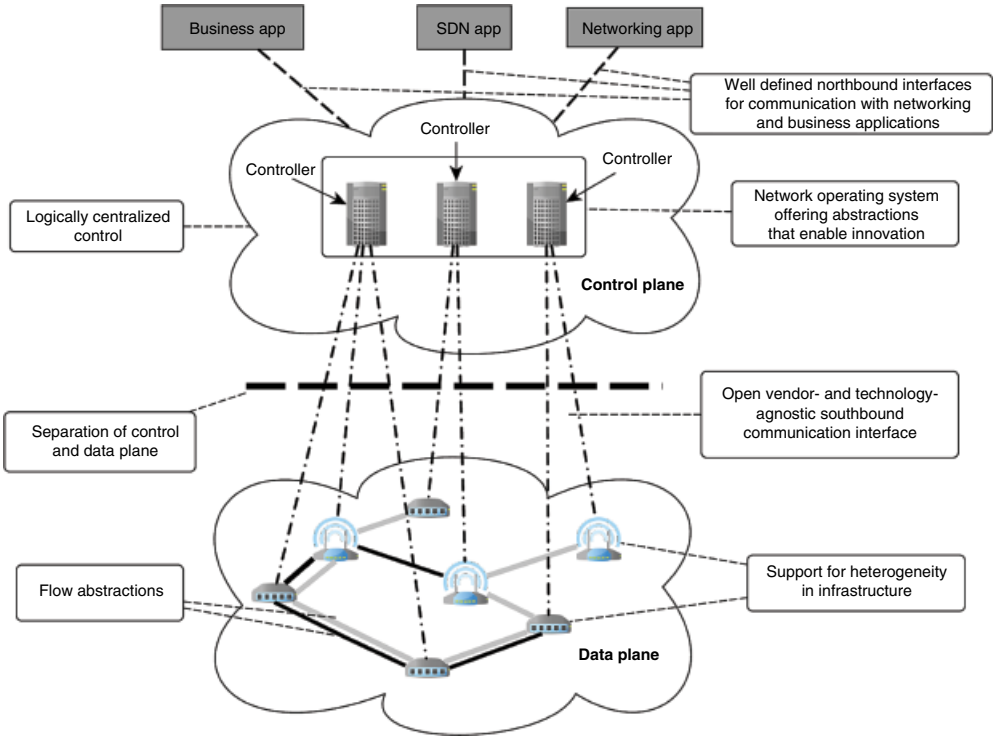

Figure 3.1 SDN in a nutshell—key ideas underlying the SDN paradigm.

SDN is a relatively new paradigm of a programmable network that changes the way that networks are designed and managed by introducing an abstraction that decouples the control from the data plane, as illustrated in Figure 3.1. In this approach, a software control program, referred to as the *controller*, has an overview of the whole network and is responsible for the decision making, while the hardware (routers, switches, etc.) is simply responsible for forwarding packets into their destination as per the controller's instructions, typically a set of packet handling rules.

The separation of the logically centralized control from the underlying data plane has quickly become the focus of vivid research interest in the networking community since it greatly simplifies network management and evolution in a number of ways. New protocols and applications can be tested and deployed over the network without affecting unrelated network traffic; additional infrastructure can be introduced without much hassle; and middleboxes can be easily integrated into the software control, allowing new potential solutions to be proposed for problems that have long been in the spotlight, like managing the highly complex core of cellular networks.

This chapter is a general overview of SDN for readers who have just been exposed to the SDN paradigm as well as for those requiring a survey of its past, present, and future. Through the discussion and the examples presented in this chapter, the reader should be able to comprehend why and how SDN shifts paradigms with respect to the design and management of networks and to understand the potential benefits that it has to offer to a number of interested parties like network operators and researchers.

The chapter begins by presenting a comprehensive history of programmable networks and their evolution to what we nowadays call SDN. Although the SDN hype is fairly recent, many of its underlying ideas are not new and have simply evolved over the past decades. Therefore, reviewing the history of programmable networks will provide to the reader a better understanding of the motivations and alternative solutions proposed over time, which helped to shape the modern SDN approach.

The next part of this chapter focuses on the building blocks of SDN, discussing the concept of the *controller* and giving an overview of the state of the art by presenting different design and implementation approaches. It also clarifies how the communication of the data and control plane could be achieved through a well-defined API by giving an overview of various emerging SDN programming languages. Moreover, it attempts to highlight the differences of SDN to other related but distinct technologies like network virtualization. Additionally, some representative examples of existing SDN applications are discussed, allowing the reader to appraise the benefits of exploiting SDN to create powerful applications.

The final part of the chapter discusses the impact of SDN to both the industry and the academic community by presenting the various working groups and research communities that have been formed over time describing their motivations and goals. This in turn demonstrates where the current research interest concentrates, which SDN-related ideas have been met with widespread acceptance, and what are the trends that will potentially drive future research in this field.

3.2 SDN History and Evolution

While the term *programmable* is used to generalize the concept of the simplified network management and reconfiguration, it is important to understand that in reality it encapsulates a wide number of ideas proposed over time, each having a different focus (e.g., control or data plane programmability) and different means of achieving their goals. This section reviews the history of programmable networks right from its early stages, when the need for network programmability first emerged, up to the present with the dominant paradigm of SDN. Along these lines, the key ideas that formed SDN will be discussed along with other alternatives that were proposed and affected SDN's evolution but that were not met with the same widespread success.

3.2.1 Early History of Programmable Networks

As already mentioned, the concept of programmable networks dates its origins back in the mid-1990s, right when the Internet was starting to experience widespread success. Until that moment, the usage of computer networks was limited to a small number of services like email and file transfers. The fast growth of the Internet outside of research facilities led to the formation of large networks, turning the interest of researchers and developers in deploying and experimenting with new ideas for network services. However, it quickly became apparent that a major obstacle toward this direction was the high complexity of managing the network infrastructure. Network devices were used as black boxes designed to support specific protocols essential for the operation of the network, without even guaranteeing vendor interoperability. Therefore, modifying the control logic of such devices was not an option, severely

restricting network evolution. To remedy this situation, various efforts focused on finding novel solutions for creating more open, extensible, and programmable networks.

Two of the most significant early ideas proposing ways of separating the control software from the underlying hardware and providing open interfaces for management and control were of the open signaling (OpenSig) [1] working group and from the active networking [2] initiative.

3.2.1.1 OpenSig

The OpenSig working group appeared in 1995 and focused on applying the concept of programmability in ATM networks. The main idea was the separation of the control and data plane of networks, with the signaling between the planes performed through an open interface. As a result, it would be possible to control and program ATM switches remotely, essentially turning the whole network into a distributed platform, greatly simplifying the process of deploying new services.

The ideas advocated by the OpenSig community for OpenSig interfaces acted as motivation for further research. Toward this direction, the Tempest framework [3], based on the OpenSig philosophy, allowed multiple switch controllers to manage multiple partitions of the switch simultaneously and consequently to run multiple control architectures over the same physical ATM network. This approach gave more freedom to network operators, as they were no longer forced to define a single unified control architecture satisfying the control requirements of all future network services.

Another project aimed at designing the necessary infrastructure for the control of ATM networks was Devolved Control of ATM Networks (DCAN) [4]. The main idea was that the control and management functions of the ATM network switches should be stripped from the devices and should be assigned to external dedicated workstations. DCAN presumed that the control and management operations of multiservice networks were inherently distributed due to the need of allocating resources across a network path in order to provide quality of service (QoS) guarantees. The communication between the management entity and the network was performed using a minimalistic protocol, much like what modern SDN protocols like OpenFlow do, adding any additional management functionality like the synchronization of streams in the management domain. The DCAN project was officially concluded in mid-1998.

