Software-Defined Networks: on the road to the softwarization of networking

Fernando M. V. Ramos
LaSIGE/FCUL, University of Lisboa, Portugal
fvramos@ciencias.ulisboa.pt

Diego Kreutz, Paulo Verissimo
SnT/University of Luxembourg, Luxembourg
{Diego.Kreutz,Paulo.Verissimo}@uni.lu

ABSTRACT

Traditional IP networks are complex and hard to manage. The vertical integration of the infrastructure, with the control and data planes tightly coupled in network equipment, makes it a challenging task to build and maintain efficient networks in an era of cloud computing. Software-Defined Networking (SDN) breaks this coupling by segregating network control from routers and switches and by logically centralizing it in an external entity that resides in commodity servers. This way, SDN provides the flexibility required to dynamically program the network, promoting the "softwarization" of networking.

In this article we introduce this new paradigm and show how it breaks the status quo in networking. We present the most relevant building blocks of the infrastructure and discuss how SDN is leading to a horizontal industry based on programmable and open components. We pay particular attention to use cases that demonstrate how IT companies such as Google, Microsoft, and VMware are embracing SDN to operate efficient networks and offer innovative networking services.

1. INTRODUCTION

Traditional computer networks are *complex and very hard to manage* [1]. To express the desired policies, network operators need to configure each individual network device, one by one, either manually or with the use of low-level scripts. In addition to configuration complexity, network environments have to endure the dynamics of faults and adapt to load changes. Enforcing the required policies in such a dynamic environment is highly challenging. Current networks are also *vertically integrated*. The control plane (that decides how to handle network traffic) and the data plane (that forwards traffic according to the decisions made by the control plane) are bundled inside the networking devices. This is a fundamental obstacle that has led to the slow pace of innovation of networking infrastructure.

Software-Defined Networking (SDN) [2] is an emerging paradigm that promises to change the current state of affairs. SDN breaks the vertical integration by separating the network's control logic from the underlying routers and switches that forward the traffic. Network switches become simple forwarding devices and the control logic is implemented in a *logically centralized* controller, simplifying policy enforcement. While the controller can be implemented as a distributed system, network applications have access to a centralized programmatic model (a global network view), making it easier to reason about network behavior. A simplified view of this architecture is shown in Figure 1.

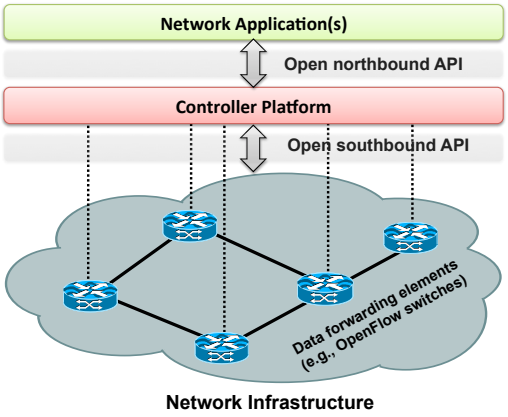

Figure 1: Simplified view of an SDN

The separation of the control plane and the data plane can be realized by means of a well-defined programming interface between the switches and the SDN controller. The controller exercises direct control over the state in the data-plane elements via this well-defined southbound interface. The most notable example of such interface is OpenFlow [3, 4]. An OpenFlow switch has one or more tables of packet-handling rules. Each rule matches a subset of the traffic and performs certain actions on the packets (dropping, forwarding to specific port(s), modifying headers, among others). Depending on the rules installed by a control application, an OpenFlow switch can – instructed by the controller – behave like a router, switch, firewall, load balancer, or perform other roles.

An important consequence of the software-defined networking principles is the *separation of concerns* intro-

duced between the *definition* of network policies, their *implementation* in switching hardware, and the *forwarding* of traffic. This separation allows modularity by breaking the network control problem into tractable pieces, and making it easier to create and introduce new abstractions in networking [5]. This is key to simplify network management and to facilitate innovation.

Although SDN and OpenFlow started as an academic experiment [3], they gained significant traction in the industry over the past few years. Most vendors of commercial switches now include OpenFlow support in their equipment. The ideas behind SDN have matured and evolved from an academic exercise to a commercial success. The world's largest IT companies have recently joined SDN consortia such as the ONF [4] and the OpenDaylight initiative [6] as an indication of the importance of SDN from an industrial perspective.

