This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks

Bruno Astuto A. Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti

***Abstract*—The idea of *programmable networks* has recently re-gained considerable momentum due to the emergence of the Software-Defined Networking (SDN) paradigm. SDN, often referred to as a "radical new idea in networking", promises to dramatically simplify network management and enable innovation through network programmability. This paper surveys the state-of-the-art in programmable networks with an emphasis on SDN. We provide a historic perspective of programmable networks from early ideas to recent developments. Then we present the SDN architecture and the OpenFlow standard in particular, discuss current alternatives for implementation and testing of SDN-based protocols and services, examine current and future SDN applications, and explore promising research directions based on the SDN paradigm.**

***Index Terms*—Software-Defined Networking, programmable networks, survey, data plane, control plane, virtualization.**

I. Introduction

COMPUTER networks are typically built from a large number of network devices such as routers, switches and numerous types of middleboxes (i.e., devices that manipulate traffic for purposes other than packet forwarding, such as a firewall) with many complex protocols implemented on them. Network operators are responsible for configuring policies to respond to a wide range of network events and applications. They have to manually transform these high level-policies into low-level configuration commands while adapting to changing network conditions. Often, they also need to accomplish these very complex tasks with access to very limited tools. As a result, network management and performance tuning is quite challenging and thus error-prone. The fact that network devices are usually vertically-integrated *black boxes* exacerbates the challenge network operators and administrators face.

Another almost unsurmountable challenge network practitioners and researchers face has been referred to as "Internet ossification". Because of its huge deployment base and the fact it is considered part of our society's critical infrastructure (just like transportation and power grids), the Internet has become extremely difficult to evolve both in terms of its physical infrastructure as well as its protocols and performance. However, as current and emerging Internet applications and services become increasingly more complex and demanding, it is imperative that the Internet be able to evolve to address these new challenges.

The idea of "programmable networks" has been proposed as a way to facilitate network evolution. In particular, Software Defined Networking (SDN) is a new networking paradigm in which the forwarding hardware is decoupled from control decisions. It promises to dramatically simplify network management and enable innovation and evolution. The main idea is to allow software developers to rely on network resources in the same easy manner as they do on storage and computing resources. In SDN, the network intelligence is logically centralized in software-based controllers (the control plane), and network devices become simple packet forwarding devices (the data plane) that can be programmed via an open interface (e.g., ForCES [1], OpenFlow [2], etc).

SDN is currently attracting significant attention from both academia and industry. A group of network operators, service providers, and vendors have recently created the Open Network Foundation [3], an industrial-driven organization, to promote SDN and standardize the OpenFlow protocol [2]. On the academic side, the OpenFlow Network Research Center [4] has been created with a focus on SDN research. There have also been standardization efforts on SDN at the IETF and IRTF and other standards producing organizations.

The field of software defined networking is quite recent, yet growing at a very fast pace. Still, there are important research challenges to be addressed. In this paper, we survey the state-of-the-art in programmable networks by providing a historic perspective of the field and also describing in detail the SDN paradigm and architecture. The paper is organized as follows: in Section II, it begins by describing early efforts focusing on programmable networks. Section III provides an overview of SDN and its architecture. It also describes the OpenFlow protocol. Section IV describes existing platforms for developing and testing SDN solutions including emulation and simulation tools, SDN controller implementations, as well as verification and debugging tools. In Section V, we discuss several SDN applications in areas such as data centers and wireless networking. Finally, Section VI discusses research challenges and future directions.

II. Early Programmable Networks

SDN has great potential to change the way networks operate, and OpenFlow in particular has been touted as a "radical new idea in networking" [5]. The proposed benefits range from centralized control, simplified algorithms, commoditizing network hardware, eliminating middleboxes, to enabling the design and deployment of third-party 'apps'.

While OpenFlow has received considerable attention from industry, it is worth noting that the idea of programmable networks and decoupled control logic has been around for many years. In this section, we provide an overview of early programmable networking efforts, precursors to the current

Manuscript received June 14, 2013; revised October 28, 2013.

B. A. A. Nunes, X. Nguyen and T. Turletti are with INRIA, France (e-mail: {bruno.astuto-arouche-nunes, xuan-nam.nguyen, thierry.turletti}@inria.fr)

M. Mendonca and K. Obraczka are with UC Santa Cruz (e-mail: {msm, katia}@soe.ucsc.edu)

Digital Object Identifier 10.1109/SURV.2014.012214.00180

1553-877X/14/$31.00 © 2014 IEEE

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

SDN paradigm that laid the foundation for many of the ideas we are seeing today.

a) **Open Signaling:** The Open Signaling (OPENSIG) working group began in 1995 with a series of workshops dedicated to "making ATM, Internet and mobile networks more open, extensible, and programmable" [6]. They believed that a separation between the communication hardware and control software was necessary but challenging to realize; this is mainly due to vertically integrated switches and routers, whose closed nature made the rapid deployment of new network services and environments impossible. The core of their proposal was to provide access to the network hardware via open, programmable network interfaces; this would allow the deployment of new services through a distributed programming environment.

Motivated by these ideas, an IETF working group was created, which led to the specification of the General Switch Management Protocol (GSMP) [7], a general purpose protocol to control a label switch. GSMP allows a controller to establish and release connections across the switch, add and delete leaves on a multicast connection, manage switch ports, request configuration information, request and delete reservation of switch resources, and request statistics. The working group is officially concluded and the latest standards proposal, GSMPv3, was published in June 2002.

b) **Active Networking:** Also in the mid 1990s, the Active Networking [8], [9] initiative proposed the idea of a network infrastructure that would be programmable for customized services. There were two main approaches being considered, namely: (1) user-programmable switches, with in-band data transfer and out-of-band management channels; and (2) capsules, which were program fragments that could be carried in user messages; program fragments would then be interpreted and executed by routers. Despite considerable activity it motivated, Active Networking never gathered critical mass and transferred to widespread use and industry deployment, mainly due to practical security and performance concerns [10].

c) **DCAN:** Another initiative that took place in the mid 1990s is the Devolved Control of ATM Networks (DCAN) [11]. The aim of this project was to design and develop the necessary infrastructure for scalable control and management of ATM networks. The premise is that control and management functions of the many devices (ATM switches in the case of DCAN) should be decoupled from the devices themselves and delegated to external entities dedicated to that purpose, which is basically the concept behind SDNs. DCAN assumes a minimalist protocol between the manager and the network, in the lines of what happens today in proposals such as OpenFlow. More on the DCAN project can be found at [12].

Still in the lines of SDNs and the proposed decoupling of control and data plane over ATM networks, amongst others, in the work proposed in [13] multiple heterogeneous control architectures are allowed to run simultaneously over single physical ATM network by partitioning the resources of that switch between those controllers.

d) **4D Project:** Starting in 2004, the 4D project [14], [15], [16] advocated a clean slate design that emphasized separation between the routing decision logic and the protocols governing the interaction between network elements. It proposed giving the "decision" plane a global view of the network, serviced by a "dissemination" and "discovery" plane, for control of a "data" plane for forwarding traffic. These ideas provided direct inspiration for later works such as NOX [17], which proposed an "operating system for networks" in the context of an OpenFlow-enabled network.

e) **NETCONF:** In 2006, the IETF Network Configuration Working Group proposed NETCONF [18] as a management protocol for modifying the configuration of network devices. The protocol allowed network devices to expose an API through which extensible configuration data could be sent and retrieved.

Another management protocol, widely deployed in the past and used until today, is the SNMP [19]. SNMP was proposed in the late 80's and proved to be a very popular network management protocol, which uses the Structured Management Interface (SMI) to fetch data contained in the Management Information Base (MIB). It could be used as well to change variables in the MIB in order to modify configuration settings. It later became apparent that in spite of what it was originally intended for, SNMP was not being used to configure network equipment, but rather as a performance and fault monitoring tool. Moreover, multiple shortcomings were detected in the conception of SNMP, the most notable of which was its lack of strong security. This was addressed in a later version of the protocol.

NETCONF, at the time it was proposed by IETF, was seen by many as a new approach for network management that would fix the aforementioned shortcomings in SNMP. Although the NETCONF protocol accomplishes the goal of simplifying device (re)configuration and acts as a building block for management, there is no separation between data and control planes. The same can be stated about SNMP. A network with NETCONF should not be regarded as fully programmable as any new functionality would have to be implemented at both the network device and the manager so that any new functionality can be provided; furthermore, it is designed primarily to aid automated configuration and not for enabling direct control of state nor enabling quick deployment of innovative services and applications. Nevertheless, both NETCONF and SNMP are useful management tools that may be used in parallel on hybrid switches supporting other solutions that enable programmable networking.

The NETCONF working group is currently active and the latest proposed standard was published in June 2011.

f) **Ethane:** The immediate predecessor to OpenFlow was the SANE / Ethane project [20], which, in 2006, defined a new architecture for enterprise networks. Ethane's focus was on using a centralized controller to manage policy and security in a network. A notable example is providing identity-based access control. Similar to SDN, Ethane employed two components: a *controller* to decide if a packet should be forwarded, and an *Ethane switch* consisting of a flow table and a secure channel to the controller.

Ethane laid the foundation for what would become Software-Defined Networking. To put Ethane in the context of today's SDN paradigm, Ethane's identity-based access control

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

would likely be implemented as an application on top of an SDN controller such as NOX [17], Maestro [21], Beacon [22], SNAC [23], Helios [24], etc.

III. Software-Defined Networking Architecture

Data communication networks typically consist of end-user devices, or hosts interconnected by the network infrastructure. This infrastructure is shared by hosts and employs switching elements such as routers and switches as well as communication links to carry data between hosts. Routers and switches are usually "closed" systems, often with limited- and vendor-specific control interfaces. Therefore, once deployed and in production, it is quite difficult for current network infrastructure to evolve; in other words, deploying new versions of existing protocols (e.g., IPv6), not to mention deploying completely new protocols and services is an almost insurmountable obstacle in current networks. The Internet, being a network of networks, is no exception.

As mentioned previously, the so-called Internet "ossification" [2] is largely attributed to the tight coupling between the data– and control planes which means that decisions about data flowing through the network are made on-board each network element. In this type of environment, the deployment of new network applications or functionality is decidedly non-trivial, as they would need to be implemented directly into the infrastructure. Even straightforward tasks such as configuration or policy enforcement may require a good amount of effort due to the lack of a common control interface to the various network devices. Alternatively, workarounds such as using "middleboxes" (e.g., firewalls, Intrusion Detection Systems, Network Address Translators, etc.) overlayed atop the underlying network infrastructure have been proposed and deployed as a way to circumvent the network ossification effect. Content Delivery Networks (CDNs) [25] are a good example.

Software-Defined Networking was developed to facilitate innovation and enable simple programmatic control of the network data-path. As visualized in Figure 1, the separation of the forwarding hardware from the control logic allows easier deployment of new protocols and applications, straightforward network visualization and management, and consolidation of various middleboxes into software control. Instead of enforcing policies and running protocols on a convolution of scattered devices, the network is reduced to "simple" forwarding hardware and the decision-making network controller(s).

A. Current SDN Architectures

In this section, we review two well-known SDN architectures, namely ForCES [1] and Openflow [2]. Both OpenFlow and ForCES follow the basic SDN principle of separation between the control and data planes; and both standardize information exchange between planes. However, they are technically very different in terms of design, architecture, forwarding model, and protocol interface.

1) **ForCES:** The approach proposed by the IETF ForCES (Forwarding and Control Element Separation) Working Group, redefines the network device's internal architecture having the control element separated from the forwarding element. However, the network device is still represented as a single entity. The driving use case provided by the working group considers the desire to combine new forwarding hardware with third-party control within a single network device. Thus, the control and data planes are kept within close proximity (e.g., same box or room). In contrast, the control plane is ripped entirely from the network device in "OpenFlow-like" SDN systems.