3.2.1.2 Active Networking

The active networking initiative appeared in the mid-1990s and was mainly supported by DARPA [5, 6]. Like OpenSig, its main goal was the creation of programmable networks that would promote network innovations. The main idea behind active networking is that resources of network nodes are exposed through a network API, allowing network operators to actively control the nodes as they desire by executing arbitrary code. Therefore, contrary to the static functionality offered by OpenSig networks, active networking allowed the rapid deployment of customized services and the dynamic configuration of networks at run-time.

The general architecture of active networks defines a three-layer stack on active nodes. At the bottom layer sits an operating system (NodeOS) multiplexing the node's communication, memory, and computational resources among the packet flows traversing the node. Various projects proposing different implementations of the NodeOS exist, with some prominent

examples being the NodeOS project [7] and Bowman [8]. At the next layer exist one or more execution environments providing a model for writing active networking applications, including ANTS [9] and PLAN [10]. Finally, at the top layer are the active applications themselves, that is, the code developed by network operators.

Two programming models fall within the work of the active networking community [6, 11]: the *capsule model*, in which the code to be executed is included in regular data packets, and the *programmable router/switch model*, in which the code to be executed at network nodes is established through out-of-band mechanisms. Out of the two, the capsule model came to be the most innovative and most closely associated with active networking [6]. The reason is that it offered a radically different approach to network management, providing a simple method of installing new data plane functionality across network paths. However, both models had a significant impact and left an important legacy, since many of the concepts met in SDN (separation of the control and data plane, network APIs, etc.) come directly from the efforts of the active networking community.

3.2.2 Evolution of Programmable Networks to SDN

3.2.2.1 Shortcomings and Contributions of Previous Approaches

Although the key concepts expressed by these early approaches envisioned programmable networks that would allow innovation and would create open networking environments, none of the proposed technologies was met with widespread success. One of the main reasons for this failure was the lack of compelling problems that these approaches managed to solve [5, 6]. While the performance of various applications like content distribution and network management appeared to benefit from the idea of network programmability, there was no real pressing need that would turn the shift to the new paradigm into a necessity, leading to the commercialization of these early ideas.

Another reason for which active networking and OpenSig did not become mainstream was their focus on the wrong user group. Until then, the programmability of network devices could be performed only by programmers working for the vendors developing them. The new paradigm advocated as one of its advantages the flexibility it would give to end users to program the network, even though in reality the use case of end user programmers was really rare [6]. This clearly had a negative impact on the view that the research community and most importantly the industry had for programmable networks, as it overshadowed their strong points, understating their value for those that could really benefit like ISPs and network operators.

Furthermore, the focus of many early programmable network approaches was in promoting data instead of control plane programmability. For instance, active networking envisioned the exposure and manipulation of resources in network devices (packet queues, processing, storage, etc.) through an open API but did not provide any abstraction for logical control. In addition, while one of the basic ideas behind programmable networks was the decoupling of the control from the data plane, most proposed solutions made no clear distinction between the two [5]. These two facts hindered any attempts for innovation in the control plane, which arguably presents more opportunities than the data plane for discovering compelling use cases.

A final reason for the failure of early programmable networks was that they focused on proposing innovative architectures, programming models, and platforms, paying little or no attention to practical issues like the performance and the security they offered [6]. While such

features are not significant key concepts of network programmability, they are important factors when it comes to the point of commercializing this idea. Therefore, even though programmable networks had many theoretical advantages, the industry was not eager to adopt such solutions unless pressing performance and security issues were resolved.

Clearly, the aforementioned shortcomings of early programmable network attempts were the stumbling blocks to their widespread success. However, these attempts were really significant, since they defined for the first time key concepts that reformed the way that networks are perceived and identified new research areas of high potential. Even their shortcomings were of high significance, since they revealed many deficiencies that should be addressed if the new paradigm was to be successful one day. All in all, these early attempts were the cornerstones that shaped the way to the more promising and now widely accepted paradigm of SDN.

3.2.2.2 Shift to the SDN Paradigm

The first years of the 2000s saw major changes in the field of networking. New technologies like ADSL emerged, providing high-speed Internet access to consumers. At that moment, it was easier than ever before for an average consumer to afford an Internet connection that could be used for all sorts of activities, from email and teleconference services to large file exchanges and multimedia. This mass adoption of high-speed Internet and of all the new services that accompanied it had cataclysmic effects for networks, which saw their size and scope increase along with traffic volumes. Industrial stakeholders like ISPs and network operators started emphasizing on network reliability, performance, and QoS and required better approaches in performing important network configuration and management functions like routing, which at the time were primitive at best. Additionally, new trends in the storage and management of information like the appearance of cloud computing and the creation of large data centers made apparent the need for virtualized environments, accompanied by network virtualization as a means to support their automated provisioning, automation, and orchestration.

All these problems constituted compelling use cases that programmable networks promised to solve and shifted the attention of the networking community and the industry to this topic once more. This shift was strengthened by the improvement of servers that became substantially better than the control processors in routers, simplifying the task of moving the control functions outside network devices [6]. A result of this technological shift was the emergence of new improved network programmability attempts, with the most prominent example being SDN.

The main reason for the apparent success of SDN is that it managed to build on the strong points of early programmable network attempts while at the same time succeeded in addressing their shortcomings. Naturally, this shift from early programmable networks to SDN did not occur at once, but, as we shall now see, went through a series of intermediate steps.

As already mentioned, one of the major drawbacks of early programmable networking attempts was the lack of a clear distinction between the control and data plane of network devices. The Internet Engineering Task Force (IETF) Forwarding and Control Element Separation (ForCES) [12] working group tried to address this by redefining the internal architecture of network devices through the separation of the control from the data plane. In ForCES, two logical entities could be distinguished: the forwarding element (FE), which operated in the data plane and was responsible for per-packet processing and handling, and the

control element (CE), which was responsible for the logic of network devices, that is, for the implementation of management protocols, for control protocol processing, etc. A standardized interconnection protocol lay between the two elements enforcing the forwarding behavior to the FE as directed by the CE. The idea behind ForCES was that by allowing the forwarding and control planes to evolve separately and by providing a standard means of interconnection, it was possible to develop different types of FEs (general purpose or specialized) that could be combined with third-party control, allowing greater flexibility for innovation.

Another approach targeting the clean separation of the CE and FE of network devices was the 4D project [13]. Like ForCES, 4D emphasized the importance of separating the decision logic from the low-level network elements. However, in contrast to previous approaches, the 4D project envisioned an architecture based on four planes: a decision plane responsible for creating a network configuration, a dissemination plane responsible for delivering information related to the view of the network to the decision plane, a discovery plane allowing network devices to discover their immediate neighbors, and a data plane responsible for forwarding traffic. One experimental system based on the 4D architecture was Tesseract [14], which enabled the direct control of a network under the constraint of a single administrative domain. The ideas expressed in the 4D project acted as direct inspiration for many projects related to the controller component of SDNs, since it gave the notion of a logically centralized control of the network.

A final project worth mentioning during the pre-SDN era is SANE/Ethane [15, 16]. Ethane was a joint attempt made in 2007 by researchers in the universities of Stanford and Berkeley to create a new network architecture for the enterprise. Ethane adopted the main ideas expressed in 4D for a centralized control architecture, expanding it to incorporate security. The researchers behind Ethane argued that security could be integrated to network management, as both require some sort of policy, the ability to observe network traffic, and a means to control connectivity. Ethane achieved this by coupling very simple flow-based Ethernet switches with a centralized controller responsible for managing the admittance and routing of flows by communicating with the switches through a secure channel. A compelling feature of Ethane was that its flow-based switches could be incrementally deployed alongside conventional Ethernet switches and without any modification to end hosts required, allowing the widespread adoption of the architecture. Ethane was implemented in both software and hardware and was deployed at the campus of Stanford University for a period of a few months. The Ethane project was very significant, as the experiences gained by its design, implementation, and deployment laid the foundation for what would soon thereafter become SDN. In particular, Ethane is considered the immediate predecessor of OpenFlow, since the simple flow-based switches it introduced formed the basis of the original OpenFlow API.