In this article we introduce Software-Defined Networking and show how this paradigm differentiates from traditional networking (Section 2). We present the most relevant building blocks of the SDN infrastructure (Section 3) and discuss how SDN is leading to a change in the networking industry (Section 4). To attest the success of SDN and its take up by the industry we present four demonstrative use cases (Section 5): Google's B4 and Microsoft's SWAN traffic engineering solutions [7], VMware's network virtualization platform [8, 9], and Statesman, a network state management service deployed by Microsoft in its data centers [10]. We conclude by speculating on what the future may yield for SDN (Section 6).

2. SOFTWARE-DEFINED NETWORKING

The term SDN was originally coined to represent the ideas and work around OpenFlow at Stanford University [11]. A software-defined network is a network architecture with four pillars:

1. The control and data planes are *decoupled*. Control functionality is removed from network devices that become simple (packet) forwarding elements.

2. Forwarding decisions are flow-based, instead of destination-based. A flow is broadly defined by a set of packet field values acting as a match (filter) criterion and a set of actions (instructions). The flow abstraction allows unifying the behavior of different types of network devices, including routers, switches, firewalls, and other middleboxes.

3. Control logic is moved to an external entity, the SDN controller. The controller is a software platform that runs on commodity server technology and provides the essential resources and abstractions to facilitate the programming of forwarding devices based on a logically centralized, abstract network view. Its purpose is therefore similar to that of a traditional operating system, but for networking resources.

4. The network is *programmable* through software applications running on top of the SDN controller. This is a fundamental characteristic of SDN, considered its main value proposition.

Following the concepts introduced in [5], an SDN can be defined by three fundamental abstractions: (*i*) forwarding, (*ii*) distribution, and (*iii*) specification. Ideally, the *forwarding abstraction* should allow any forwarding behavior desired by the network application (the control program) while hiding details of the underlying hardware. OpenFlow is one realization of such abstraction, which can be seen as the equivalent to a "device driver" in an operating system. The *distribution abstraction* should shield SDN applications from the vagaries of distributed state, logically centralizing the network control, guaranteeing its consistency. Its realization requires a common distribution layer, which in SDN resides in the controller. The last abstraction is *specification*, which should allow a network application to express the desired network behavior without being responsible for implementing that behavior itself.

3. SDN BUILDING BLOCKS

An SDN architecture can be depicted as a composition of different layers, as shown in Figure 2. Each layer has its own specific function.

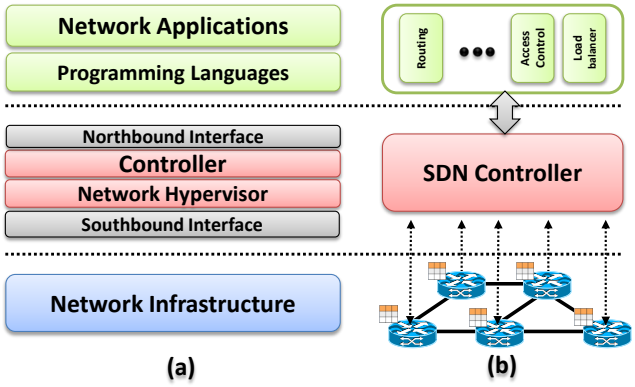

Figure 2: A layered view of SDN

3.1 Network infrastructure

An SDN infrastructure, similarly to a traditional network, is composed of a set of networking equipment (switches, routers, and middlebox appliances). The main difference resides in the fact that those traditional physical devices are now simple forwarding elements without (or with limited) embedded control or software to

take autonomous decisions. The network intelligence is removed from the data plane devices to a logically-centralized control system. An OpenFlow-enabled forwarding device is based on a pipeline of flow tables where each entry of a flow table has three parts: (1) a matching rule, (2) actions to be executed on matching packets, and (3) counters that keep statistics of matching packets.

3.2 Southbound interfaces

Southbound interfaces are the connecting bridges between control and forwarding elements, thus being the crucial instrument for clearly separating control and data plane functionality. OpenFlow is the most widely deployed open southbound standard for SDN. It provides a common specification to implement OpenFlow-enabled forwarding devices, and for the communication channel between data and control plane devices. The OpenFlow protocol provides three information sources for controllers. First, event-based messages are sent by forwarding devices to the controller when a link or port change is triggered. Second, flow statistics are generated by the forwarding devices and collected by the controller. Third, packet-in messages are sent by forwarding devices to the controller when they do not known what to do with an incoming packet or because there is an explicit "send to controller" action in the matched entry of the flow table. These information channels are the essential means to provide flow-level information to the controller.