ForCES defines two logic entities called the Forwarding Element (FE) and the Control Element (CE), both of which implement the ForCES protocol to communicate. The FE is responsible for using the underlying hardware to provide per-packet handling. The CE executes control and signaling functions and employs the ForCES protocol to instruct FEs on how to handle packets. The protocol works based on a master-slave model, where FEs are slaves and CEs are masters.

An important building block of the ForCES architecture is the LFB (Logical Function Block). The LFB is a well-defined functional block residing on the FEs that is controlled by CEs via the ForCES protocol. The LFB enables the CEs to control the FEs' configuration and how FEs process packets.

ForCES has been undergoing standardization since 2003, and the working group has published a variety of documents including: an applicability statement, an architectural framework defining the entities and their interactions, a modeling language defining the logical functions within a forwarding element, and the protocol for communication between the control and forwarding elements within a network element. The working group is currently active.

2) **OpenFlow:** Driven by the SDN principle of decoupling the control and data forwarding planes, OpenFlow [2], like ForCES, standardizes information exchange between the two planes.

In the OpenFlow architecture, illustrated in Figure 2, the forwarding device, or OpenFlow switch, contains one or more *flow tables* and an abstraction layer that securely communicates with a *controller* via OpenFlow protocol. Flow tables consist of flow entries, each of which determines how packets belonging to a flow will be processed and forwarded. Flow entries typically consist of: (1) *match fields*, or matching rules, used to match incoming packets; match fields may contain information found in the packet header, ingress port, and metadata; (2) *counters*, used to collect statistics for the particular flow, such as number of received packets, number of bytes and duration of the flow; and (3) a *set of instructions*, or *actions*, to be applied upon a match; they dictate how to handle matching packets.

Upon a packet arrival at an OpenFlow switch, packet header fields are extracted and matched against the matching fields portion of the flow table entries. If a matching entry is found, the switch applies the appropriate set of instructions, or actions, associated with the matched flow entry. If the flow table look-up procedure does not result on a match, the action taken by the switch will depend on the instructions defined by the *table-miss* flow entry. Every flow table must contain a

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

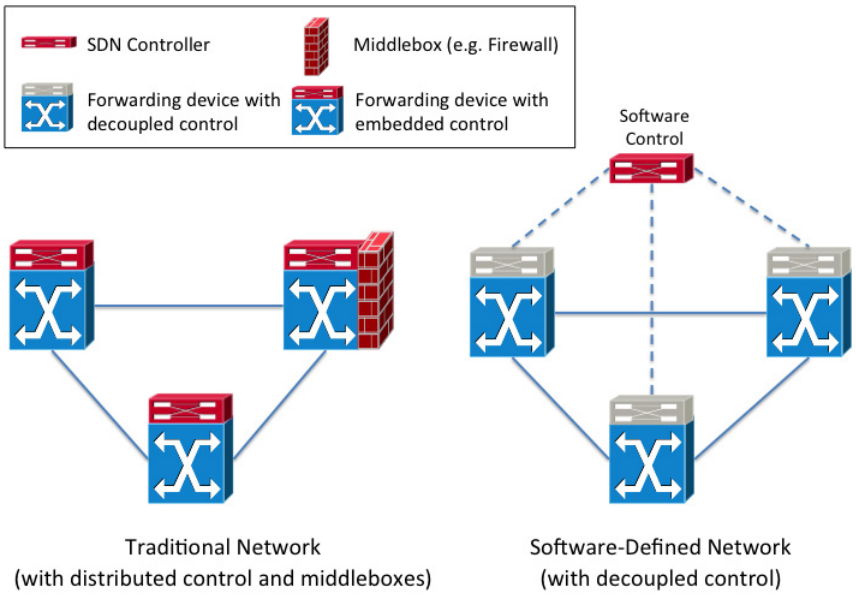

Fig. 1. The SDN architecture decouples control logic from the forwarding hardware, and enables the consolidation of middleboxes, simpler policy management, and new functionalities. The solid lines define the data-plane links and the dashed lines the control-plane links.

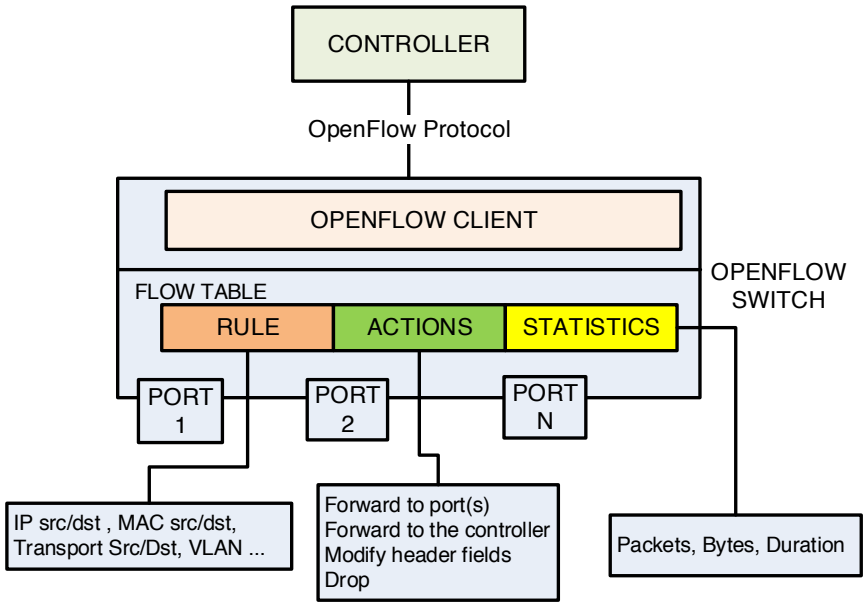

Fig. 2. Communication between the controller and the forwarding devices happens via OpenFlow protocol. The flow tables are composed by matching rules, actions to be taken when the flow matches the rules, and counters for collecting flow statistics.

table-miss entry in order to handle table misses. This particular entry specifies a set of actions to be performed when no match is found for an incoming packet, such as dropping the packet, continue the matching process on the next flow table, or forward the packet to the controller over the OpenFlow channel. It is worth noting that from version 1.1 OpenFlow supports multiple tables and pipeline processing. Another possibility, in the case of *hybrid switches*, i.e., switches that have both OpenFlow– and non-OpenFlow ports, is to forward non-matching packets using regular IP forwarding schemes.

The communication between controller and switch happens via OpenFlow protocol, which defines a set of messages that can be exchanged between these entities over a secure channel. Using the OpenFlow protocol a remote controller can, for example, add, update, or delete flow entries from the switch's flow tables. That can happen *reactively* (in response to a packet arrival) or *proactively*.

3) **Discussion:** In [26], the similarities and differences between ForCES and OpenFlow are discussed. Among the differences, they highlight the fact that the forwarding model used by ForCES relies on the Logical Function Blocks (LFBs), while OpenFlow uses flow tables. They point out that in OpenFlow actions associated with a flow can be combined to provide greater control and flexibility for the purposes

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

of network management, administration, and development. In ForCES the combination of different LFBs can also be used to achieve the same goal.

We should also re-iterate that ForCES does not follow the same SDN model underpinning OpenFlow, but can be used to achieve the same goals and implement similar functionality [26].

The strong support from industry, research, and academia that the Open Networking Foundation (ONF) and its SDN proposal, OpenFlow, has been able to gather is quite impressive. The resulting critical mass from these different sectors has produced a significant number of deliverables in the form of research papers, reference software implementations, and even hardware. So much so that some argue that OpenFlow's SDN architecture is the current SDN de-facto standard. In line with this trend, the remainder of this section focuses on OpenFlow's SDN model. More specifically, we will describe the different components of the SDN architecture, namely: the switch, the controller, and the interfaces present on the controller for communication with forwarding devices (southbound communication) and network applications (northbound communication). Section IV also has an OpenFlow focus as it describes existing platforms for SDN development and testing, including emulation and simulation tools, SDN controller implementations, as well as verification and debugging tools. Our discussion of future SDN applications and research directions is more general and is SDN architecture agnostic.

B. Forwarding Devices

The underlaying network infrastructure may involve a number of different physical network equipment, or forwarding devices such as routers, switches, virtual switches, wireless access points, to name a few. In a software-defined network, such devices are often represented as basic forwarding hardware accessible via an open interface at an abstraction layer, as the control logic and algorithms are off-loaded to a controller. Such forwarding devices are commonly referred to, in SDN terminology, simply as "switches", as illustrated in Figure 3.

In an OpenFlow network, switches come in two varieties: pure and hybrid. Pure OpenFlow switches have no legacy features or on-board control, and completely rely on a controller for forwarding decisions. Hybrid switches support OpenFlow in addition to traditional operation and protocols. Most commercial switches available today are hybrids.

1) Processing Forwarding Rules: Flow-based SDN architectures such as OpenFlow may utilize additional forwarding table entries, buffer space, and statistical counters that are difficult to implement in traditional ASIC switches. Some recent proposals [27], [28] have advocated adding a general-purpose CPU, either on-switch or nearby, that may be used to supplement or take over certain functions and reduce the complexity of the ASIC design. This would have the added benefit of allowing greater flexibility for on-switch processing as some aspects would be software-defined.

In [29], network processor based acceleration cards were used to perform OpenFlow switching. They proposed and described the design options and reported results that showed a 20% reduction on packet delay. In [30], an architectural design to improve look-up performance of OpenFlow switching in Linux was proposed. Preliminary results reported showed a packet switching throughput increase of up to 25% compared to the throughput of regular software-based OpenFlow switching. Another study on data-plane performance over Linux based Openflow switching was presented in [31], which compared OpenFlow switching, layer-2 Ethernet switching and layer-3 IP routing performance. Fairness, forwarding throughput and packet latency in diverse load conditions were analyzed. In [32], a basic model for the forwarding speed and blocking probability of an OpenFlow switch was derived, while the parameters for the model were drawn from measurements of switching times of current OpenFlow hardware, combined with an OpenFlow controller.

2) Installing Forwarding Rules: Another issue regarding the scalability of an OpenFlow network is memory limitation in forwarding devices. OpenFlow rules are more complex than forwarding rules in traditional IP routers. They support more flexible matchings and matching fields and also different actions to be taken upon packet arrival. A commodity switch normally supports between a few thousand up to tens of thousands forwarding rules [33]. Also, Ternary Content-Addressable Memory (TCAM) has been used to support forwarding rules, which can be expensive and power-hungry. Therefore, the rule space is a bottleneck to the scalability of OpenFlow, and the optimal use of the rule space to serve a scaling number of flow entries while respecting network policies and constraints is a challenging and important topic.

Some proposals address memory limitations in OpenFlow switches. Devoflow [34] is an extension to OpenFlow for high-performance networks. It handles mice flows (i.e. short flows) at the OpenFlow switch and only invokes the controller in order to handle elephant flows (i.e larger flows). The performance evaluation conducted in [34] showed that Devoflow uses 10 to 53 times less flow table space. In DIFANE [35], "ingress" switches redirect packets to "authority" switches that store all the forwarding rules while ingress switches cache flow table rules for future use. The controller is responsible for partitioning rules over authority switches.

Palette [36] and One Big Switch [37] address the rule placement problem. Their goal is to minimize the number of rules that need to be installed in forwarding devices and use end-to-end policies and routing policies as input to a rule placement optimizer. End-to-end policies consist of a set of prioritized rules dictating, for example, access control and load balancing, while viewing the whole network as a single virtual switch. Routing policies, on the other hand, dictate through what paths traffic should flow in the network. The main idea in Palette is to partition end-to-end policies into sub tables and then distribute them over the switches. Their algorithm consists of two steps: determine the number k of tables needed and then partition the rules set over k tables. One Big Switch, on the other hand, solves the rule placement problem separately for each path, choosing the paths based on network metrics (e.g. latency, congestion and bandwidth), and then combining the result to reach a global solution.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

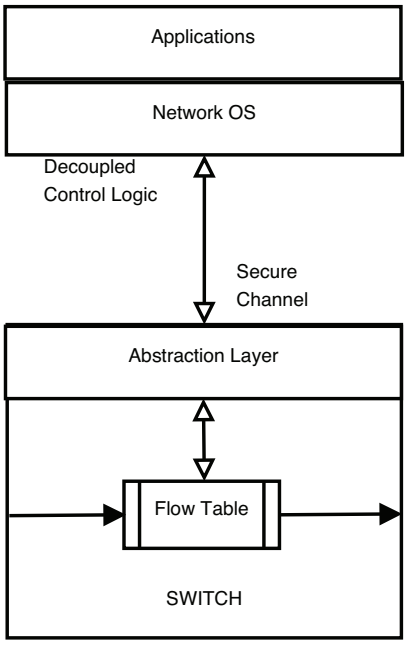

Fig. 3. The separated control logic can be viewed as a network operating system, upon which applications can be built to "program" the network.