3.2.2.3 The Emergence of SDN

In the second half of the 2000s, funding agencies and researchers started showing interest in the idea of network experimentation at scale [6]. This interest was mainly motivated by the need to deploy new protocols and services, targeting better performance and QoS in large enterprise networks and the Internet, and was further strengthened by the success of experimental infrastructures like PlanetLab [17] and by the emergence of various initiatives like the US National Science Foundation's Global Environment for Network Innovations (GENI).

Until then, large-scale experimentation was not an easy task to perform; researchers were mostly limited in using simulation environments for evaluation, which, despite their value, could not always capture all the important network-related parameters in the same manner as a realistic testbed would.

One important requirement of such infrastructure-based efforts was the need for network programmability, which would simplify network management and network service deployment and would allow multiple experiments to be run simultaneously at the same infrastructure, each using a different set of forwarding rules. Motivated by this idea, a group of researchers at Stanford created the Clean Slate Program. In the context of this project, which had as a mission to "reinvent the Internet," the OpenFlow protocol was proposed as a means for researchers to run experimental protocols in everyday networking environments. Similarly to previous approaches like ForCES, OpenFlow followed the principle of decoupling the control and forwarding plane and standardized the information exchanges between the two using a simple communication protocol. The solution proposed by OpenFlow, which provided architectural support for programming the network, led to the creation of the term SDN to encapsulate all the networks following similar architectural principles. The fundamental idea behind SDNs compared to the conventional networking paradigm is the creation of horizontally integrated systems through the separation of the control and the data plane while providing an increasingly sophisticated set of abstractions.

Looking back at all the milestones and important programmable network projects presented in this section, we can conclude that the road to SDN was indeed a long one with various ideas being proposed, tested, and evaluated, driving research in this field even further. SDN was not so much of a new idea, as it was the promising result of the distilled knowledge and experience obtained through many of the ideas presented in this section. What SDN managed to do differently compared to these ideas is that it integrated the most important network programmability concepts into an architecture that emerged at the right time and had compelling use cases for a great number of interested parties. Even though it remains to be seen whether SDN will be the next major paradigm shift in networking, the promise it demonstrates is undeniably very high.

3.3 SDN Paradigm and Applications

In this section, we focus on the key ideas underlying the SDN paradigm, the most recent instance in the evolution of programmable networks. In order to better understand the SDN concepts and to comprehend the benefits that this paradigm promises to deliver, we need to examine it both macro- and microscopically. For this, we begin this section by presenting a general overview of its architecture before going into an in-depth analysis of its building blocks.

3.3.1 Overview of SDN Building Blocks

As already mentioned, the SDN approach allows the management of network services through the abstraction of lower-level functionality. Instead of dealing with low-level details of network devices regarding the way that packets and flows are managed, network administrators now only need to use the abstractions available in the SDN architecture. The way that this is achieved is by decoupling the control plane from the data plane following the layered architecture illustrated in Figure 3.1.

At the bottom layer, we can observe the data plane, where the network infrastructure (switches, routers, wireless access points, etc.) lies. In the context of SDN, these devices have been stripped of all control logic (e.g., routing algorithms like BGP) simply implementing a set of forwarding operations for manipulating network data packets and flows, providing an abstract open interface for the communication with the upper layers. In the SDN terminology, these devices are commonly referred to as network *switches*.

Moving to the next layer, we can observe the control plane, where an entity referred to as the *controller* lies. This entity encapsulates the networking logic and is responsible for providing a programmatic interface to the network, which is used to implement new functionality and perform various management tasks. Unlike previous approaches like ForCES, the control plane of SDN is ripped entirely from the network device and is considered to be logically centralized, while physically it can be either centralized or decentralized residing in one or more servers, which control the network infrastructure as a whole.

An important aspect that distinguishes SDN from previous programmable network attempts is that it has introduced the notion of the network operating system abstraction [18]. Recall that previous efforts like active networking proposed some sort of node operating system (e.g., NodeOS) for controlling the underlying hardware. A network operating system offers a more general abstraction of network state in switches, revealing a simplified interface for controlling the network. This abstraction assumes a logically centralized control model, in which the applications view the network as a single system. In other words, the network operating system acts as an intermediate layer responsible for maintaining a consistent view of network state, which is then exploited by control logic to provide various networking services for topological discovery, routing, management of mobility, and statistics.

At the top of the SDN stack lies the application layer, which includes all the applications that exploit the services provided by the controller in order to perform network-related tasks, like load balancing, network virtualization, etc. One of the most important features of SDN is the openness it provides to third-party developers through the abstractions it defines for the easy development and deployment of new applications in various networked environments from data centers and WANs to wireless and cellular networks. Moreover, the SDN architecture eliminates the need for dedicated middleboxes like firewalls and IDS in the network topology, as it is now possible for their functionality to be implemented in the form of software applications that monitor and modify the network state through the network operating system services. Obviously, the existence of this layer adds great value to SDN, since it gives rise to a wide range of opportunities for innovation, making SDN a compelling solution both for researchers and the industry.

Finally, the communication of the controller to the data plane and the application layer can be achieved through well-defined interfaces (APIs). We can distinguish two main APIs in the SDN architecture: (i) a *southbound* API for the communication between the controller and the network infrastructure and (ii) a *northbound* API defining an interface between the network applications and the controller. This is similar to the way communication is achieved among the hardware, the operating system, and the user space in most computer systems.

Having seen the general overview of the SDN architecture, it is now time for an in-depth discussion of each of the building blocks just presented. Some examples of SDN applications will be discussed in the next section.

3.3.2 SDN Switches

In the conventional networking paradigm, the network infrastructure is considered the most integral part of the network. Each network device encapsulates all the functionality that would be required for the operation of the network. For instance, a router needs to provide the proper hardware like a ternary content-addressable memory (TCAM) for quickly forwarding packets, as well as sophisticated software for executing distributed routing protocols like BGP. Similarly, a wireless access point needs to have the proper hardware for wireless connectivity as well as software for forwarding packets, enforcing access control, etc. However, dynamically changing the behavior of network devices is not a trivial task due to their closed nature.

The three-layered SDN architecture presented in Section 3.3.1 changes this by decoupling the control from the forwarding operations, simplifying the management of network devices. As already mentioned, all forwarding devices retain the hardware that is responsible for storing the forwarding tables (e.g., application-specific integrated circuits (ASICs) with a TCAM) but are stripped of their logic. The controller dictates to the switches how packets should be forwarded by installing new forwarding rules through an abstract interface. Each time a packet arrives to a switch, its forwarding table is consulted and the packet is forwarded accordingly.

Even though in the earlier overview of SDN a clean three-layered architecture was presented, it remains unclear what the boundaries between the control and the data plane should be. For example, active queue management (AQM) and scheduling configuration are operations that are still considered part of the data plane even in the case of SDN switches. However, there is no inherent problem preventing these functions from becoming part of the control plane by introducing some sort of abstraction allowing the control of low-level behavior in switching devices. Such an approach could turn out to be beneficial, since it would simplify the deployment of new more efficient schemes for low-level switch operations [19].

On the other hand, while moving all control operations to a logically centralized controller has the advantage of easier network management, it can also raise scalability issues if physical implementation of the controller is also centralized. Therefore, it might be beneficial to retain some of the logic in the switches. For instance, in the case of DevoFlow [20], which is a modification of the OpenFlow model, the packet flows are distinguished into two categories: small (“mice”) flows handled directly by the switches and large (“elephant”) flows requiring the intervention of the controller. Similarly, in the DIFANE [21] controller, intermediate switches are used for storing the necessary rules, and the controller is relegated to the simple task of partitioning the rules over the switches.