OVSDB [12] is another type of southbound API, designed to provide advanced management capabilities for Open vSwitches [13]. Beyond OpenFlow's capabilities to configure flows and thus influence forwarding behavior, OVSDB allows the creation of tunnels, turning on or off certain features, get configuration data, etc. OVSDB is therefore a complementary protocol to OpenFlow for Open vSwitch.

3.3 Network hypervisor

Long standing virtualization primitives such as VLANs, NAT, and MPLS provide only limited forms of network virtualization. These solutions are also anchored on a box-by-box basis configuration. The programmability offered by an SDN has the means to provide full network virtualization – not only isolation of virtual networks, but also topology and addressing virtualization.

FlowVisor [14] was the earliest attempt to virtualize an SDN. This platform acts as a proxy between the controller and the forwarding devices to provide an abstraction layer that slices an OpenFlow data plane, allowing multiple controllers each to control its share (its slice) of a single physical infrastructure. The isolation properties provided by FlowVisor thus allow several networks to co-exist. OpenVirteX [15] extends FlowVisor to provide not only isolation of network control, but also topology and address virtualization. Contrary to these platforms that virtualize an SDN, VMware's Network Virtualization Platform (described in more detail in Section 5) provides full virtualization in data centers without requiring SDN-based hardware (the only requirement is that all servers are virtualized).

3.4 Controller

The controller is the fundamental element in an SDN architecture, as it is the key supporting piece for the control logic (applications) to generate the network configuration based on the policies defined by the network operator. Similar to a traditional operating system, the control platform abstracts the lower-level details of the interaction with forwarding devices.

There is a diverse set of controllers with different design and architectural choices [2]. Existing controllers can be categorized based on many aspects. From an architectural point of view, one of the most relevant is if they are centralized or distributed. A centralized controller (such as NOX [16]) is a single entity that manages all forwarding devices of the network. Naturally, it represents a single point of failure and may have scaling limitations. Contrary to a centralized design, a distributed controller (such as Onix [17]) can be scaled up to meet the requirements of potentially any environment, from small to large-scale networks.

3.5 Northbound interface

The north and southbound interfaces are two key abstractions of the SDN ecosystem. The southbound interface has already a widely accepted proposal (OpenFlow), but a common northbound interface is still an open issue. At this moment it may still be a bit too early to define a standard northbound interface, as use-cases are still being worked out [18]. Anyway, it is to be expected a common (or a *de facto*) northbound interface to arise as SDN evolves.

3.6 Programming languages

OpenFlow resembles an assembly-like machine language, essentially mimicking the behavior of forwarding devices, forcing developers to spend too much time on low-level details rather than on the problem to solve. Raw OpenFlow programs have to deal with hardware behavior details such as overlapping rules, the priority ordering of rules, and data-plane inconsistencies that arise from in-flight packets whose flow rules are under installation. The use of these low-level languages makes it difficult to reuse software, to create modular and extensive code, and leads to a more error-prone development process. Abstractions provided by high level network programming languages [19] can significantly help address many of the challenges of these lower-level

instruction sets.

3.7 Network applications

Network applications can be seen as the "network brains". They implement the control-logic that will be translated into commands to be installed in the data plane, dictating the behavior of the forwarding devices. Take a simple application as routing as an example. The logic of this application is to define the path through which packets will flow from a point A to a point B. To achieve this goal a routing application has to, based on the topology input, decide on the path to use and instruct the controller to install the respective forwarding rules in all forwarding devices on the chosen path, from A to B.

Existing SDN applications either perform traditional functionality such as routing, load balancing, and security policy enforcement, or explore novel approaches, such as reducing power consumption [20]. Other examples include fail-over and reliability functionalities to the data plane, end-to-end QoS enforcement, network virtualization, mobility management in wireless networks, among many others [2]. The variety of network applications, combined with real use case deployments, is expected to be one of the major forces on fostering a broad adoption of SDN.