C. The Controller

The decoupled system has been compared to an operating system [17], in which the controller provides a programmatic interface to the network. That can be used to implement management tasks and offer new functionalities. A layered view of this model is illustrated in Figure 3. This abstraction assumes the control is centralized and applications are written as if the network is a single system. It enables the SDN model to be applied over a wide range of applications and heterogeneous network technologies and physical media such as wireless (e.g. 802.11 and 802.16), wired (e.g. Ethernet) and optical networks.

As a practical example of the layering abstraction accessible through open application programming interfaces (APIs), Figure 4 illustrates the architecture of an SDN controller based on the OpenFlow protocol. This specific controller is a fork of the Beacon controller [22] called Floodlight [38]. In this figure it is possible to observe the separation between the controller and the application layers. Applications can be written in Java and can interact with the built-in controller modules via a JAVA API. Other applications can be written in different languages and interact with the controller modules via the REST API. This particular example of an SDN controller allows the implementation of built-in modules that can communicate with their implementation of the OpenFlow controller (i.e. OpenFlow Services). The controller, on the other hand, can communicate with the forwarding devices via the OpenFlow protocol through the abstraction layer present at the forwarding hardware, illustrated in Figure 3.

While the aforementioned layering abstractions accessible via open APIs allow the simplification of policy enforcement and management tasks, the bindings must be closely maintained between the control and the network forwarding elements. The choices made while implementing such layering architectures can dramatically influence the performance and scalability of the network. In the following, we address some such scalability concerns and go over some proposals that aim on overcoming these challenges. We leave a more detailed discussion on the application layer and the implementation of services and policy enforcement to Section VI-C.

1) Control Scalability: An initial concern that arises when offloading control from the switching hardware is the scalability and performance of the network controller(s). The original Ethane [20] controller, hosted on a commodity desktop machine, was tested to handle up to 11,000 new flow requests per second and responded within 1.5 milliseconds. A more recent study [39] of several OpenFlow controller implementations (NOX-MT, Maestro, Beacon), conducted on a larger emulated network with 100,000 endpoints and up to 256 switches, found that all were able to handle at least 50,000 new flow requests per second in each of the tested scenarios. On an eight-core machine, the multi-threaded NOX-MT implementation handled 1.6 million new flow requests per second with an average response time of 2 milliseconds. As the results show, a single controller is able to handle a surprising number of new flow requests, and should be able to manage all but the largest networks. Furthermore, new controllers under development such as McNettle [40] target powerful multicore servers and are being designed to scale up to large data center workloads (around 20 million flows requests per second and up to 5000 switches). Nonetheless, multiple controllers may be used to reduce latency or increase fault tolerance.

A related concern is the controller placement problem [41], which attempts to determine both the optimal number of controllers and their location within the network topology, often choosing between optimizing for average and worst case latency. The latency of the link used for communication between controller and switch is of great importance when dimensioning a network or evaluating its performance [34]. That was one of the main motivations behind the work in [42] which evaluated how the controller and the network perform with bandwidth and latency issues on the control link. This work concludes that bandwidth in the control link arbitrates how many flows can be processed by the controller, as well as the loss rate when under saturation conditions. The switch-to-control latency on the other hand, has a major impact on the overall behavior of the network, as each switch cannot forward data until it receives the message from the controller that inserts the appropriate rules in the flow table. This interval can grow with the link latency and impact dramatically the performance of network applications.

Also, control modeling greatly impacts the network scalability. Some important scalability issues are presented in [43], along with a discussion about scalability trade-offs in software-defined network design.

2) Control models: In the following, we go over some of these SDN design options and discuss different methods of controlling a software-defined network, many of which are interrelated:

- **Centralized vs. Distributed**
  Although protocols such as OpenFlow specify that a switch is controlled by a controller and therefore appears to imply centralization, software-defined networks may have either a centralized or distributed control-

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

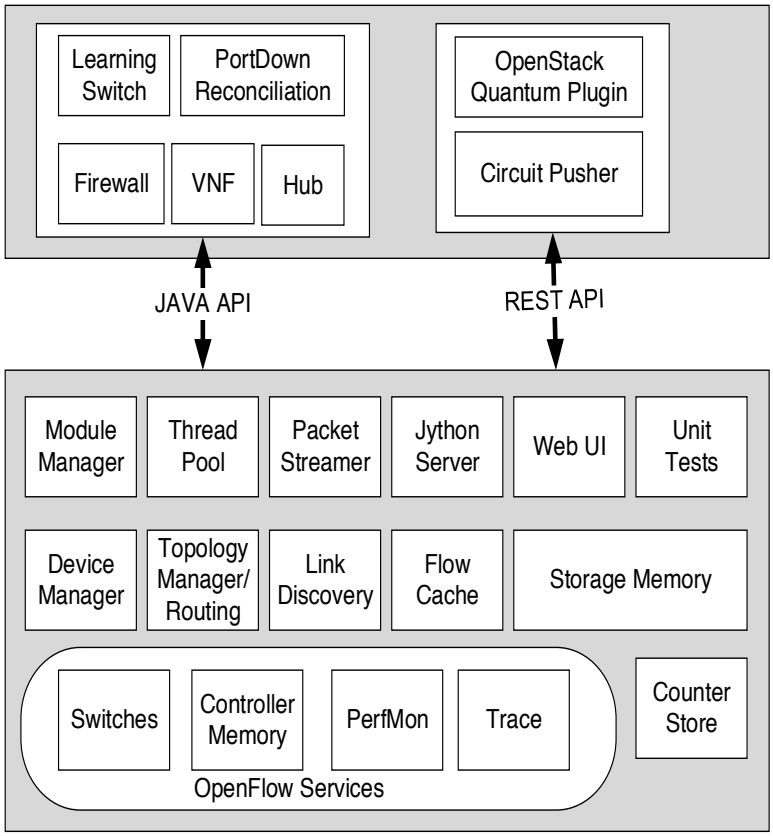

Fig. 4. The Floodlight architecture as an example of an OpenFlow controller.

plane. Though controller-to-controller communication is not defined by OpenFlow, it is necessary for any type of distribution or redundancy in the control-plane.

A physically centralized controller represents a single point of failure for the entire network; therefore, OpenFlow allows the connection of multiple controllers to a switch, which would allow backup controllers to take over in the event of a failure.

Onix [44] and HyperFlow [45] take the idea further by attempting to maintain a logically centralized but physically distributed control plane. This decreases the look-up overhead by enabling communication with local controllers, while still allowing applications to be written with a simplified central view of the network. The potential downside are trade-offs [46] related to consistency and staleness when distributing state throughout the control plane, which has the potential to cause applications that believe they have an accurate view of the network to act incorrectly.

A hybrid approach, such as Kandoo [47], can utilize local controllers for local applications and redirect to a global controller for decisions that require centralized network state. This reduces the load on the global controller by filtering the number of new flow requests, while also providing the data-path with faster responses for requests that can be handled by a local control application.

A software-defined network can also have some level of logical decentralization, with multiple logical controllers. An interesting type of proxy controller, called Flowvisor [48], can be used to add a level of network virtualization to OpenFlow networks and allow multiple controllers to simultaneously control overlapping sets of physical switches. Initially developed to allow experimental research to be conducted on deployed networks alongside production traffic, it also facilitates and demonstrates the ease of deploying new services in SDN environments.

A logically decentralized control plane would be needed in an inter-network spanning multiple administrative domains. Though the domains may not agree to centralized control, a certain level of sharing may be appropriate (e.g., to ensure service level agreements are met for traffic flowing between domains).

- **Control Granularity**

  Traditionally, the basic unit of networking has been the packet. Each packet contains address information necessary for a network switch to make routing decisions. However, most applications send data as a flow of many individual packets. A network that wishes to provide QoS or service guarantees to certain applications may benefit from individual flow-based control. Control can be further abstracted to an aggregated flow-match, rather than individual flows. Flow aggregation may be based on source, destination, application, or any combination thereof.

  In a software-defined network where network elements are controlled remotely, overhead is caused by traffic between the data-plane and control-plane. As such, using packet level granularity would incur additional delay as

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

the controller would have to make a decision for each arriving packet. When controlling individual flows, the decision made for the first packet of the flow can be applied to all subsequent packets of that flow. The overhead may be further reduced by grouping flows together, such as all traffic between two hosts, and performing control decisions on the aggregated flows.

- **Reactive vs. Proactive Policies**
  Under a *reactive* control model, such as the one proposed by Ethane [20], forwarding elements must consult a controller each time a decision must be made, such as when a packet from a new flow reaches a switch. In the case of flow-based control granularity, there will be a small performance delay as the first packet of each new flow is forwarded to the controller for decision (e.g., forward or drop), after which future packets within that flow will travel at line rate within the forwarding hardware. While the delay incurred by the first-packet may be negligible in many cases, it may be a concern if the controller is geographically remote (though this can be mitigated by physically distributing the controller [45]) or if most flows are short-lived, such as single-packet flows. There are also some scalability issues in larger networks, as the controller must be able to handle a larger volume of new flow requests.
  Alternatively, *proactive* control approaches push policy rules from the controller to the switches. A good example of proactive control is DIFANE [35], which partitions rules over a hierarchy of switches, such that the controller rarely needs to be consulted about new flows and traffic is kept within the data-plane. In their experiments, DIFANE reduces first-packet delay from a 10ms average round-trip time (RTT) with a centralized NOX controller to a 0.4ms average RTT for new single-packet flows. It was also shown to increase the new flow throughput, as the tested version of NOX achieved a peak of 50,000 single-packet flows per second while the DIFANE solution achieved 800,000 single-packet flows per second. Interestingly, it was observed that the OpenFlow switch's local controller implementation becomes a bottleneck before the central NOX controller. This was attributed to the fact that commercial OpenFlow switch implementations were limited to sending 60-330 new flows requests per second at the time of their publication (2010).

As shown in Figure 5, a controller that acts as a network operating system must implement at least two interfaces: a "southbound" interface that allows switches to communicate with the controller and a "northbound" interface that presents an API to network control and high-level applications/services.

D. Southbound Communication: Controller-Switch

An important aspect of SDNs is the link between the data-plane and the control-plane. As forwarding elements are controlled by an open interface, it is important that this link remains available and secure.

The OpenFlow protocol can be viewed as one possible implementation of controller-switch interactions, as it defines the communication between the switching hardware and a network

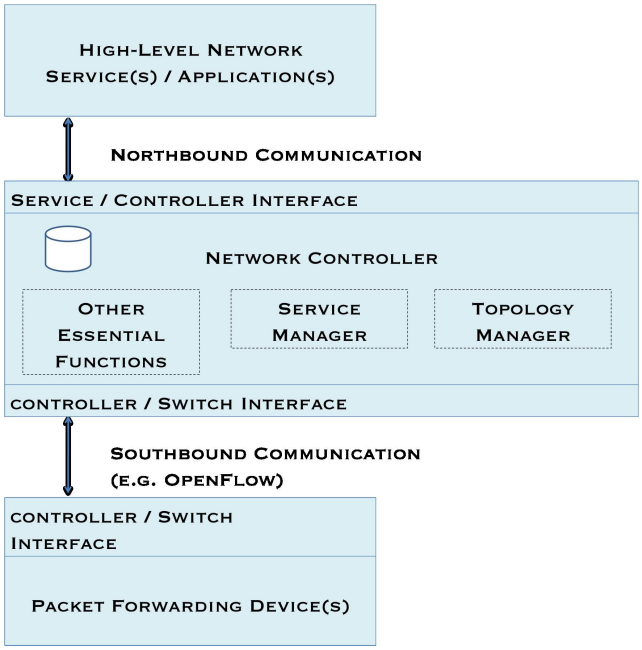

Fig. 5. A controller with a northbound and southbound interface.

controller. For security, OpenFlow 1.3.0 provides optional support for encrypted Transport Layer Security (TLS) communication and a certificate exchange between the switches and the controller(s); however, the exact implementation and certificate format is not currently specified. Also outside the scope of the current specification are fine-grained security options regarding scenarios with multiple controllers, as there is no method specified to only grant partial access permissions to an authorized controller. We examine OpenFlow controller implementation options in greater detail in Section IV.