Another issue of SDN switches is that the forwarding rules used in the case of SDN are more complex than those of conventional networks, using wildcards for forwarding packets, considering multiple fields of the packet like source and destination addresses, ports, application, etc. As a result, the switching hardware cannot easily cope with the management of packets and flows. In order for the forwarding operation to be fast, ASICs using TCAM are required. Unfortunately, such specialized hardware is expensive and power consuming, and as a result, only a limited number of forwarding entries for flow-based forwarding schemes can be supported in each switch, hindering network scalability. A way to cope with this would be to introduce an assisting CPU to the switch or somewhere nearby to perform not only control plane but also data plane functionalities, for example, let the CPU forward the “mice” flows [22], or to introduce new architectures that would be more expressive and would allow more actions related to packet processing to be performed [23].

The issue of hardware limitations is not only restricted to fixed networks but is extended to the wireless and mobile domains as well. The wireless data plane needs to be redesigned in order to offer more useful abstractions similarly to what happened with the data plane of fixed networks. While the data plane abstractions offered by protocols like OpenFlow support the idea of decoupling the control from the data plane, they cannot be extended to the wireless and mobile field unless the underlying hardware (e.g., switches in backhaul cellular networks and wireless access points) starts providing equally sophisticated and useful abstractions [5].

Regardless of the way that SDN switches are implemented, it should be made clear that in order for the new paradigm to gain popularity, backward compatibility is a very important factor. While pure SDN switches that completely lack integrated control exist, it is the hybrid approach (i.e., support of SDN along with traditional operation and protocols) that would probably be the most successful at these early steps of SDN [11]. The reason is that while the features of SDN present a compelling solution for many realistic scenarios, the infrastructure in most enterprise networks still follows the conventional approach. Therefore, an intermediate hybrid network form would probably ease the transition to SDN.

3.3.3 SDN Controllers

As already mentioned, one of the core ideas of the SDN philosophy is the existence of a network operating system placed between the network infrastructure and the application layer. This network operating system is responsible for coordinating and managing the resources of the whole network and for revealing an abstract unified view of all components to the applications executed on top of it. This idea is analogous to the one followed in a typical computer system, where the operating system lies between the hardware and the user space and is responsible for managing the hardware resources and providing common services for user programs. Similarly, network administrators and developers are now presented with a homogeneous environment easier to program and configure much like a typical computer program developer would.

The logically centralized control and the generalized network abstraction it offers make the SDN model applicable to a wider range of applications and heterogeneous network technologies compared to the conventional networking paradigm. For instance, consider a heterogeneous environment composed of a fixed and a wireless network comprised of a large number of related network devices (routers, switches, wireless access points, middleboxes, etc.). In the traditional networking paradigm, each network device would require individual low-level configuration by the network administrator in order to operate properly. Moreover, since each device targets a different networking technology, it would have its own specific management and configuration requirements, meaning that extra effort would be required by the administrator to make the whole network operate as intended. On the other hand, with the logically centralized control of SDN, the administrator would not have to worry about low-level details. Instead, the network management would be performed by defining a proper high-level policy, leaving the network operating system responsible for communicating with and configuring the operation of network devices.

Having discussed the general concepts behind the SDN controller, the following subsections take a closer look at specific design decisions and implementation choices made at this core component that can prove to be critical for the overall performance and scalability of the network.

3.3.3.1 Centralization of Control in SDN

As already discussed, the SDN architecture specifies that the network infrastructure is logically controlled by a central entity responsible for management and policy enforcement. However, it should be made clear that logically centralized control does not necessarily also imply physical centralization.

There have been various proposals for physically centralized controllers, like NOX [18] and Maestro [24]. A physically centralized control design simplifies the controller implementation. All switches are controlled by the same physical entity, meaning that the network is not subject to consistency-related issues, with all the applications seeing the same network state (which comes from the same controller). Despite its advantages, this approach suffers from the same weakness that all centralized systems do, that is, the controller acts as a single point of failure for the whole network. A way to overcome this is by connecting multiple controllers to a switch, allowing a backup controller to take over in the event of a failure. In this case, all controllers need to have a consistent view of the network; otherwise, applications might fail to operate properly. Moreover, the centralized approach can raise scalability concerns, since all network devices need to be managed by the same entity.

One approach that further generalizes the idea of using multiple controllers over the network is to maintain a logically centralized but physically decentralized control plane. In this case, each controller is responsible for managing only one part of the network, but all controllers communicate and maintain a common network view. Therefore, applications view the controller as a single entity, while in reality control operations are performed by a distributed system. The advantage of this approach, apart from not having a single point of failure anymore, is the increase in performance and scalability, since only a part of the network needs to be managed by each individual controller component. Some well-known controllers that belong to this category are Onix [25] and HyperFlow [26]. One potential downside of decentralized control is once more related to the consistency of the network state among controller components. Since the state of the network is distributed, it is possible that applications served by different controllers might have a different view of the network, which might make them operate improperly.

A hybrid solution that tries to encompass both scalability and consistency is to use two layers of controllers like the Kandoo [27] controller does. The bottom layer is composed of a group of controllers that do not have knowledge of the whole network state. These controllers only run control operations that require knowing the state of a single switch (local information only). On the other hand, the top layer is a logically centralized controller responsible for performing network-wide operations that require knowledge of the whole network state. The idea is that local operations can be performed faster this way and do not incur any additional load to the high-level central controller, effectively increasing the scalability of the network.

Apart from the ideas related to the level of physical centralization of controllers, there have been other proposed solutions related to their logical decentralization. The idea of logical decentralization comes directly from the early era of programmable networks and from the Tempest project. Recall that the Tempest architecture allowed multiple virtual ATM networks to operate on top of the same set of physical switches. Similarly, there have been proposals for SDN proxy controllers like FlowVisor [28] that allow multiple controllers to share the same forwarding plane. The motivation for this idea was to enable the simultaneous deployment of experimental and enterprise networks over the same infrastructure without affecting one another.

Before concluding our discussion on the degree of centralization with SDN controllers, it is important to examine the concerns that can be raised regarding their performance and applicability over large networking environments.

One of the most frequent concerns raised by SDN skeptics is the ability of SDN networks to scale and be responsive in cases of high network load. This concern comes mainly from the fact that in the new paradigm control moves out of network devices and goes in a single entity responsible for managing the whole network traffic. Motivated by this concern, performance studies of SDN controller implementations [29] have revealed that even physically centralized controllers can perform really well, having very low response times. For instance, it has been shown that even primitive single-threaded controllers like NOX can handle an average workload of up to 200 thousand new flows per second with a maximum latency of 600 ms for networks composed of up to 256 switches. Newer multithreaded controller implementations have been shown to perform significantly better. For instance, NOX-MT [30] can handle 1.6 million new flows per second in a 256-switch network with an average response time of 2 ms in a commodity eight-core machine of 2 GHz CPUs. Newer controller designs targeting large industrial servers promise to improve the performance even further. For instance, the McNettle [31] controller claims to be able to serve networks of up to 5000 switches using a single controller of 46 cores with a throughput of over 14 million flows per second and latency under 10 ms.

Another important performance concern raised in the case of a physically decentralized control plane is the way that controllers are placed within the network, as the network performance can be greatly affected by the number and the physical location of controllers, as well as by the algorithms used for their coordination. In order to address this, various solutions have been proposed, from viewing the placement of controllers as an optimization problem [32] to establishing connections of this problem to the fields of local algorithms and distributed computing for developing efficient controller coordination protocols [33].