4. AN INDUSTRY CHANGE

SDN is leading to a profound change in the networking industry. The emergence of this new paradigm gave rise to analogies with the computer industry [21]. In the 1980s, computers were based on specialized hardware, specialized operating systems and specialized applications, all from the same vender (Figure 3). This industry was thus vertically integrated, closed, proprietary, and small. The pace of innovation was relatively slow.

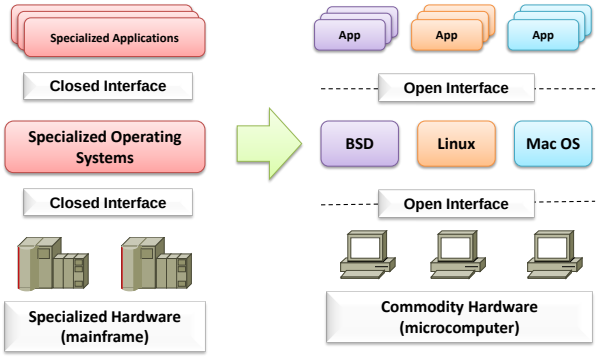

Figure 3: Change in the computer industry

The development of the microprocessor has provoked a fundamental change, leading to the commoditization of computers. The microprocessor and the creation of open interfaces to program it led, over time, to the development of many Operating Systems. In addition, the open interfaces to program on the operating systems led to a myriad of applications developed by hundreds to thousands of different companies. In summary, the commoditization of hardware and the creation of open interfaces "horizontalized" the computer industry, increasing its pace of innovation and ultimately resulting in a huge industry.

Something similar is happening in networking today (Figure 4). The networking industry was, until recently, based on specialized hardware, specialized control planes and specialized control programs. As computers in the 1980s, this naturally led to a vertically integrated industry, closed, proprietary, with a slow pace of innovation.

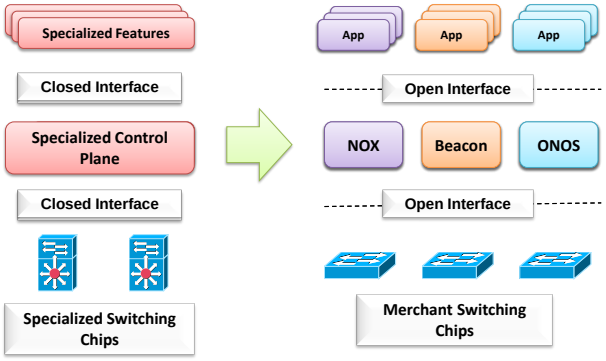

Figure 4: Change in the networking industry

The availability of merchant switching chips is leading to the commoditization of networking equipment, a trend comparable to the one originated by the microprocessor. The advent of open interfaces to the switches, such as OpenFlow, has led to the emergence of a large number of controllers and network control applications. This trend shows no sign of decreasing, and the materialization of open northbound APIs will arguably foster the emergence of an even larger number of network applications. As with compute, the hardware commoditization and the availability of open interfaces to program it are leading to a "horizontalization" of the networking industry. Indeed, we are already experiencing a surge in network innovation (network virtualization is a good example) and the emergence of a significant number of networking start-ups.

4.1 Is such change really happening?

In one word, we claim: yes. Table 1 shows a (non-exhaustive) list of a) switches with OpenFlow support, b) SDN controllers and c) applications. As is clear, the trend illustrated in the rightmost part of Figure 4 is being fulfilled. Most vendors of commercial switches already include OpenFlow support in their equipment. Also, the number of controllers is already significant, as

is the number of control programs.

The world's largest IT companies are now part of a number of SDN consortia, which is demonstrative of its impact in the industry. The Open Networking Foundation (ONF) [4], for instance, is an organization dedicated to the promotion and adoption of SDN through open standards development. Its signature accomplishment is the introduction of the OpenFlow standard. The ONF already has over 150 member companies including dozens of start-ups exclusively dedicated to SDN technology. Another good example of this trend is the OpenDaylight initiative [6], a collaborative open source project hosted by The Linux Foundation. The goal of the project is to design and implement an open source SDN platform. For this purpose its members – which already add up to more than 40 – commit to donate software and engineering resources for the project.

5. USE CASES

Another indication of the importance of SDN from an industrial perspective is its adoption by IT companies such as Google, Microsoft, and VMware. The use cases we present in this section are paradigmatic in that respect.