E. Northbound Communication: Controller-Service

External management systems or network services may wish to extract information about the underlying network or control an aspect of network behavior or policy. Additionally, controllers may find it necessary to communicate with each other for a variety of reasons. For example, an internal control application may need to reserve resources across multiple domains of control or a "primary" controller may need to share policy information with a backup, etc.

Unlike controller-switch communication, there is no currently accepted standard for northbound interactions and they are more likely to be implemented on an ad hoc basis for particular applications. We discuss this further in Section VI.

F. Standardization Efforts

Recently, several standardization organizations have been turning the spotlights towards SDN. For example, as previously mentioned, the IETF's Forwarding and Control Element Separation (ForCES) Working Group [1] has been working on standardizing mechanisms, interfaces, and protocols aiming at the centralization of network control and abstraction of network infrastructure. The Open Network Foundation (ONF) [3] has been trying to standardize the OpenFlow

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE I
CURRENT SOFTWARE SWITCH IMPLEMENTATIONS COMPLIANT WITH THE OPENFLOW STANDARD.

Software Switch	Implementation	Overview	Version
Open vSwitch [55]	C/Python	Open source software switch that aims to implement a switch platform in virtualized server environments. Supports standard management interfaces and enables programmatic extension and control of the forwarding functions. Can be ported into ASIC switches.	v1.0
Pantou/OpenWRT [56]	C	Turns a commercial wireless router or Access Point into an OpenFlow-enabled switch.	v1.0
ofsoftswitch13 [57]	C/C++	OpenFlow 1.3 compatible user-space software switch implementation.	v1.3
Indigo [58]	C	Open source OpenFlow implementation that runs on physical switches and uses the hardware features of Ethernet switch ASICs to run OpenFlow.	v1.0

protocol. As the control plane abstracts network applications from underlying hardware infrastructure, they focus on standardizing the interfaces between: (1) network applications and the controller (i.e. northbound interface) and (2) the controller and the switching infrastructure (i.e., southbound interface) which defines the OpenFlow protocol itself. Some of the Study Groups (SGs) of ITU's Telecommunication Standardization Sector (ITU-T) [49] are currently working towards discussing requirements and creating recommendations for SDNs under different perspectives. For instance, the SG13 focuses on Future Networks, including cloud computing, mobile and next generation networks, and is establishing requirements for network virtualization. Other ITU-T SGs such as the SG11 for protocols and test specifications started, in early 2013, requirements and architecture discussions on SDN signaling. The Software-Defined Networking Research Group (SDNRG) at IRTF [50] is also focusing on SDN under various perspectives with the goal of identifying new approaches that can be defined and deployed, as well as identifying future research challenges. Some of their main areas of interest include solution scalability, abstractions, security and programming languages and paradigms particularly useful in the context of SDN.

These and other working groups perform important work, coordinating efforts to evolve existing standards and proposing new ones. The goal is to facilitate smooth transitions from legacy networking technology to the new protocols and architectures, such as SDN Some of these groups, such as ITU-T's SG13, advocate the establishment of a Joint Coordination Activity on SDN (JCA-SDN) for collaboration and coordination between standardizing efforts and also taking advantage of the work performed by the Open Source Software (OSS) community, such as OpenStack [51] and OpenDayLight [52] as they start developing the building blocks for SDN implementation.

IV. SDN DEVELOPMENT TOOLS

SDN has been proposed to facilitate network evolution and innovation by allowing rapid deployment of new services and protocols. In this section, we provide an overview of currently available tools and environments for developing SDN-based services and protocols.

A. Emulation and Simulation Tools

Mininet [53] allows an entire OpenFlow network to be emulated on a single machine, simplifying the initial development and deployment process. New services, applications and protocols can first be developed and tested on an emulation of the anticipated deployment environment before moving to the actual hardware. By default Mininet supports OpenFlow v1.0, though it may be modified to support a software switch that implements a newer release.

The `ns-3` [54] network simulator supports OpenFlow switches within its environment, though the current version only implements OpenFlow v0.89.

TABLE II
MAIN CURRENT AVAILABLE COMMODITY SWITCHES BY MAKERS, COMPLIANT WITH THE OPENFLOW STANDARD.

Maker	Switch Model	Version
Hewlett-Packard	8200zl, 6600, 6200zl, 5400zl, and 3500/3500yl	v1.0
Brocade	NetIron CES 2000 Series	v1.0
IBM	RackSwitch G8264	v1.0
NEC	PF5240 PF5820	v1.0
Pronto	3290 and 3780	v1.0
Juniper	Junos MX-Series	v1.0
Pica8	P-3290, P-3295, P-3780 and P-3920	v1.2

B. Available Software Switch Platforms

There are currently several SDN software switches available that can be used, for example, to run an SDN testbed or when developing services over SDN. Table I presents a list of current software switch implementations with a brief description including implementation language and the OpenFlow standard version that the current implementation supports.

C. Native SDN Switches

One of the main SDN enabling technologies currently being implemented in commodity networking hardware is the OpenFlow standard. In this section we do not intend to present a detailed overview of OpenFlow enabled hardware and makers, but rather provide a list of native SDN switches currently available in the market and provide some information about them, including the version of OpenFlow they implement.

One clear evidence of industry's strong commitment to SDN is the availability of commodity network hardware that are OpenFlow enabled. Table II lists commercial switches that are currently available, their manufacturer, and the version of OpenFlow they implement.

D. Available Controller Platforms

Table III shows a snapshot of current controller implementations. To date, all the controllers in the table support the

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

OpenFlow protocol version 1.0, unless stated otherwise. This table also provides a brief overview of the listed controllers.

Included in Table III are also two special purpose controller implementations: Flowvisor [48], mentioned previously, and RouteFlow [66]. The former acts as a transparent proxy between OpenFlow switches and multiple OpenFlow controllers. It is able to create network slices and can delegate control of each slice to a different controller, also promoting isolation between slices. RouteFlow, on the other hand, is an open source project to provide virtualized IP routing over OpenFlow capable hardware. It is composed of an OpenFlow Controller application, an independent server, and a virtual network environment that reproduces the connectivity of a physical infrastructure and runs IP routing engines. The routing engines generate the forwarding information base (FIB) into the Linux IP tables according to the routing protocols configured (e.g., OSPF, BGP). An extension of RouteFlow is presented in [67], which discusses Routing Control Platforms (RCPs) in the context of OpenFlow/SDN. They proposed a controller-centric networking model along with a prototype implementation of an autonomous-system-wide abstract BGP routing service.

E. Code Verification and Debugging

Verification and debugging tools are vital resources for traditional software development and are no less important for SDN. Indeed, for the idea of portable network "apps" to be successful, network behavior must be thoroughly tested and verified.

NICE [68] is an automated testing tool used to help uncover bugs in OpenFlow programs through model checking and symbolic execution.

Anteater [69] takes a different approach by attempting to check network invariants that exist in the data plane, such as connectivity or consistency. The main benefit of this approach is that it is protocol-agnostic; it will also catch errors that result from faulty switch firmware or inconsistencies with the control plane communication. VeriFlow [70] has a similar goal, but goes further by proposing a real-time verification tool that resides between the controller and the forwarding elements. This adds the potential benefit of being able to halt bad rules that will cause anomalous behavior before they reach the network.

Other efforts proposed debugging tools that provide insights gleaned from control plane traffic. OFRewind [71] allows network events (control and data) to be recorded at different granularities and later replayed to reproduce a specific scenario, granting the opportunity to localize and troubleshoot the events that caused the network anomaly. *ndb* [72] implements breakpoints and packet-backtraces for SDN. Just as with the popular software debugger *gdb*, users can pinpoint events that lead to error by pausing execution at a breakpoint, or, using a packet backtrace, show the sequence of forwarding actions seen by that packet. STS [73] is a software-defined network troubleshooting simulator. It is written in python and depends on POX. It simulates the devices in a given network allowing for testing cases and identifying the set of inputs that generates a given error.

V. SDN Applications

Software-defined networking has applications in a wide variety of networked environments. By decoupling the control– and data planes, programmable networks enable customized control, an opportunity to eliminate middleboxes, as well as simplified development and deployment of new network services and protocols. Below, we examine different environments for which SDN solutions have been proposed or implemented.

A. Enterprise Networks

Enterprises often run large networks, while also having strict security and performance requirements. Furthermore, different enterprise environments can have very different requirements, characteristics, and user population, For example, University networks can be considered a special case of enterprise networks: in such an environment, many of the connecting devices are temporary and not controlled by the University, further challenging security and resource allocation. Additionally, Universities must often provide support for research testbeds and experimental protocols.

Adequate management is critically important in Enterprise environments, and SDN can be used to programmatically enforce and adjust network policies as well as help monitor network activity and tune network performance.

Additionally, SDN can be used to simplify the network by ridding it from middleboxes and integrating their functionality within the network controller. Some notable examples of middlebox functionality that has been implemented using SDN include NAT, firewalls, load balancers [74] [75], and network access control [76]. In the case of more complex middleboxes with functionalities that cannot be directly implemented without performance degradation (e.g., deep packet inspection), SDN can be used to provide unified control and management[77].

The work presented in [78] addresses the issues related to consistent network updates. Configuration changes are a common source of instability in networks and can lead to outages, security flaws, and performance disruptions. In [78], a set of high-level abstractions are proposed that allow network administrators to update the entire network, guaranteeing that every packet traversing the network is processed by exactly one consistent global network configuration. To support these abstractions, several OpenFlow-based update mechanisms were developed.

As discussed in earlier sections, OpenFlow evolved from Ethane [20], a network architecture designed specifically to address the issues faced by enterprise networks.

B. Data Centers

Data centers have evolved at an amazing pace in recent years, constantly attempting to meet increasingly higher and rapidly changing demand. Careful traffic management and policy enforcement is critical when operating at such large scales, especially when any service disruption or additional delay may lead to massive productivity and/or profit loss. Due to the challenges of engineering networks of this scale and

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE III
CURRENT CONTROLLER IMPLEMENTATIONS COMPLIANT WITH THE OPENFLOW STANDARD.