A final concern raised in the case of physically distributed SDN controllers is related to the consistency of the network state maintained at each controller when performing policy updates due to concurrency issues that might occur by the error-prone, distributed nature of the logical controller. The solutions of such a problem can be similar to those of transactional databases, with the controller being extended with a transactional interface defining semantics for either completely committing a policy update or aborting [34].

3.3.3.2 Management of Traffic

Another very important design issue of SDN controllers is related to the way that traffic is managed. The decisions about traffic management can have a direct impact on the performance of the network, especially in cases of large networks composed of many switches and with high traffic loads. We can divide the problems related to traffic management into two categories: control granularity and policy enforcement.

Control Granularity

The control granularity applied over network traffic refers to how fine or coarse grained the controller inspection operations should be in relation to the packets traversing the network [11]. In conventional networks, each packet arriving at a switch is examined individually, and a routing decision is made as to where the packet should be forwarded depending on the

information it carries (e.g., destination address). While this approach generally works for conventional networks, the same cannot be said for SDN. In this case, the per-packet approach becomes infeasible to implement across any sizeable network, since all packets would have to pass through the controller that would need to construct a route for each one of them individually.

Due to the performance issues raised by the per-packet approach, most SDN controllers follow a flow-based approach, where each packet is assigned to some flow according to a specific property (e.g., the packet's source and destination address and the application it is related with). The controller sets up a new flow by examining the first packet arriving for that flow and configuring the switches accordingly. In order to further off-load the controller, an extra coarse-grained approach would be to enforce control based on an aggregation flow match instead of using individual flows.

The main trade-off when examining the level of granularity is the load in the controller versus the QoS offered to network applications. The more fine grained the control, the higher the QoS. In the per-packet approach, the controller can always make the best decisions for routing each individual packet, therefore leading to improved QoS. On the other end, enforcing control over an aggregation of flows means that the controller decisions for forwarding packets do not fully adapt to the state of the network. In this case, packets might be forwarded through a suboptimal route, leading to degraded QoS.

Policy Enforcement

The second issue in the management of traffic is related to the way that network policies are applied by the controller over network devices [11]. One approach, followed by systems like Ethane, is to have a *reactive* control model, where the switching device consults the controller every time a decision for a new flow needs to be made. In this case, the policy for each flow is established to the switches only when an actual demand arises, making network management more flexible. A potential downside of this approach is the degradation of performance, due to the time required for the first packet of the flow to go to the controller for inspection. This performance drop could be significant, especially in cases of controllers that are physically located far away from the switch.

An alternative policy enforcement approach would be to use a *proactive* control model. In this case, the controller populates the flow tables ahead of time for any traffic that could go through the switches and then pushes the rules to all the switches of the network. Using this approach, a switch no longer has to request directions by the controller to set up a new flow and instead can perform a simple lookup at the table already stored in the TCAM of the device. The advantage of proactive control is that it eliminates the latency induced by consulting the controller for every flow.

3.3.4 SDN Programming Interfaces

As already mentioned, the communication of the controller with the other layers is achieved through a southbound API for the controller–switch interactions and through a northbound API for the controller–application interactions. In this section, we briefly discuss the main concepts and issues related to SDN programming by separately examining each point of communication.

3.3.4.1 Southbound Communication

The southbound communication is very important for the manipulation of the behavior of SDN switches by the controller. It is the way that SDN attempts to "program" the network. The most prominent example of a standardized southbound API is OpenFlow [35]. Most projects related to SDN assume that the communication of the controller with the switches is OpenFlow based, and therefore, it is important to make a detailed presentation of the OpenFlow approach. However, it should be made clear that OpenFlow is just one (rather popular) out of many possible implementations of controller–switch interactions. Other alternatives, for example, DevoFlow [20], also exist, attempting to solve performance issues that OpenFlow faces.

Overview of OpenFlow

Following the SDN principle of decoupling the control and data planes, OpenFlow provides a standardized way of managing traffic in switches and of exchanging information between the switches and the controller, as Figure 3.2 illustrates. The OpenFlow switch is composed of two logical components. The first component contains one or more flow tables responsible for maintaining the information required by the switch in order to forward packets. The second component is an OpenFlow client, which is essentially a simple API allowing the communication of the switch with the controller.

The flow tables consist of flow entries, each of which defines a set of rules determining how the packets belonging to that particular flow will be managed by the switch (i.e., how they will be processed and forwarded). Each entry in the flow table has three fields: (i) a packet header defining the flow, (ii) an action determining how the packet should be processed, and (iii) statistics, which keep track of information like the number of packets and bytes of each flow and the time since a packet of the flow was last forwarded.

Once a packet arrives at the OpenFlow switch, its header is examined, and the packet is matched to the flow that has the most similar packet header field. If a matching flow is found, the action defined in the action field is performed. These actions include the forwarding of the

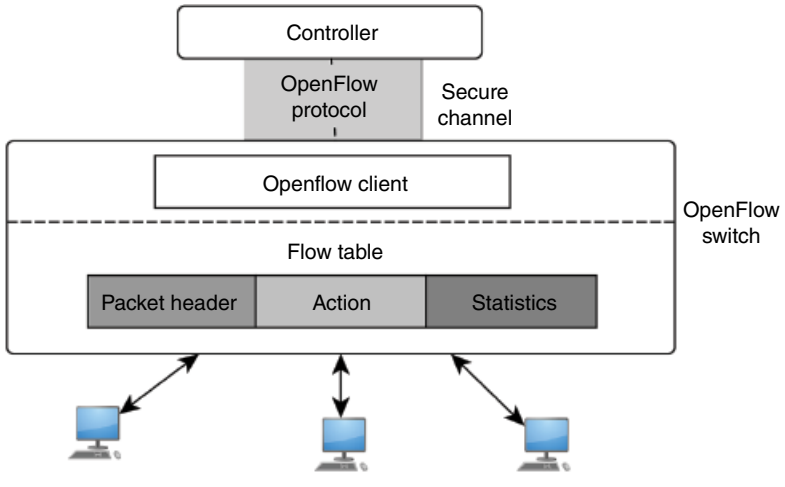

Figure 3.2 Design of an OpenFlow switch and communication with the controller.

packet to a particular port in order to be routed through the network, the forwarding of the packet in order to be examined by the controller, or the rejection of the packet. If the packet cannot be matched to any flow, it is treated according to the action defined in a table-miss flow entry.

The exchange of information between the switch and the controller happens by sending messages through a secure channel in a standardized way defined by the OpenFlow protocol. This way, the controller can manipulate the flows found in the flow table of the switch (i.e., add, update, or delete a flow entry) either proactively or reactively as discussed in the basic controller principles. Since the controller is able to communicate with the switch using the OpenFlow protocol, there is no longer a need for network operators to interact directly with the switch.

A particularly compelling feature of OpenFlow is that the packet header field can be a wildcard, meaning that the matching to the header of the packet does not have to be exact. The idea behind this approach is that various network devices like routers, switches, and middleboxes have a similar forwarding behavior, differing only in terms of which header fields they use for matching and the actions they perform. OpenFlow allows the use of any subset of these header fields for applying rules on traffic flows, meaning that it conceptually unifies many different types of network devices. For instance, a router could be emulated by a flow entry using a packet header performing a match only on the IP address, while a firewall would be emulated through a packet header field containing additional information like the source and destination IP addresses and port numbers as well as the transport protocol employed.

3.3.4.2 Northbound API

As already discussed, one of the basic ideas advocated in the SDN paradigm is the existence of a network operating system, lying between the network infrastructure and the high-level services and applications, similarly to how a computer operating system lies between the hardware and the user space. Assuming such a centralized coordination entity and based on the basic operating system principles, a clearly defined interface should also exist in the SDN architecture for the interaction of the controller with applications. This interface should allow the applications to access the underlying hardware, manage the system resources, and allow their interaction with other applications without having any knowledge of low-level network information.