5.1 Traffic engineering

The wide area network (WAN) that connects the datacenters of cloud providers is critical for the performance of Internet services. WAN links are very expensive, and to guarantee the required performance WAN routers consist of high-end, specialized equipment. To compound the problem, providers are unable to fully leverage their high investment on the infrastructure. Given the extreme lack of efficiency of these inter-datacenter networks they are provisioned for an average utilization of 30-40%.

Recognizing this problem, Google and Microsoft have deployed large-scale SDN infrastructures for boosting the utilization of their inter-datacenter links. These systems – Google's B4 [7] and Microsoft's SWAN [22] – leverage on SDN to substantially increase the utilization of their WAN links. In particular, the logical centralization of network control enables centralized traffic engineering and simpler traffic prioritization, making it possible to have these links used up to nearly 100% utilization without impairing the quality of service. Besides avoiding link usage inefficiencies, the network equipment used in these solutions is built from merchant switch silicon, further reducing costs and increasing flexibility.

5.2 Network management

Datacenter operators have developed, over the years, automated systems for managing their traffic and infrastructure, in order to keep their networks running smoothly. Management applications are sophisticated in their own right. More challenging still, the different applications that continuously run in these networks (it is common to have, for instance, a traffic engineering application alongside another to mitigate link failures) can conflict with each other.

Statesman [10] is an SDN-based solution deployed in Microsoft Azure datacenters that provides a state management abstraction aimed to solve this problem. Specifically, this service introduces two main ideas to simplify the design and deployment of network management applications. First, it maintain different views of network state to prevent conflicts and invariant violations. Applications cannot change the state of the network directly. Instead, each application applies its own logic to the network's *observed* state to generate *proposed* states that may change one or more state variables. Statesman merges all proposed states into one *target* state while ensuring safety and performance invariants. Second, Statesman uses a dependency model to capture the domain-specific dependencies among state variables.

5.3 Network virtualization

As mentioned before, networking has long supported virtualized primitives such as virtual links (tunnels such as MPLS) or broadcast domains (VLANs to slice L2 networks). Traditional network virtualization technologies do not provide full network virtualization and require significant manual network management on a box-by-box basis. As a consequence, current network provisioning can take months, while compute provisioning takes only minutes.

VMware has recently started offering a network virtualization solution using SDN principles, the Network Virtualization Platform (NVP) [8] (commercialized as NSX [9]). NVP is designed for multi-tenant datacenters and has been deployed in dozens of production environments over the last few years. NVP is an edge-based implementation that does not require SDN-based hardware. The only requirement is for the datacenter computing resources controlled by NVP to be fully virtualized. In NVP a centralized controller cluster is responsible for configuring the virtual switches in the hypervisor of every host (OpenvSwitch [13]) with the appropriate logical forwarding. The solution then leverages on a set of tunnels between every pair of host-hypervisors to forward traffic.

6. THE FUTURE OF SDN

We conclude this article with our view on some important challenges faced by SDN. The way they will be addressed will certainly shape its future.

First, we expect the SDN software stack to evolve in at least two ways: in offering new network abstractions

Table 1: List of switches with OpenFlow support, controllers and applications

Switch Manufacturers	Controllers	Applications
Arista Networks	Beacon	Access control
Big Switch Networks	Floodlight	Energy-efficient network
Brocade Networks	Mul	Integrated security
Cisco	NOX	Load balancing
Dell	ONIX	Mitigation of DoS attacks
Extreme Networks	ONOS	Network monitoring
HP	Open Daylight	Network virtualization
Huawei	OpenContrail	Network-as-a-Service
IBM	POX	QoS policy enforcement
Juniper	ProgrammableFlow	Traffic engineering
NEC	Rosemary	Traffic steering
NetGear	Ryu	User mobility
Pica8	Trema	VM migration

and in increasing its security and resilience. The work around the Frenetic project [19] has pointed the direction for the former, but new abstractions will eventually arise, including for network monitoring, traffic engineering, security, and others. We also expect the security and dependability of the infrastructure to be an increasing focus of concern in the future [23].

The initial deployment of SDNs has been mostly focused in data center environments. It is only natural for SDN to expand to other, less homogeneous, environments, ranging from carrier-grade backbones and cellular networks to inter-domain routing (e.g., Internet exchange points).

The problem of migration from traditional networks to SDN and the emergence of hybrid deployments is a related challenge. Panopticon [24] is one example of an architecture and methodology that implements SDN inside enterprise legacy networks. We anticipate other such solutions to appear to increase the pace of SDN adoption.