Controller	Implementation	Open Source	Developer	Overview
POX [59]	Python	Yes	Nicira	General, open-source SDN controller written in Python.
NOX [17]	Python/C++	Yes	Nicira	The first OpenFlow controller written in Python and C++.
MUL [60]	C	Yes	Kulcloud	OpenFlow controller that has a C-based multi-threaded infrastructure at its core. It supports a multi-level north-bound interface (see Section III-E) for application development.
Maestro [21]	Java	Yes	Rice University	A network operating system based on Java; it provides interfaces for implementing modular network control applications and for them to access and modify network state.
Trema [61]	Ruby/C	Yes	NEC	A framework for developing OpenFlow controllers written in Ruby and C.
Beacon [22]	Java	Yes	Stanford	A cross-platform, modular, Java-based OpenFlow controller that supports event-based and threaded operations.
Jaxon [62]	Java	Yes	Independent Developers	a Java-based OpenFlow controller based on NOX.
Helios [24]	C	No	NEC	An extensible C-based OpenFlow controller that provides a programmatic shell for performing integrated experiments.
Floodlight [38]	Java	Yes	BigSwitch	A Java-based OpenFlow controller (supports v1.3), based on the Beacon implementation, that works with physical- and virtual- OpenFlow switches.
SNAC [23]	C++	No	Nicira	An OpenFlow controller based on NOX-0.4, which uses a web-based, user-friendly policy manager to manage the network, configure devices, and monitor events.
Ryu [63]	Python	Yes	NTT, OSRG group	An SDN operating system that aims to provide logically centralized control and APIs to create new network management and control applications. Ryu fully supports OpenFlow v1.0, v1.2, v1.3, and the Nicira Extensions.
NodeFlow [64]	JavaScript	Yes	Independent Developers	An OpenFlow controller written in JavaScript for Node.JS [65].
ovs-controller [55]	C	Yes	Independent Developers	A simple OpenFlow controller reference implementation with Open vSwitch for managing any number of remote switches through the OpenFlow protocol; as a result the switches function as L2 MAC-learning switches or hubs.
Flowvisor [48]	C	Yes	Stanford/Nicira	Special purpose controller implementation.
RouteFlow [66]	C++	Yes	CPqD	Special purpose controller implementation.

complexity to dynamically adapt to application requirements, it is often the case that data centers are provisioned for peak demand; as a result, they run well below capacity most of the time but are ready to rapidly service higher workloads.

An increasingly important consideration is energy consumption, which has a non-trivial cost in large-scale data centers. Heller et al. [79] indicates that much research has been focused on improved servers and cooling (70% of total energy) through better hardware or software management, but the data center's network infrastructure (which accounts for 10-20% of the total energy cost) still consumed 3 billion kWh in 2006. They proposed ElasticTree, a network-wide power manager that utilizes SDN to find the minimum-power network subset which satisfies current traffic conditions and *turns off* switches that are not needed. As a result, they show energy savings between 25-62% under varying traffic conditions. One can imagine that these savings can be further increased if used in parallel with server management and virtualization; one possibility is the Honeyguide[80] approach to energy optimization which uses virtual machine migration to increase the number of machines and switches that can be shutdown.

However, not all SDN solutions may be appropriate in high performance networks. While simplified traffic management and visibility are useful, it must be sensibly balanced with scalability and performance overhead. Curtis et al. [34] believe that OpenFlow excessively couples central control and complete visibility, when in reality only "significant" flows need to be managed; this may lead to bottlenecks as the control-data communication adds delay to flow setup while switches are overloaded with thousands of flow table entries. Though aggressive use of proactive policies and wild-card rules may resolve that issue, it may undermine the ability of the controller to have the right granularity to effectively manage traffic and gather statistics. Their framework, DevoFlow, proposes some modest design changes to keep flows in the data plane as much as possible while maintaining enough visibility for effective flow management. This is accomplished by pushing responsibility over most flows back to the switches and adding more efficient statistics collection mechanisms, through which "significant" flows (e.g. long-lived, high-throughput) are identified and managed by the controller. In a load-balancing simulation, their solution had 10-53 times fewer flow table entries and 10-42 times fewer control messages on average over OpenFlow.

A practical example of a real application of the SDN concept and architecture in the context of data centers was presented by Google in early 2012. The company presented at the Open Network Summit [81] a large scale implementation of an SDN-based network connecting its data centers. The work in [82] presents in more detail the design, implementation, and evaluation of B4, a WAN connecting Google's data-centers world wide. This work describes one of the first and largest SDN deployments. The motivation was the need for customized routing and traffic engineering and the fact that the level of scalability, fault tolerance, cost efficiency and control required, could not be achieved by means of a traditional WAN architecture. A customized solution was proposed and an OpenFlow-based SDN architecture was built to control individual switches. After three years in production, B4 is shown to be efficient in the sense that it drives many links at near 100% utilization while splitting flows among multiple paths. Furthermore, the experience reported in the work shows that the bottleneck resulting from control-plane to data-plane communication and overhead in hardware programming are important issues to be considered in future work.

C. Infrastructure-based Wireless Access Networks

Several efforts have focused on ubiquitous connectivity in the context of infrastructure-based wireless access networks, such as cellular and WiFi.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

For example, the OpenRoads project [83], [84] envisions a world in which users could freely and seamlessly move across different wireless infrastructures which may be managed by various providers. They proposed the deployment of an SDN-based wireless architecture that is backwards-compatible, yet open and sharable between different service providers. They employ a testbed using OpenFlow-enabled wireless devices such as WiFi APs and WiMAX base stations controlled by NOX– and Flowvisor controllers and show improved performance on handover events. Their vision provided inspiration for subsequent work [85] that attempts to address specific requirements and challenges in deploying a software-defined cellular network.

Odin[86] introduces programmability in enterprise wireless LAN environments. In particular, it builds an access point abstraction on the controller that separates the association state from the physical access point, enabling proactive mobility management and load balancing without changes to the client.

At the other end of the spectrum, OpenRadio [87] focuses on deploying a programmable wireless data plane that provides flexibility at the PHY and MAC layers (as opposed to layer-3 SDN) while meeting strict performance and time deadlines. The system is designed to provide a modular interface that is able to process traffic subsets using different protocols such as WiFi, WiMAX, 3GPP LTE-Advanced, etc. Based on the idea of separation of the decision and forwarding planes, an operator may express decision plane rules and corresponding actions, which are assembled from processing plane modules (e.g., FFT, Viterbi decoding, etc); the end result is a state machine that expresses a fully-functional protocol.

D. Optical Networks

Handling data traffic as flows, allows software-defined networks, and OpenFlow networks in particular, to support and integrate multiple network technologies. As a result, it is possible to provide also technology-agnostic unified control for optical transport networks and facilitating interaction between both packet and circuit-switched networks. According to the Optical Transport Working Group (OTWG) created in 2013 by the Open Network Foundation (ONF), the benefits from applying SDN and the OpenFlow standard in particular to optical transport networks include: improving optical transport network control and management flexibility, enabling deployment of third-party management and control systems, and deploying new services by leveraging virtualization and SDN [88].

There has been several attempts and proposals to control both circuit switched and packet switched networks using the OpenFlow protocol. In [89] a NetFPGA [90] platform is used in the proposal of a packet switching and circuit switched networks architectures based on Wavelength Selective Switching (WSS), using the OpenFlow protocol. Another control plane architecture based on OpenFlow for enabling SDN operations in optical networks was proposed in [91], which discusses specific requirements and describes implementation of OpenFlow protocol extensions to support optical transport networks.

A proof-of-concept demonstration of an OpenFlow-based wavelength path control in transparent optical networks is presented in [92]. In this work, virtual Ethernet interfaces (*veths*) are introduced. These veths, are mapped to physical interfaces of an optical node (e.g. photonic cross-connect - PXC), and enable an SDN controller (e.g. the NOX controller in this case) to operate the optical lightpaths (e.g., via the OpenFlow protocol). In their experimental setup, they quantitatively evaluate network performance metrics, such as the latency of lightpath setup and release, and verify the feasibility of routing and wavelength assignment, and the dynamic control of optical nodes in an OpenFlow-based network composed by four PXCs nodes in a mesh topology.

A Software Defined Optical Network (SDON) architecture is introduced in [93] and a QoS-aware unified control protocol for optical burst switching in OpenFlow-based SDON is developed. The performance of the proposed protocol was evaluated with the conventional GMPLS-based distributed protocol and the results indicate that SDON offers an infrastructure to support unified control protocols to better optimize network performance and improve capacity.

E. Home and Small Business

Several projects have examined how SDN could be used in smaller networks, such as those found in the home or small businesses. As these environments have become increasingly complex and prevalent with the widespread availability of low-cost network devices, the need for more careful network management and tighter security has correspondingly increased. Poorly secured networks may become unwitting targets or hosts for malware, while outages due to network configuration issues may cause frustration or lost business. Unfortunately, it is not practical to have a dedicated network administrator in every home and office.

Calvert et al. [94] assert that the first step in managing home networks is to know what is actually happening; as such, they proposed instrumenting the network gateway/controller to act as a "Home Network Data Recorder" to create logs that may be utilized for troubleshooting or other purposes.

Feamster [95] proposes that such networks should operate in a "plug in and forget" fashion, namely by outsourcing management to third-party experts, and that this could be accomplished successfully through the remote control of programmable switches and the application of distributed network monitoring and inference algorithms used to detect possible security problems.

In contrast, Mortier et al. [96] believe that users desire greater understanding and control over their networks' behavior; rather than following traditional policies, a home network may be better managed by their users who better understand the dynamics and needs of their environment. Towards this goal, they created a prototype network in which SDN is used to provide users a view into how their network is being utilized while offering a single point of control.

Mehdi et al. [97] argues that an Anomaly Detection System (ADS) implemented within a programmable home network provides a more accurate identification of malicious activity as compared to one deployed at the ISP; additionally, the implementation would be able to operate at line rate with no performance penalty, while, at the same time, offloading the

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

ISP from having to monitor these large number of networks. The ADS algorithm could operate alongside other controller services, such as a HomeOS that may react to suspicious activity and report anomalies to the ISP or local administrator.

VI. Research Challenges and Future Directions

As SDN becomes more widely adopted and protocols such as OpenFlow are further defined, new solutions are proposed and new challenges arise. In this section we discuss various challenges posed by SDN as well as future research directions, namely: (1) controller and switch design, (2) scalability and performance in SDNs, (3) controller-service interfacing, (4) virtualization and cloud service applications, (5) information centric networking, and (6) enabling heterogeneous networking with SDN.

A. Controller and Switch Design

SDN raises significant scalability, performance, robustness, and security challenges. Below we review a number of research efforts focusing on addressing these issues at the switch– and controller design level.

In DIFANE [35], flow entries are proactively pushed to switches in an attempt to reduce the number of requests to the controller. Devoflow [34] proposes to handle "short-lived" flows in switches and "long-lived" flows in the controller to mitigate flow setup delay and controller overhead. The work proposed in [28] advocates replacing counters on ASIC by a stream of rule-matching records and processing them in the CPU to allow efficient access to counters. FLARE [98] is a new network node model focusing on "deeply programmable networks" that provides programmability for the data plane, the control plane, as well as the interface between them. The work presented in [99] discusses important aspects in controller design including hierarchical control, data model, scalability, and extensibility.

As far as performance and scalability, the study presented in [100] showed that one single controller can handle up to 6 million flows per second. A more recent study [101], focusing on the Beacon controller, showed that a controller can handle 12.8 million new flows per second in a 12 cores machine, with an average latency of 24.7 us for each flow. However, for increased scalability and especially for reliability and robustness purposes, it has been recognized that the logically-centralized controller must be physically distributed. Onix [44], Kando [47], and HyperFlow [45] use this approach to achieve robust and scalable control plane. In [46], trade-offs related to control distribution, such as staleness versus optimality and application logic complexity versus robustness to inconsistency are identified and quantified. In [41], the controller placement problem is discussed in terms of the number of controllers needed and where to place them in the network. In more recent work on distributed control, the need for dynamic assignment of switches to controllers is addressed in [102], which proposes an algorithm to increase or decrease the pool of controllers based on controllers' load estimates. They also propose a mechanism to dynamically handover switches from one controller to another as needed.

In [103] an SDN variant inspired by MPLS was proposed along with the notions of *edge controllers* and *fabric controllers*: the former control ingress and egress switches and handle the host-network interface, while the latter handle fabric switches and the operator-network interface.

Although control and measurement are two important components of network management, little thought has gone into designing APIs for measurement. The work presented in [104] proposes a software-defined traffic measurement architecture, which separates the measurement data plane from the control plane.

B. Software-Defined Internetworking

The Internet has revolutionized the way we, as individuals and as a society, live, work, conduct business, socialize, get entertainment, etc. As a result, the Internet is now considered part of our society's critical infrastructure much like the power, water, and transportation grids.