In contrast to the southbound communication, where the interactions between the switches and the controller are well defined through a standardized open interface (i.e., OpenFlow), there is currently no accepted standard for the interaction of the controller with applications [11]. Therefore, each controller model needs to provide its own methods for performing controller–application communication. Moreover, even the interfaces current controllers implement provide very low-level abstractions (i.e., flow manipulation), which make it difficult to implement applications with different and many times conflicting objectives that are based in more high-level concepts. As an example, consider a power management and a firewall application. The power management application needs to reroute traffic using as few links as possible in order to deactivate idle switches, while the firewall might need these extra switches to route traffic as they best fit the firewall rules. Leaving the programmer to deal with these conflicts could become a very complex and cumbersome process.

To solve this problem, many ideas have been proposed, advocating the use of high-level network programming languages responsible for translating policies to low-level flow constraints,

which in turn will be used by the controller to manage the SDN switches. These network programming languages can also be seen as an intermediate layer in the SDN architecture, placed between the application layer and the controller in a similar manner as to how high-level programming languages like C++ and Python exist on top of the assembly language for hiding the complex low-level details of the assembly language from the programmer. Some examples of such high-level network programming languages include Frenetic [36] and Pyretic [37].

3.3.5 SDN Application Domains

In order to demonstrate the applicability of SDN in a wide range of networking domains, we briefly present two characteristic examples in which SDN could prove to be beneficial: data centers and cellular networks. Of course, the list of SDN applications is not only limited to these domains but is also extended in many others, from enterprise networks, WLANs, and heterogeneous networks to optical networks and the Internet of Things [5, 11].

3.3.5.1 Data Center Networks

One of the most important requirements for data center networks is to find ways to scale in order to support hundreds of thousands of servers and millions of virtual machines. However, achieving such scalability can be a challenging task from a network perspective. First of all, the size of forwarding tables increases along with the number of servers, leading to a requirement for more sophisticated and expensive forwarding devices. Moreover, traffic management and policy enforcement can become very important and critical issues, since data centers are expected to continuously achieve high levels of performance.

In traditional data centers, the aforementioned requirements are typically met through the careful design and configuration of the underlying network. This operation is in most cases performed manually by defining the preferred routes for traffic and by placing middleboxes at strategic choke points on the physical network. Obviously, this approach contradicts the requirement for scalability, since manual configuration can become a very challenging and error-prone task, especially as the size of the network grows. Additionally, it becomes increasingly difficult to make the data center operate at its full capacity, since it cannot dynamically adapt to the application requirements.

The advantages that SDN offers to network management come to fill these gaps. By decoupling the control from the data plane, forwarding devices become much simpler and therefore cheaper. At the same time, all control logic is delegated to one logically centralized entity. This allows the dynamic management of flows, the load balancing of traffic, and the allocation of resources in a manner that best adjusts the operation of the data center to the needs of running applications, which in turn leads to increased performance [38]. Finally, placing middleboxes in the network is no longer required, since policy enforcement can now be achieved through the controller entity.

3.3.5.2 Cellular Networks

The market of cellular mobile networks is perhaps one of the most profitable in telecommunications. The rapid increase in the number of cellular devices (e.g., smartphones and tablets) during the past decade has pushed the existing cellular networks to their limits. Recently, there

has been significant interest in integrating the SDN principles in current cellular architectures like the 3G Universal Mobile Telecommunications System (UMTS) and the 4G Long-Term Evolution (LTE) [39].

One of the main disadvantages of current cellular network architectures is that the core of the network has a centralized data flow, with all traffic passing through specialized equipment, which packs multiple network functions from routing to access control and billing (e.g., packet gateway in LTE), leading to an increase of the infrastructural cost due to the complexity of the devices and raising serious scalability concerns. Moreover, cell sizes of the access network tend to get smaller in order to cover the demands of the ever-increasing traffic and the limited wireless spectrum for accessing the network. However, this leads to increased interference among neighboring base stations and to the fluctuation of load from one base station to another due to user mobility, rendering the static allocation of resources no longer adequate.

Applying the SDN principles to cellular networks promises to solve some of these deficiencies. First of all, decoupling the control from the data plane and introducing a centralized controller that has a complete view of the whole network allow network equipment to become simpler and therefore reduce the overall infrastructural cost. Moreover, operations like routing, real-time monitoring, mobility management, access control, and policy enforcement can be assigned to different cooperating controllers, making the network more flexible and easier to manage. Furthermore, using a centralized controller acting as an abstract base station simplifies the operations of load and interference management, no longer requiring the direct communication and coordination of base stations. Instead, the controller makes the decisions for the whole network and simply instructs the data plane (i.e., the base stations) on how to operate. One final advantage is that the use of SDN eases the introduction of virtual operators to the telecommunication market, leading to increased competitiveness. By virtualizing the underlying switching equipment, all providers become responsible for managing the flows of their own subscribers through their own controllers, without the requirement to pay large sums for obtaining their own infrastructure.

3.3.6 Relation of SDN to Network Virtualization and Network Function Virtualization

Two very popular technologies closely related to SDN are network virtualization and network function virtualization (NFV). In this subsection, we briefly attempt to clarify their relationship to SDN, since these technologies tend to become the cause of confusion especially for those recently introduced to the concept of SDN.

Network virtualization is the separation of the network topology from the underlying physical infrastructure. Through virtualization, it is possible to have multiple "virtual" networks deployed over the same physical equipment, with each of them having a much simpler topology compared to that of the physical network. This abstraction allows network operators to construct networks as they see fit without having to tamper with the underlying infrastructure, which can turn out to be a difficult or even impossible process. For instance, through network virtualization, it becomes possible to have a virtual local area network (VLAN) of hosts spanning multiple physical networks or to have multiple VLANs on top of a single physical subnet.

The idea behind network virtualization of decoupling the network from the underlying physical infrastructure bears resemblance to that advocated by SDN for decoupling the control

from the data plane and therefore naturally becomes a source of confusion. The truth is that none of the two technologies is dependent on the other. The existence of SDN does not readily imply network virtualization. Similarly, SDN is not necessarily a prerequisite for achieving network virtualization. On the contrary, it is possible to deploy a network virtualization solution over an SDN network, while at the same time an SDN network could be deployed in a virtualized environment.

Since its appearance, SDN has closely coexisted with network virtualization, which acted as one of the first and perhaps the most important use cases of SDN. The reason is that the architectural flexibility offered by SDN acted as an enabler for network virtualization. In other words, network virtualization can be seen as a solution focusing on a particular problem, while SDN is one (perhaps the best at this moment) architecture for achieving this. However, as already stressed, network virtualization needs to be seen independently from SDN. In fact, it has been argued by many that network virtualization could turn out to be even bigger technological innovation than SDN [6].

Another technology that is closely related but different from SDN is NFV [40]. NFV is a carrier-driven initiative with a goal to transform the way that operators architect networks by employing virtualization-related technologies to virtualize network functions such as intrusion detection, caching, domain name service (DNS), and network address translation (NAT) so that they can run in software. Through the introduction of virtualization, it is possible to run these functions over generic industry-standard high-volume servers, switches, and storage devices instead of using proprietary purpose-built network devices. This approach reduces operational and deployment costs, since operators no longer need to rely on expensive proprietary hardware solutions. Finally, flexibility in network management increases as it is possible to quickly modify or introduce new services to address changing demands.

The decoupling of network functions from the underlying hardware is closely related to the decoupling of the control from the data plane advocated by SDN, and therefore, the distinction of the two technologies can be a bit vague. It is important to understand that even though closely related, SDN and NFV refer to different domains. NFV is complementary to SDN but does not depend on it, and vice versa. For instance, the control functions of SDN could be implemented as virtual functions based on the NFV technology. On the other hand, an NFV orchestration system could control the forwarding behavior of physical switches through SDN. However, neither technology is a requirement for the operation of other, but both could benefit from the advantages each can offer.