Finally, the integration of Network Function Virtualization (NFV) with SDN is another important challenge. NFV is an emergent concept that proposes the use of virtualization technologies to virtualize network node functions, therefore enabling middlebox functionality to run on virtual machines. Its integration with SDN is increasingly being considered fundamental by the networking industry – it is one of the main goals of the OpenDaylight project, for instance – as a promoter of the softwarization of the complete networking infrastructure.

7. REFERENCES

References

[1] T. Benson, A. Akella, and D. Maltz. "Unraveling the Complexity of Network Management". In: *NSDI*. NSDI'09. Boston, Massachusetts, 2009, pp. 335–348.

[2] D. Kreutz et al. "Software-Defined Networking: A Comprehensive Survey". In: *Proceedings of the IEEE* 103.1 (2015), pp. 14–76.

[3] N. McKeown et al. "OpenFlow: enabling innovation in campus networks". In: *SIGCOMM CCR* 38.2 (Mar. 2008), pp. 69–74.

[4] ONF. *Open Networking Foundation*. `http://www.opennetworking.org/`. 2014.

[5] S. Shenker. *The Future of Networking, and the Past of Protocols*. 2011.

[6] OpenDaylight. *OpenDaylight: A Linux Foundation Collaborative Project*. `http://www.opendaylight.org`. 2013.

[7] S. Jain et al. "B4: experience with a globally-deployed software defined wan". In: *SIGCOMM*. SIGCOMM '13. Hong Kong, China: ACM, 2013, pp. 3–14.

[8] T. Koponen et al. "Network Virtualization in Multi-tenant Datacenters". In: *NSDI*. Seattle, WA, Apr. 2014, pp. 203–216.

[9] VMware, Inc. *VMware NSX Virtualization Platform*. 2013.

[10] P. Sun et al. "A Network-state Management Service". In: *SIGCOMM*. SIGCOMM '14. Chicago, Illinois, USA: ACM, 2014, pp. 563–574.

[11] K. Greene. *MIT Tech Review 10 Breakthrough Technologies: Software-defined Networking*. goo.gl/KVDOD1. 2009.

[12] B. Pfaff and B. Davie. *The Open vSwitch Database Management Protocol*. RFC 7047 (Informational). Internet Engineering Task Force, Dec. 2013.

[13] B. Pfaff et al. "The Design and Implementation of Open vSwitch". In: *NSDI*. Oakland, CA: USENIX Association, May 2015, pp. 117–130.

[14] R. Sherwood et al. "Can the production network be the testbed?" In: *OSDI*. OSDI'10. Vancouver, BC, Canada, 2010, pp. 1–6.

[15] A. Al-Shabibi et al. "OpenVirteX: Make Your Virtual SDNs Programmable". In: *HotSDN*. HotSDN '14. Chicago, Illinois, USA: ACM, 2014, pp. 25–30.

[16] N. Gude et al. "NOX: towards an operating system for networks". In: *CCR* (2008).

[17] T. Koponen et al. "Onix: a distributed control platform for large-scale production networks". In: *OSDI*. OSDI'10. Vancouver, BC, Canada: USENIX Association, 2010, pp. 1–6.

[18] J. Dix. *Clarifying the role of software-defined networking northbound APIs*. goo.gl/7MXw4o. 2013.

[19] N. Foster et al. "Frenetic: a network programming language". In: *SIGPLAN Not.* (2011).

[20] B. Heller et al. "ElasticTree: saving energy in data center networks". In: *NSDI*. NSDI'10. San Jose, California: USENIX Association, 2010, pp. 17–17.

[21] N. Mckeown. *How SDN will Shape Networking*. 2011.

[22] C.-Y. Hong et al. "Achieving high utilization with software-driven WAN". In: *SIGCOMM*. SIGCOMM '13. Hong Kong, China: ACM, 2013, pp. 15–26.

[23] D. Kreutz, F. M. Ramos, and P. Verissimo. "Towards secure and dependable software-defined networks". In: *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. HotSDN '13. Hong Kong, China: ACM, 2013, pp. 55–60.

[24] D. Levin et al. "Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks". In: *USENIX ATC*. USENIX ATC '14. Philadelphia, PA: USENIX Association, 2014, pp. 333–345.