Scalability and performance requirements from increasingly complex applications have posed a variety of challenges difficult to address with the current Internet architecture. This has led the research community to examine "clean-slate" solutions [105]. As the Internet has grown beyond the point at which a "flag day", such as the one used to "upgrade" the ARPANET with the TCP/IP protocol suite, would be realistic, another considerable challenge is evolving its physical infrastructure and protocols. A notable example is the deployment of IPv6: despite over a decade in the standards track and two worldwide deployment events, IPv4 still makes up the majority of Internet traffic.

Much of the current work on SDN examines or proposes solutions within the context of a single administrative domain which matches quite well SDN's logically centralized control model. However, environments whose administration is inherently decentralized, like the Internet, call for a control plane that is logically distributed. This will allow participating autonomous systems (ASes) to be controlled independently by their own (logically centralized and possibly physically distributed) controller. To-date, a few efforts have explored the idea of a Software-Defined Internet. For example, the work in [106] proposed a software-defined Internet architecture that borrows from MPLS the distinction between network edge and core to split tasks between inter-domain and intra-domain components. As only the boundary routers and their associated controller in each domain are involved in inter-domain tasks, changes to inter-domain service models would be limited to software modifications at the inter-domain controllers rather than the entire infrastructure. Examples of how this architecture could be used to realize new Internet services such as information-centric networking, and middlebox service sharing are explored.

Another approach to inter-AS routing [107] uses NOX and OpenFlow to implement BGP-like functionality. Alternatively, an extensible session protocol [108] supports application-driven configuration of network resources across domains.

C. Controller-Service Interaction

While controller-switch ("southbound") interaction is fairly well defined in protocols such as OpenFlow and ForCES,

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

there is no standard for interactions between controllers and network services or applications ("northbound"). One possible explanation is that the northbound interface is defined entirely in software, while controller-switch interactions must enable hardware implementation.

If we think of the controller as a "network operating system", then there should be a clearly defined interface by which applications can access the underlying hardware (switches), co-exist and interact with other applications, and utilize system services (e.g. topology discovery, forwarding), without requiring the application developer to know the implementation details of the controller. While there are several controllers that exist, their application interfaces are still in the early stages and independent from each other.

Some proposals (e.g., Procera [109], Frenetic [110], FML [111], Nettle [112]) advocate the use of a network configuration language to express policies. For example, Procera [109] builds a policy layer on top of existing controllers to interface with configuration files, GUIs, and external sensors; the proposed policy layer is responsible for converting high-level policies to flow constraints given to be used by the controller. In [113], network configuration and management mechanisms are proposed that focus on enabling changes to network condition and state, supporting network configuration and policy definitions, and providing visibility and control over tasks for network diagnostics and troubleshooting. The specification of a northbound interface via a policy layer and a high level language such as Procera is discussed.

Additionally, the northbound API should allow applications to apply different policies to the same flow (e.g. forwarding by destination and monitoring by source IP). The work in [114] proposed modularization to ensure that rules installed to perform one task do not override other rules. This was accomplished by means of an abstraction layer implemented with a language based on Frenetic.

Until a clear northbound interface standard emerges, SDN applications will continue to be developed in an "ad hoc" fashion and the concept of flexible and portable "network apps" may have to wait.

D. Virtualization and Cloud Services

The demand for virtualization and cloud services has been growing rapidly and attracting considerable interest from industry and academia. The challenges it presents include rapid provisioning, efficient resource management, and scalability which can be addressed using SDN's control model.

For example, FlowVisor [48] and AutoSlice [115] create different slices of network resources (e.g., bandwidth, topology, CPU, forwarding table), delegate them to different controllers, and enforce isolation between slices. Other SDN controllers can be used as a network backend to support virtualization in cloud operating systems, such as Floodlight for OpenStack [38] and NOX for Mirage [116]. FlowN [117] aims to offer a scalable solution for network virtualization by providing an efficient mapping between virtual and physical networks and by leveraging scalable database systems.

In [118], an algorithm for efficient migration with bandwidth guarantees using OpenFlow was proposed. LIME [119] is an SDN-based solution for live migration of Virtual Machines, which handles the network state during migration and automatically configures network devices at new locations. NetGraph [120] provides a set of APIs for customers to access its virtual network functions such as real-time monitoring and diagnostics.

On the context of cloud data centers providing Infrastructure as a Service (IaaS), [121] presents a management framework for resources in cloud data centers and addresses multiple management issues. In this paper, authors proposed a data-centric and event-driven architecture with open management interfaces, that leverages SDN techniques to integrate network resources into datacenter orchestration and service provisioning with the aim of improving service-level agreements and faster service delivery.

E. Information-Centric Networking

Information-Centric Networking (ICN) is a new paradigm proposed for the future architecture of the Internet, which aims to increase the efficiency of content delivery and content availability. This new concept has been popularized recently by a number of architecture proposals, such as Content-Centric Networking (CCN), also known as the Named Data Networking (NDN) project [122]. Their driving motivation is that the current Internet is information-driven, yet networking technology is still focused on the idea of location-based addressing and host-to-host communication.By proposing an architecture that addresses *named data* rather than *named hosts*, content distribution is implemented directly into the network fabric rather than relying on the complicated mapping, availability, and security mechanisms currently used to map content to a single location.

The separation between information processing and forwarding in ICN is aligned with the decoupling of the data plane and control plane in SDN. The question then becomes how to combine ICN with SDN towards "Software-Defined Information-Centric Networks". A number of projects [123], [124], [125], [126], [127], [128] have proposed using SDN concepts to implement ICNs. As OpenFlow expands to support customized header matchings, SDN can be employed as a key enabling technology for ICNs.

F. Heterogeneous Network Support

Future networks will become increasingly more heterogeneous, interconnecting users and applications over networks ranging from wired, infrastructure-based wireless (e.g., cellular–based networks, wireless mesh networks), to infrastructure-less wireless networks (e.g. mobile ad-hoc networks, vehicular networks). In the meantime, mobile traffic has been increasing exponentially over the past several years, and is expected to increase 18–fold by 2016, with more mobile-connected devices than the world's population, which is already a reality [129]. As mobile devices with multiple network interfaces become commonplace, users will demand high quality communication service regardless of location or type of network access. Self-organizing networks (e.g., wireless multi-hop ad-hoc networks) may form to extend the range

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

of infrastructure-based networks or handle episodic connectivity disruptions. Self-organizing networks may thus enable a variety of new applications such as cloud-based services, vehicular communication, community services, healthcare delivery, emergency response, and environmental monitoring, to name a few. Efficient content delivery over wireless access networks will become essential, and self–organizing networks may become a prevalent part of the future hybrid Internet.

A major challenge facing future networks is efficient utilization of resources; this is especially the case in wireless multi-hop ad-hoc networks as the available wireless capacity is inherently limited. This is due to a number of factors including the use of shared physical medium compounded, wireless channel impairments, and the absence of managed infrastructure. Though these self–organizing networks can be used to supplement or "fill the gaps" in an overburdened infrastructure [130], their lack of dedicated resources and shifting connectivity makes capacity sharing difficult. The heterogeneous characteristics of the underlying networks (e.g., physical medium, topology, stability) and nodes (e.g., buffer size, power limitations, mobility) also add another important factor when considering routing and resource allocation.

SDN has the potential to facilitate the deployment and management of network applications and services with greater efficiency. However, SDN techniques to–date, such as OpenFlow, largely target infrastructure–based networks. They promote a centralized control mechanism that is ill–suited to the level of decentralization, disruption, and delay present in infrastructure-less environments.

While previous work has examined the use of SDN in wireless environments, the scope has primarily focused on infrastructure-based deployments (e.g., WiMAX, Wi-Fi access points). A notable example is the OpenRoads project [83], which envisioned a world in which users could freely move between wireless infrastructures while also providing support to the network provider. Other studies such as [128], [131], [132] have examined OpenFlow in wireless mesh environments.

VII. Concluding Remarks

In this paper, we provided an overview of *programmable networks* and, in this context, examined the emerging field of Software-Defined Networking (SDN). We look at the history of programmable networks, from early ideas until recent developments. In particular we described the SDN architecture in detail as well as the OpenFlow [2] standard. We presented current SDN implementations and testing platforms and examined network services and applications that have been developed based on the SDN paradigm. We concluded with a discussion of future directions enabled by SDN ranging from support for heterogeneous networks to Information Centric Networking (ICN).

Acknowledgments

The authors would like to thank the reviewers for their careful examination of the manuscript and valuable comments which helped to considerably improve the quality of the paper. This work is partly funded by the Community Associated Team between INRIA and UCSC and the French ANR under the "ANR-13-INFR-013" project, and by NSF grant CNS 1150704.

References

[1] A. Doria, J. Hadi Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, R. Gopal, and J. Halpern. Forwarding and Control Element Separation (ForCES) Protocol Specification. RFC 5810 (Proposed Standard), March 2010.

[2] 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 Commun. Review*, 38(2):69–74, 2008.

[3] Open networking foundation. https://www.opennetworking.org/about.

[4] Open Networking Research Center (ONRC). http://onrc.net.

[5] Thomas A. Limoncelli. Openflow: a radical new idea in networking. *Commun. ACM*, 55(8):42–47, August 2012.

[6] A.T. Campbell, I. Katzela, K. Miki, and J. Vicente. Open signaling for atm, internet and mobile networks (opensig'98). *ACM SIGCOMM Computer Commun. Review*, 29(1):97–108, 1999.

[7] A. Doria, F. Hellstrand, K. Sundell, and T. Worster. General Switch Management Protocol (GSMP) V3. RFC 3292 (Proposed Standard), June 2002.

[8] D.L. Tennenhouse, J.M. Smith, W.D. Sincoskie, D.J. Wetherall, and G.J. Minden. A survey of active network research. *IEEE Commun. Mag.*, 35(1):80–86, 1997.

[9] D.L. Tennenhouse and D.J. Wetherall. Towards an active network architecture. In *DARPA Active NEtworks Conf. and Exposition, 2002. Proc.*, pages 2–15. IEEE, 2002.

[10] J.T. Moore and S.M. Nettles. Towards practical programmable packets. In *Proc. 20th Conf. on Computer Commun. (INFOCOM)*. Citeseer, 2001.

[11] Devolved Control of ATM Networks. http://www.cl.cam.ac.uk/research/srg/netos/old-projects/dcan/#pub.

[12] J. E. Van Der Merwe and I. M. Leslie. Switchlets and dynamic virtual atm networks. In *Proc Integrated Network Management V*, pages 355–368. Chapman and Hall, 1997.

[13] J.E. Van der Merwe, S. Rooney, I. Leslie, and S. Crosby. The tempest-a practical framework for network programmability. *IEEE Netw.*, 12(3):20–28, 1998.

[14] J. Rexford, A. Greenberg, G. Hjalmtysson, D.A. Maltz, A. Myers, G. Xie, J. Zhan, and H. Zhang. Network-wide decision making: Toward a wafer-thin control plane. In *Proc. HotNets*, pages 59–64. Citeseer, 2004.

[15] 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 Commun. Review*, 35(5):41–54, 2005.

[16] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. Design and implementation of a routing control platform. In *Proc. 2nd conf. on Symp. on Networked Systems Design & Implementation-Volume 2*, pages 15–28. USENIX Association, 2005.

[17] 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 Commun. Review*, 38(3):105–110, 2008.

[18] R. Enns. NETCONF Configuration Protocol. RFC 4741 (Proposed Standard), December 2006. Obsoleted by RFC 6241.

[19] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin. Simple network management protocol (snmp), rfc1157, 1990.

[20] M. Casado, M.J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. *ACM SIGCOMM Computer Commun. Review*, 37(4):1–12, 2007.

[21] Z. Cai, AL Cox, and TSE Ng. Maestro: A system for scalable openflow control. Technical Report TR10-08, Rice University, December 2010.

[22] Beacon. https://openflow.stanford.edu/display/Beacon/Home.

[23] Simple Network Access Control (SNAC). http://www.openflow.org/wp/snac/.

[24] Helios by nec. http://www.nec.com/.

[25] Andrea Passarella. Review: A survey on content-centric technologies for the current internet: Cdn and p2p solutions. *Comput. Commun.*, 35(1):1–32, January 2012.