3.4 Impact of SDN to Research and Industry

Having seen the basic concepts of SDN and some important applications of this approach, it is now time to briefly discuss the impact of SDN to the research community and the industry. While the focus of each interested party might be different, from designing novel solutions exploiting the benefits of SDN to developing SDN-enabled products ready to be deployed in commercial environments, their involvement in the evolution of SDN helps in shaping the future of this technology. Seeing what the motivation and the focus of current SDN-related attempts will provide us with indications of what will potentially drive future research in this field.

3.4.1 Overview of Standardization Activities and SDN Summits

Recently, several standardization organizations have started focusing on SDN, each working in providing standardized solutions for a different part of the SDN space. The benefits of such efforts are very significant, since standardization is the first step toward the wide adoption of a technology.

The most relevant standardization organization for SDN is considered the Open Networking Foundation (ONF) [41], which is a nonprofit industry consortium founded in 2011. It has more than 100 company members including telecom operators, network and service providers, equipment vendors, and networking and virtualization software suppliers. Its vision is to make SDN the new norm for networks by transforming the networking industry to a software industry through the open SDN standards. To achieve this, it attempts to standardize and commercialize SDN and its underlying technologies, with its main accomplishment the standardization of the OpenFlow protocol, which is also the first SDN standard. ONF has a number of working groups working in different aspects of SDN from forwarding abstractions, extensibility, configuration, and management to educating the community on the SDN value proposition.

The IETF, which is a major driving force in developing and promoting Internet standards, also has a number of working groups focusing on SDN in a broader scope than just OpenFlow. The Software Defined Networking Research Group (SDNRG) [42] focuses on identifying solutions related to the scalability and applicability of the SDN model as well as for developing abstractions and programming languages useful in the context of SDN. Finally, it attempts to identify SDN use cases and future research challenges. On a different approach, the Interface to the Routing System (I2RS) [43] working group is developing an SDN strategy, counter to the OpenFlow approach, in which traditional distributed routing protocols can run on network hardware to provide information to a centrally located manager. Other SDN-related IETF working groups include application-layer traffic optimization (ALTO) [44] using SDN and CDNI [45] studying how SDN can be used for content delivery network (CDN) interconnection.

Some study groups (SGs) of ITU's Telecommunication Standardization Sector (ITU-T) [46] are also looking on SDN for public telecommunication networks. For instance, Study Group 13 (SG13) is focusing on a framework of telecom SDN and on defining requirements of formal specification and verification methods for SDN. Study Group 11 (SG11) is developing requirements and architectures on SDN signaling, while Study Group 15 (SG15) has started discussions on transport SDN.

Other standardization organizations that have also been interested in applying the SDN principles include the Optical Internetworking Forum (OIF) [47], the Broadband Forum (BBF) [48], and the Metro Ethernet Forum (MEF) [49]. OIF is responsible for promoting the development and deployment of interoperable optical networking systems, and it supports a working group to define the requirements for a transport network SDN architecture. BBF is a forum for fixed line broadband access and core networks, working on a cloud-based gateway that could be implemented using SDN concepts. Finally, MEF has as its goal to develop, promote, and certify technical specifications for carrier Ethernet services. One of its directions is to investigate whether MEF services could fit within an ONF SDN framework.

Apart from the work performed on standardizing SDN solutions, there exist a number of summits for sharing and exploring new ideas and key developments produced in the SDN research community. The Open Networking Summit (ONS) is perhaps the most important

SDN event having as a mission "to help the SDN revolution succeed by producing high-impact SDN events." Other SDN-related venues have also started emerging like the SDN & NFV Summit for solutions on network virtualization, the SDN & OpenFlow World Congress, the SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN), and the IEEE/IFIP International Workshop on SDN Management and Orchestration (SDNMO).

3.4.2 SDN in the Industry

The advantages that SDN offers compared to traditional networking have also made the industry focus on SDN either for using it as a means to simplify management and improve services in their own private networks or for developing and providing commercial SDN solutions.

Perhaps one of the most characteristic examples for the adoption of SDN in production networks is Google, which entered in the world of SDN with its B4 network [50] developed for connecting its data centers worldwide. The main reason for moving to the SDN paradigm, as explained by Google engineers, was the very fast growth of Google's back-end network. While computational power and storage become cheaper as scale increases, the same cannot be said for the network. By applying SDN principles, the company was able to choose the networking hardware according to the features it required, while it managed to develop innovative software solutions. Moreover, the centralized network control made the network more efficient and fault tolerant providing a more flexible and innovative environment, while at the same time it led to a reduction of operational expenses. More recently, Google revealed Andromeda [51], a software defined network underlying its cloud, which is aimed at enabling Google's services to scale better, cheaper, and faster. Other major companies in the field of networking and cloud services like Facebook and Amazon are also planning on building their next-generation network infrastructure based on the SDN principles.

Networking companies have also started showing interest in developing commercial SDN solutions. This interest is not limited in developing specific products like OpenFlow switches and network operating systems; rather, there is a trend for creating complete SDN ecosystems targeting different types of customers. For instance, companies like Cisco, HP, and Alcatel have entered the SDN market, presenting their own complete solutions intended for enterprises and cloud service providers, while telecommunication companies like Huawei are designing solutions for the next generation of telecom networks, with a specific interest in LTE and LTE-Advanced networks. In 2012, VMware acquired an SDN start-up called Nicira in order to integrate its network virtualization platform (NVP) to NSX, VMware's own network virtualization and security platform for software defined data centers. The list of major companies providing SDN solutions constantly grows, with many others like Broadcom, Oracle, NTT, Juniper, and Big Switch Networks recognizing the benefits of SDN and proposing their own solutions.

3.4.3 Future of SDN

Going back to the beginning of this discussion and looking at all the intermediate steps that led to modern software defined networks, it is tricky to predict what lies in the future. Previous attempts for redesigning the network architecture have shown that very promising technologies

can fail due to lack of the proper conditions, while success depends on a number of factors from finding compelling use cases for the emerging technology to managing its adoption not only by the research community but by the industry as well. The way that SDN deals with these matters makes it a very promising candidate for being the next major disruption in the networking field. The benefits of applying the SDN principles in different types of networks, the unification of heterogeneous environments, and the wide number of applications that this paradigm offers demonstrate its very high potential to become a major driving force commercially in the very near future especially for cloud service providers, network operators, and mobile carriers. It remains to be seen whether these predictions will be confirmed and to what extent SDN will deliver its promises.

References

[1] A. T. Campbell, I. Katzela, K. Miki, and J. Vicente. "Open signaling for ATM, internet and mobile networks (OPENSIG'98)." ACM SIGCOMM Computer Communication Review 29.1 (1999): 97–108.

[2] D. L. Tennenhouse, J. Smith, D. Sincoskie, D. J. Wetherall, and G. J. Minden. "A survey of active network research." IEEE Communications Magazine 35.1 (1997): 80–86.

[3] J. E. Van der Merwe, S. Rooney, I. Leslie, and S. Crosby. "The tempest-a practical framework for network programmability." IEEE Network 12.3 (1998): 20–28.

[4] "Devolved control of ATM networks." Available from http://www.cl.cam.ac.uk/research/srg/netos/old-projects/dcan/ (accessed January 19, 2015).

[5] J. Qadir, N. Ahmed, and N. Ahad. "Building programmable wireless networks: An architectural survey." arXiv preprint arXiv:1310.0251 (2013).

[6] N. Feamster, J. Rexford, and E. Zegura. "The road to SDN." ACM Queue 11.12 (2013): 20–40.