[26] Zhiliang Wang, Tina Tsou, Jing Huang, Xingang Shi, and Xia Yin. Analysis of Comparisons between OpenFlow and ForCES, March 2012.

[27] Guohan Lu, Rui Miao, Yongqiang Xiong, and Chuanxiong Guo. Using cpu as a traffic co-processing unit in commodity switches. In *Proc. 1st workshop on Hot topics in software defined networks*, HotSDN '12, pages 31–36, New York, NY, USA, 2012. ACM.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

[28] Jeffrey C. Mogul and Paul Congdon. Hey, you darned counters!: get off my asic! In *Proc. 1st workshop on Hot topics in software defined networks*, HotSDN '12, pages 25–30, New York, NY, USA, 2012. ACM.

[29] Yan Luo, Pablo Cascon, Eric Murray, and Julio Ortega. Accelerating openflow switching with network processors. In *Proc. 5th ACM/IEEE Symp. on Architectures for Networking and Commun. Syst.*, ANCS '09, pages 70–71, New York, NY, USA, 2009. ACM.

[30] Voravit Tanyingyong, Markus Hidell, and Peter Sjödin. Improving pc-based openflow switching performance. In *Proc. 6th ACM/IEEE Symp. on Architectures for Networking and Commun. Syst.*, ANCS '10, pages 13:1–13:2, New York, NY, USA, 2010. ACM.

[31] A. Bianco, R. Birke, L. Giraudo, and M. Palacin. Openflow switching: Data plane performance. In *2010 IEEE Int. Conf. Commun. (ICC)*, pages 1–5, May.

[32] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia. Modeling and performance evaluation of an openflow architecture. In *Teletraffic Congress (ITC), 2011 23rd Int.*, pages 1–7, Sept.

[33] Brent Stephens, Alan Cox, Wes Felter, Colin Dixon, and John Carter. Past: scalable ethernet for data centers. In *Proc. 8th int. conf. on Emerging networking experiments and technologies*, CoNEXT '12, pages 49–60, New York, NY, USA, 2012. ACM.

[34] Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. Devoflow: scaling flow management for high-performance networks. *SIGCOMM Comput. Commun. Rev.*, 41(4):254–265, August 2011.

[35] M. Yu, J. Rexford, M.J. Freedman, and J. Wang. Scalable flow-based networking with difane. In *Proc. ACM SIGCOMM 2010 conf. on SIGCOMM*, pages 351–362. ACM, 2010.

[36] Yossi Kanizo, David Hay, and Isaac Keslassy. Palette: Distributing tables in software-defined networks. In *INFOCOM*, pages 545–549, 2013.

[37] Nanxi Kang, Zhenming Liu, Jennifer Rexford, and David Walker. Optimizing the one big switch abstraction in software-defined networks.

[38] Floodlight, an open sdn controller. http://floodlight.openflowhub.org/.

[39] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood. On controller performance in software-defined networks. In *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, 2012.

[40] Andreas Voellmy and Junchang Wang. Scalable software defined network controllers. In *Proc. ACM SIGCOMM 2012 conf. on Applications, technologies, architectures, and protocols for computer commun.*, SIGCOMM '12, pages 289–290, New York, NY, USA, 2012. ACM.

[41] Brandon Heller, Rob Sherwood, and Nick McKeown. The controller placement problem. In *Proc. 1st workshop on Hot topics in software defined networks*, HotSDN '12, pages 7–12, New York, NY, USA, 2012. ACM.

[42] K. Phemius and M. Bouet. Openflow: Why latency does matter. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE Int. Symp. on*, pages 680–683, 2013.

[43] S.H. Yeganeh, A. Tootoonchian, and Y. Ganjali. On scalability of software-defined networking. *IEEE Commun. Mag.*, 51(2):136–141, February.

[44] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, et al. Onix: A distributed control platform for large-scale production networks. *OSDI, Oct*, 2010.

[45] A. Tootoonchian and Y. Ganjali. Hyperflow: A distributed control plane for openflow. In *Proc. 2010 internet network management conf. on Research on enterprise netw.*, pages 3–3. USENIX Association, 2010.

[46] Dan Levin, Andreas Wundsam, Brandon Heller, Nikhil Handigol, and Anja Feldmann. Logically centralized?: state distribution trade-offs in software defined networks. In *Proc. 1st workshop on Hot topics in software defined networks*, HotSDN '12, pages 1–6, New York, NY, USA, 2012. ACM.

[47] Soheil Hassas Yeganeh and Yashar Ganjali. Kandoo: a framework for efficient and scalable offloading of control applications. In *Proc. 1st workshop on Hot topics in software defined networks*, HotSDN '12, pages 19–24, New York, NY, USA, 2012. ACM.

[48] R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.Y. Huang, P. Kazemian, M. Kobayashi, J. Naous, et al. Carving research slices out of your production networks with openflow. *ACM SIGCOMM Computer Commun. Review*, 40(1):129–130, 2010.

[49] http://www.itu.int/en/ITU T/sdn/Pages/default.aspx. Itu telecommunication standardization sector's sdn portal, 2013.

[50] http://irtf.org/sdnrg. Software-defined networking research group - sdnrg at irtf, 2013.

[51] http://www.openstack.org/. Openstack, 2013.

[52] http://www.opendaylight.org/. Opendaylight, 2013.

[53] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proc. 9th ACM SIGCOMM Workshop on Hot Topics in Netw.*, 2010.

[54] T.R. Henderson, M. Lacage, G.F. Riley, C. Dowell, and JB Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 2008.

[55] Open vswitch and ovs-controller. http://openvswitch.org/.

[56] Pantou: Openflow 1.0 for openwrt. http://www.openflow.org/wk/index.php/ OpenFlow_1.0_for_OpenWRT.

[57] ofsoftswitch13 - cpqd. https://github.com/CPqD/ofsoftswitch13.

[58] Indigo: Open source openflow switches. http://www.openflowhub.org/display/Indigo/.

[59] Pox. http://www.noxrepo.org/pox/about-pox/.

[60] Mul. http://sourceforge.net/p/mul/wiki/Home/.

[61] Trema openflow controller framework. https://github.com/trema/trema.

[62] Jaxon:java-based openflow controller. http://jaxon.onuos.org/.

[63] Ryu. http://osrg.github.com/ryu/.

[64] The nodeflow openflow controller. http://garyberger.net/?p=537.

[65] Node.js. http://nodejs.org/.

[66] Marcelo R. Nascimento, Christian E. Rothenberg, Marcos R. Salvador, Carlos N. A. Corrêa, Sidney C. de Lucena, and Maurício F. Magalhães. Virtual routers as a service: the routeflow approach leveraging software-defined networks. In *Proc. 6th Int. Conf. on Future Internet Technol.*, CFI '11, pages 34–37, New York, NY, USA, 2011. ACM.

[67] Christian Esteve Rothenberg, Marcelo Ribeiro Nascimento, Marcos Rogerio Salvador, Carlos Nilton Araujo Corrêa, Sidney Cunha de Lucena, and Robert Raszuk. Revisiting routing control platforms with the eyes and muscles of software-defined networking. In *Proc. 1st workshop on Hot topics in software defined networks*, HotSDN '12, pages 13–18, New York, NY, USA, 2012. ACM.

[68] M. Canini, D. Venzano, P. Peresini, D. Kostic, and J. Rexford. A nice way to test openflow applications. *NSDI, Apr*, 2012.

[69] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, and Samuel Talmadge King. Debugging the data plane with anteater. In *Proc. ACM SIGCOMM 2011 conf.*, SIGCOMM '11, pages 290–301, New York, NY, USA, 2011. ACM.

[70] Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. Veriflow: verifying network-wide invariants in real time. In *Proc. 1st workshop on Hot topics in software defined networks*, HotSDN '12, pages 49–54, New York, NY, USA, 2012. ACM.

[71] Andreas Wundsam, Dan Levin, Srini Seetharaman, and Anja Feldmann. Ofrewind: enabling record and replay troubleshooting for networks. In *Proc. 2011 USENIX conf. on USENIX annu. technical conf.*, USENIXATC'11, pages 29–29, Berkeley, CA, USA, 2011. USENIX Association.

[72] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Maziéres, and Nick McKeown. Where is the debugger for my software-defined network? In *Proc. 1st workshop on Hot topics in software defined networks*, HotSDN '12, pages 55–60, New York, NY, USA, 2012. ACM.

[73] Sdn troubleshooting simulator. http://ucb-sts.github.com/sts/.

[74] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari. Plug-n-serve: Load-balancing web traffic using openflow. *ACM SIGCOMM Demo*, 2009.

[75] R. Wang, D. Butnariu, and J. Rexford. Openflow-based server load balancing gone wild. In *Workshop of HotICE*, volume 11, 2011.

[76] A.K. Nayak, A. Reimers, N. Feamster, and R. Clark. Resonance: Dynamic access control for enterprise networks. In *Proc. 1st ACM workshop on Research on enterprise networking*, pages 11–18. ACM, 2009.

[77] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella. Toward software-defined middlebox networking. 2012.

[78] Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. Abstractions for network update. In *Proc. ACM SIGCOMM 2012 conf. on Applications, technologies, architectures, and protocols for computer commun.*, SIGCOMM '12, pages 323–334, New York, NY, USA, 2012. ACM.

[79] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. Elastictree: Saving energy in data center networks. In *Proc. 7th USENIX conf. on Networked systems design and implementation*, pages 17–17. USENIX Association, 2010.

[80] H. Shirayanagi, H. Yamada, and K. Kono. Honeyguide: A vm migration-aware network topology for saving energy consumption in data center networks. In *2012 IEEE Symp. Computers and Commun. (ISCC)*, pages 000460–000467. IEEE, 2012.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

[81] Inter-datacenter wan with centralized te using sdn and openflow. In *Open Networking Summit*, April 2012.

[82] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. In *Proc. ACM SIGCOMM 2013 conf. on SIGCOMM*, pages 3–14. ACM, 2013.

[83] K.K. Yap, R. Sherwood, M. Kobayashi, T.Y. Huang, M. Chan, N. Handigol, N. McKeown, and G. Parulkar. Blueprint for introducing innovation into wireless mobile networks. In *Proc. 2nd ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*, pages 25–32. ACM, 2010.

[84] K.K. Yap, M. Kobayashi, R. Sherwood, T.Y. Huang, M. Chan, N. Handigol, and N. McKeown. Openroads: Empowering research in mobile networks. *ACM SIGCOMM Computer Commun. Review*, 40(1):125–126, 2010.

[85] L.E. Li, Z.M. Mao, and J. Rexford. Toward software-defined cellular networks. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 7–12, 2012.

[86] Lalith Suresh, Julius Schulz-Zander, Ruben Merz, Anja Feldmann, and Teresa Vazao. Towards programmable enterprise wlans with odin. In *Proc. 1st workshop on Hot topics in software defined networks*, HotSDN '12, pages 115–120, New York, NY, USA, 2012. ACM.

[87] M. Bansal, J. Mehlman, S. Katti, and P. Levis. Openradio: a programmable wireless dataplane. In *Proc. 1st workshop on Hot topics in software defined networks*, pages 109–114. ACM, 2012.

[88] Optical transport working group otwg. In *Open Networking Foundation ONF*, 2013.

[89] V. Gudla, S. Das, A. Shastri, G. Parulkar, N. McKeown, L. Kazovsky, and S. Yamashita. Experimental demonstration of openflow control of packet and circuit switches. In *Optical Fiber Commun. (OFC), collocated National Fiber Optic Engineers Conf., 2010 Conf. on (OFC/NFOEC)*, pages 1–3, 2010.

[90] Netfpga platform. http://netfpga.org.

[91] Dimitra E. Simeonidou, Reza Nejabati, and Mayur Channegowda. Software defined optical networks technology and infrastructure: Enabling software-defined optical network operations. In *Optical Fiber Commun. Conf./National Fiber Optic Engineers Conf. 2013*, page OTh1H.3. Optical Society of America, 2013.

[92] Lei Liu, T. Tsuritani, I. Morita, Hongxiang Guo, and Jian Wu. Openflow-based wavelength path control in transparent optical networks: A proof-of-concept demonstration. In *Optical Commun. (ECOC), 2011 37th European Conf. and Exhibition on*, pages 1–3, 2011.

[93] A.N. Patel, P.N. Ji, and Ting Wang. Qos-aware optical burst switching in openflow based software-defined optical networks. In *Optical Netw. Design and Modeling (ONDM), 2013 17th Int. Conf. on*, pages 275–280, 2013.

[94] K.L. Calvert, W.K. Edwards, N. Feamster, R.E. Grinter, Y. Deng, and X. Zhou. Instrumenting home networks. *ACM SIGCOMM Computer Commun. Review*, 41(1):84–89, 2011.

[95] N. Feamster. Outsourcing home network security. In *Proc. 2010 ACM SIGCOMM workshop on Home networks*, pages 37–42. ACM, 2010.

[96] R. Mortier, T. Rodden, T. Lodge, D. McAuley, C. Rotsos, AW Moore, A. Koliousis, and J. Sventek. Control and understanding: Owning your home network. In *2012 4th Int. Conf. on Commun. Syst. and Netw. (COMSNETS)*, , pages 1–10. IEEE, 2012.

[97] S. Mehdi, J. Khalid, and S. Khayam. Revisiting traffic anomaly detection using software defined networking. In *Recent Advances in Intrusion Detection*, pages 161–180. Springer, 2011.

[98] Akihiro Nakao. Flare : Open deeply programmable network node architecture. http://netseminar.stanford.edu/10_18_12.html.

[99] R. Bifulco, R. Canonico, M. Brunner, P. Hasselmeyer, and F. Mir. A practical experience in designing an openflow controller. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 61–66, Oct.

[100] Controller performance comparisons. http://www.openflow.org/wk/index.php/Controller_Performance_Comparisons.

[101] David Erickson. The beacon openflow controller, 2012.

[102] Advait Dixit, Fang Hao, Sarit Mukherjee, T.V. Lakshman, and Ramana Kompella. Towards an elastic distributed sdn controller. In *Proc. 2nd ACM SIGCOMM workshop on Hot topics in software defined networking*, HotSDN '13, pages 7–12, New York, NY, USA, 2013. ACM.

[103] Martin Casado, Teemu Koponen, Scott Shenker, and Amin Tootoonchian. Fabric: a retrospective on evolving sdn. In *Proc. 1st workshop on Hot topics in software defined networks*, HotSDN '12, pages 85–90, New York, NY, USA, 2012. ACM.

[104] Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with opensketch. In *Proc. 10th USENIX Symp. on Networked Systems Design and Implementation, NSDI'13*, 2013.

[105] Anja Feldmann. Internet clean-slate design: what and why? *SIGCOMM Comput. Commun. Rev.*, 37(3):59–64, July 2007.

[106] Barath Raghavan, Martín Casado, Teemu Koponen, Sylvia Ratnasamy, Ali Ghodsi, and Scott Shenker. Software-defined internet architecture: decoupling architecture from infrastructure. In *Proc. 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 43–48, New York, NY, USA, 2012. ACM.

[107] R. Bennesby, P. Fonseca, E. Mota, and A. Passito. An inter-as routing component for software-defined networks. In *Network Operations and Management Symp. (NOMS), 2012 IEEE*, pages 138–145, 2012.

[108] E. Kissel, G. Fernandes, M. Jaffee, M. Swany, and M. Zhang. Driving software defined networks with xsp. In *SDN12: Workshop on Software Defined Networks*, 2012.

[109] Andreas Voellmy, Hyojoon Kim, and Nick Feamster. Procera: a language for high-level reactive network control. In *Proc. 1st workshop on Hot topics in software defined networks*, HotSDN '12, pages 43–48, New York, NY, USA, 2012. ACM.

[110] Nate Foster, Rob Harrison, Michael J. Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. Frenetic: a network programming language. In *Proc. 16th ACM SIGPLAN int. conf. on Functional programming*, ICFP '11, pages 279–291, New York, NY, USA, 2011. ACM.

[111] Timothy L. Hinrichs, Natasha S. Gude, Martin Casado, John C. Mitchell, and Scott Shenker. Practical declarative network management. In *Proc. 1st ACM workshop on Research on enterprise networking*, WREN '09, pages 1–10, New York, NY, USA, 2009. ACM.

[112] Andreas Voellmy and Paul Hudak. Nettle: taking the sting out of programming network routers. In *Proc. 13th int. conf. on Practical aspects of declarative languages*, PADL'11, pages 235–249, Berlin, Heidelberg, 2011. Springer-Verlag.

[113] Hyojoon Kim and N. Feamster. Improving network management with software defined networking. *IEEE Commun. Mag.*, 51(2):114–119, February.

[114] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. Composing software-defined networks. In *Proc. 10th USENIX Symp. on Networked Systems Design and Implementation, NSDI'13*, 2013.

[115] Zdravko Bozakov and Panagiotis Papadimitriou. Autoslice: automated and scalable slicing for software-defined networks. In *Proc. 2012 ACM conf. on CoNEXT student workshop*, CoNEXT Student '12, pages 3–4, New York, NY, USA, 2012. ACM.

[116] Connected cloud control: Openflow in mirage. http://www.openmirage.org/blog/announcing-mirage-openflow.

[117] D. Drutskoy, E. Keller, and J. Rexford. Scalable network virtualization in software-defined networks. *IEEE Internet Comput.*, PP(99):1–1.

[118] Soudeh Ghorbani and Matthew Caesar. Walk the line: consistent network updates with bandwidth guarantees. In *Proc. 1st workshop on Hot topics in software defined networks*, HotSDN '12, pages 67–72, New York, NY, USA, 2012. ACM.

[119] Eric Keller, Soudeh Ghorbani, Matt Caesar, and Jennifer Rexford. Live migration of an entire network (and its hosts). In *Proc. 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 109–114, New York, NY, USA, 2012. ACM.

[120] Ramya Raghavendra, Jorge Lobo, and Kang-Won Lee. Dynamic graph query primitives for sdn-based cloudnetwork management. In *Proc. 1st workshop on Hot topics in software defined networks*, HotSDN '12, pages 97–102, New York, NY, USA, 2012. ACM.

[121] Xiang Wang, Zhi Liu, Yaxuan Qi, and Jun Li. Livecloud: A lucid orchestrator for cloud datacenters. In *2012 IEEE 4th Int. Conf. Cloud Computing Technology and Science (CloudCom)*, pages 341–348, 2012.

[122] V. Jacobson, D.K. Smetters, J.D. Thornton, M.F. Plass, N.H. Briggs, and R.L. Braynard. Networking named content. In *Proc. 5th int. conf. on Emerging networking experiments and technologies*, pages 1–12. ACM, 2009.

[123] Xuan-Nam Nguyen, Damien Saucez, and Thierry Turletti. Efficient caching in Content-Centric Networks using OpenFlow. In *2013 IEEE Conf. Computer Commun. Workshops (INFOCOM WKSHPS)*, pages 67–68. IEEE, April 2013.

[124] L. Veltri, G. Morabito, S. Salsano, N. Blefari-Melazzi, and A. Detti. Supporting information-centric functionality in software defined networks. *IEEE ICC Workshop on Software Defined Netw.*, June 2012.

[125] N. Blefari-Melazzi, A. Detti, G. Mazza, G. Morabito, S. Salsano,

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

and L. Veltri. An openflow-based testbed for information centric networking. *Future Network & Mobile Summit*, pages 4–6, 2012.

[126] Junho Suh, Hyogi Jung, Taekyoung Kwon, and Yanghee Choi. C-flow: Content-oriented networking over openflow. In *Open Networking Summit*, April 2012.

[127] D. Syrivelis, G. Parisis, D. Trossen, P. Flegkas, V. Sourlas, T. Korakis, and L. Tassiulas. Pursuing a software defined information-centric network. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 103–108, Oct.

[128] X.N. Nguyen. Software defined networking in wireless mesh network. Msc. thesis, INRIA, UNSA, August 2012.

[129] Cisco visual networking index: Global mobile data traffic forecast update, 2011–2016. Technical report, Cisco, February 2012.

[130] B. Rais, M. Mendonca, T. Turletti, and K. Obraczka. Towards truly heterogeneous internets: Bridging infrastructure-based and infrastructureless networks. In *2011 3rd Int. Conf. Commun. Syst. and Netw. (COMSNETS)*, pages 1–10. IEEE, 2011.

[131] A. Coyle and H. Nguyen. A frequency control algorithm for a mobile adhoc network. In *Military Commun. and Information Syst. Conf. (MilCIS)*, Canberra, Australia, November 2010.

[132] P. Dely, A. Kassler, and N. Bayer. Openflow for wireless mesh networks. In *Proc. 20th Int. Conf. on Computer Commun. and Netw. (ICCCN)*, pages 1–6. IEEE, 2011.

Bruno Astuto A. Nunes is a Pos-doc researcher at INRIA Sophia Antipolis, France. He received his B.Sc. in Electronic Engineering at the Federal University of Rio de Janeiro (UFRJ), Brazil, where he also completed his M.Sc. degree in Computer Science. He received his PhD degree in Computer Engineering from UC Santa Cruz, USA.

Marc Mendonca was a visiting student researcher at the INRIA Planete project in Sophia Antipolis. He recently received his M.S. in Computer Engineering from the University of California, Santa Cruz and previously completed his B.S. in Computer Engineering at the University of California, Santa Barbara.

NGUYEN Xuan-Nam is a Ph.D. candidate in Computer Science at INRIA Sophia Antipolis, France since October, 2012. He received the Bachelor of Engineering from Hanoi University of Science and Technology in 2010, and Master of Science from University of Nice Sophia Antipolis in 2012. From 2010 to 2011, he was a software engineer at Viettel Telecom. His research interests include Software-Defined Networking and Information-Centric Networking.

Katia Obraczka is Professor of Computer Engineering at UC Santa Cruz. Before joining UCSC, she held a research scientist position at USCs Information Sciences Institute and a joint appointment at USCs Computer Science Department. Her research interests span the areas of computer networks, distributed systems, and Internet information systems. She is the director of the Internetwork Research Group (i-NRG) at UCSC and has been a PI and a co-PI in a number of projects sponsored by government agencies (NSF, DARPA, NASA, ARO, DoE, AFOSR) as well as industry. Prof. Obraczka has edited one book, wrote a number of book chapters, and published over 200 technical papers in journals and conferences. She is co-recipient of the Best Paper Award at IFIP Networking 2007 for the paper On-Demand Routing in Disrupted Environments. She is also co-recipient of one of three Best Paper Awards at IEEE MASS 2007 for the paper DYNAMMA: A DYNAmic Multi-channel Medium Access Framework for Wireless Ad Hoc Networks. She has served on the Editorial Board of the Elsevier Ad Hoc Networks Journal. She has been involved with the organization of several technical conferences in her area including IEEE Global Internet, IEEE SECON, IEEE MASS, ACM Mobicom, and ACM Mobihoc, IEEE NetSciCom. She received the USC ISIs Meritorious Service Award in 1999. She is a senior member of the IEEE.

Thierry Turletti received the M.S. (1990) and the Ph.D. (1995) degrees in computer science from the University of Nice - Sophia Antipolis, France. He has done his PhD studies in the RODEO group at INRIA where he designed the Inria Videoconferencing System (IVS). During the year 1995-96, he was a postdoctoral fellow in the Telemedia, Networks and Systems group at LCS, MIT and worked in the area of Software Defined Radio (SDR). He is currently a senior research scientist at the Diana team at INRIA Sophia Antipolis. His current research interests include information centric network architectures, trustable network evaluation platforms and wireless networking. Dr. Turletti has been serving on the Editorial Boards of the following journals: Wireless Communications and Mobile Computing (2001-2010), Wireless Networks (since 2005) and Advance on Multimedia (since 2007).