[7] N. Shalaby, Y. Gottlieb, M. Wawrzoniak, and L. Peterson. "Snow on silk: A nodeOS in the Linux kernel." *Active Networks*. Springer, Berlin (2002): 1–19.

[8] S. Merugu, S. Bhattacharjee, E. Zegura, and K. Calvert. "Bowman: A node OS for active networks." INFOCOM 2000. Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies; Tel Aviv 3 (2000): 1127–1136.

[9] D. J. Wetherall, J. V. Guttag, and D. L. Tennenhouse. "ANTS: A toolkit for building and dynamically deploying network protocols." Open Architectures and Network Programming, 1998 IEEE: pp. 117, 129; April 3–4, 1998. doi: 10.1109/OPNARC.1998.662048.

[10] M. Hicks, P. Kakkar, J. T. Moore, C. A. Gunter, and S. Nettles. "PLAN: A packet language for active networks." ACM SIGPLAN Notices 34.1 (1998): 86–93.

[11] B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti. "A survey of software-defined networking: Past, present, and future of programmable networks." Communications Surveys & Tutorials, IEEE, 16 (3) (2014): 1617–1634, third quarter.

[12] L. Yang, R. Dantu, T. Anderson, and R. Gopal. "Forwarding and control element separation (ForCES) framework." RFC 3746, (2004). Available at https://tools.ietf.org/html/rfc3746 (accessed February 17, 2015).

[13] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. "A clean slate 4D approach to network control and management." ACM SIGCOMM Computer Communication Review 35.5 (2005): 41–54.

[14] H. Yan, D. A. Maltz, T. S. Eugene Ng, H. Gogineni, H. Zhang, and Z. Cai. "Tesseract: A 4D network control plane." 4th USENIX Symposium on Networked Systems Design & Implementation 7; Cambridge, MA (2007): 369–382.

[15] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker. "SANE: A protection architecture for enterprise networks." 15th USENIX Security Symposium; Vancouver, BC, Canada (2006): 137–151.

[16] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. "Ethane: Taking control of the enterprise." ACM SIGCOMM Computer Communication Review 37.4 (2007): 1–12.

[17] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. "Planetlab: An overlay testbed for broad-coverage services." ACM SIGCOMM Computer Communication Review 33.3 (2003): 3–12.

[18] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. "NOX: Towards an operating system for networks." ACM SIGCOMM Computer Communication Review 38.3 (2008): 105–110.

[19] A. Sivaraman, K. Winstein, S. Subramanian, and H. Balakrishnan. "No silver bullet: Extending SDN to the data plane." Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks 19; Maryland (2013): 1–7.

[20] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. "Devoflow: Scaling flow management for high-performance networks." ACM SIGCOMM Computer Communication Review 41.4 (2011): 254–265.

[21] M. Yu, J. Rexford, M. J. Freedman, and J. Wang. "Scalable flow-based networking with DIFANE." ACM SIGCOMM Computer Communication Review 40.4 (2010): 351–362.

[22] G. Lu, R. Miao, Y. Xiong, and C. Guo. "Using cpu as a traffic co-processing unit in commodity switches." Proceedings of the First Workshop on Hot Topics in Software Defined Networks; Helsinki, Finland (2012): 31–36.

[23] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN." SIGCOMM Computer Communication Review 43.4 (2013): 99–110.

[24] Z. Cai, A. L. Cox, and T. E. Ng. "Maestro: A system for scalable OpenFlow control." Technical Report TR10-08. Texas: Rice University (2010).

[25] 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." 9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 10; Vancouver, BC, Canada (2010): 1–6.

[26] A. Tootoonchian and Y. Ganjali. "Hyperflow: A distributed control plane for openflow." Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking. San Jose, CA: USENIX Association (2010): 3–8.

[27] S. H. Yeganeh and Y. Ganjali. "Kandoo: A framework for efficient and scalable offloading of control applications." Proceedings of the First Workshop on Hot Topics in Software Defined Networks. Helsinki, Finland: ACM (2012): 19–24.

[28] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. "Flowvisor: A network virtualization layer." Technical Report, OpenFlow Switch Consortium (2009). Available from http://archive.openflow.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf (accessed February 17, 2015).

[29] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky. "Advanced study of SDN/OpenFlow controllers." Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia. Moscow: ACM (2013).

[30] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood. "On controller performance in software-defined networks." USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE) 54; San Jose, CA (2012).

[31] A. Voellmy and J. Wang. "Scalable software defined network controllers." Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. Helsinki, Finland: ACM (2012): 289–290.

[32] B. Heller, R. Sherwood, and N. McKeown. "The controller placement problem." Proceedings of the First Workshop on Hot Topics in Software Defined Networks. Helsinki, Finland: ACM (2012): 7–12.

[33] S. Schmid and J. Suomela. "Exploiting locality in distributed sdn control." Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. Hong Kong, China: ACM (2013): 121–126.

[34] M. Canini, P. Kuznetsov, D. Levin, and S. Schmid. "Software transactional networking: Concurrent and consistent policy composition." Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. Hong Kong, China: ACM (2013): 1–6.

[35] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. "OpenFlow: enabling innovation in campus networks." ACM SIGCOMM Computer Communication Review 38.2 (2008): 69–74.

[36] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. "Frenetic: A network programming language." ACM SIGPLAN Notices 46.9 (2011): 279–291.

[37] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker. "Composing software-defined networks." 10th USENIX Symposium on Networked Systems Design & Implementation; Lombard, IL (2013): 1–13.

[38] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. “Hedera: Dynamic flow scheduling for data center networks.” 7th USENIX Symposium on Networked Systems Design & Implementation 10; San Jose, CA (2010): 19–34.

[39] L. E. Li, Z. M. Mao, and J. Rexford. “Toward software-defined cellular networks.” European Workshop on Software Defined Networking (EWSDN). Darmstadt, Germany: IEEE (2012): 7–12.

[40] C. Cui, H. Deng, D. Telekom, U. Michel, and H. Damker. “Network functions virtualisation.” Available from http://portal.etsi.org/NFV/NFV_White_Paper.pdf (accessed 19 January 2015).

[41] Open Networking Foundation (ONF). Available from https://www.opennetworking.org (accessed 19 January 2015).

[42] IRTF. “Software-defined networking research group (SDNRG).” Available from https://irtf.org/sdnrg (accessed 19 January 2015).

[43] IETF. “Interface to the routing system (i2rs).” Available from http://datatracker.ietf.org/wg/i2rs/ (accessed 19 January 2015).

[44] IETF. “ALTO and software defined networking (SDN).” Available from http://www.ietf.org/proceedings/84/slides/slides-84-alto-5 (accessed 19 January 2015).

[45] IETF. “CDNI request routing with SDN.” Available from http://www.ietf.org/proceedings/84/slides/slides-84-cdni-1.pdf (accessed 19 January 2015).

[46] “ITU Telecommunication Standardization Sector.” Available from http://www.itu.int/en/ITU-T (accessed 19 January 2015).

[47] Optical Internetworking Forum (OIF). Available from http://www.oiforum.com (accessed 19 January 2015).

[48] “Broadband Forum and SDN.” Available from http://www.broadband-forum.org/technical/technicalwip.php (accessed 19 January 2015).

[49] “MEF—Metro Ethernet Forum.” Available from http://metroethernetforum.org (accessed 19 January 2015).

[50] 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.” Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM. Hong Kong, China: ACM (2013): 3–14.

[51] A. Vahdat. “Enter the Andromeda zone—Google Cloud Platform’s latest networking stack.” Available from http://googlecloudplatform.blogspot.gr/2014/04/enter-andromeda-zone-google-cloud-platforms-latest-networking-stack.html (accessed 19 January 2015).