Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2014, Article ID 735142, 10 pages
http://dx.doi.org/10.1155/2014/735142

Review Article

SDN: Evolution and Opportunities in the Development IoT Applications

Ángel Leonardo Valdivieso Caraguay, Alberto Benito Peral, Lorena Isabel Barona López, and Luis Javier García Villalba

Group of Analysis, Security and Systems (GASS), Department of Software Engineering and Artificial Intelligence (DISIA), School of Computer Science, Office 431, Universidad Complutense de Madrid (UCM), Calle Profesor José García Santesmases s/n, Ciudad Universitaria, 28040 Madrid, Spain

Correspondence should be addressed to Luis Javier García Villalba; javiergv@fdi.ucm.es

Received 9 December 2013; Accepted 27 December 2013; Published 4 May 2014

Academic Editor: Young-Sik Jeong

The exponential growth of devices connected to the network has resulted in the development of new IoT applications and on-line services. However, these advances are limited by the rigidity of the current network infrastructure, in which the administrator has to implement high-level network policies adapting and configuring protocols manually and usually through a command line interface (CLI). At this point, Software-Defined Networking (SDN) appears as a viable alternative network architecture that allows for programming the network and opening the possibility of creating new services and more efficient applications to cover the actual requirements. In this paper, we describe this new technology and analyze its opportunities in the development of IoT applications. Similarly, we present the first applications and projects based on this technology. Finally, we discuss the issues and challenges in its implementation.

1. Introduction

The emergence of new services and applications on-line, both in fixed terminals and mobile devices, has made the communication networks a strategic point in companies, institutions, and homes. The continued evolution of these services and the growth of the information circulating the Internet, bring unanticipated challenges to developers and companies. The advances in micro-electro-mechanical systems (MEMS) increase the development of devices that automatically record, process and send information through the network. This kind of device, mainly consisting of sensors and actuators (RFID, Bluetooth Devices, Wireless Sensor Networks (WSN), Embedded Systems, and Near Field Communication (NFC)), has led to the origin of new ideas, concepts, and paradigms such as the Internet of Things (IoT).

This device uses different ways to connect to the network including the traditional network infrastructure. At present, there are 9 billion connected devices and a number of 24 billon is expected for 2020 [1]. This device uses different ways to connect to the network including the traditional network infrastructure. However, the traditional equipment and network protocols are not designed to support the high level of scalability, high amount of traffic and mobility. The current architectures are inefficient and have significant limitations to satisfy these new requirements.

The infrastructure responsible for transmitting the information coming of IoT devices (routers, switches, 3G and 4G networks, and access points) should be adapted to the new post-PC services (VoIP, sensor virtualization, QoS, cloud computing, and IoT applications) while providing security, stability, high rate, and availability amongst others. Some efforts such as the European Union Projects of SENSEI [2], Internet of Things-Architecture (IoT-A) [3], or Cognitive Management Framework for IoT [4] as well as the new protocols for Wireless Sensor Networks including the Data Driving Routing Protocol [5] have tried to get smarter connectivity between network elements (fully integrated Future Internet). However, these may not be the best options for a particular device or application domains (Smart Grid,

Intelligent Transportation, Smart Home, Health Care, and Environmental Monitoring and others). For this reason, in the last years, the idea of customizing the network behavior has emerged and gives users the flexibility to use the network resources according to their needs. Additionally, the development of new technologies to take decisions on IoT networks uses different calculation algorithms (genetic algorithms, neural networks, evolutionary algorithms, and other artificial intelligence techniques). It is desirable that these algorithms can be easily and dynamically implemented in the network equipment without waiting to be published in a protocol.

Software Defined Networking (SDN) is a network architecture that eliminates the rigidity present in traditional networks. Its structure allows the behavior of the network to be more flexible and adaptable to the needs of each organization, campus, or group of users. Besides, its centralized design allows important information to be collected from the network and used to improve and adapt their policies dynamically. The development in recent years has impulsed new concepts, such as the network operating system (NOS). NOS tries to emulate the progress in computer systems. In this paper, the evolution of the main NOS is also analyzed. With this tool, it is possible to test the SDN concept in multiple projects (Home Networking, Data centers, Security, Virtualization, and Multimedia among others). Similarly, SDN has led to the design of models that integrate and finally achieve convergence of commonly separate architectures (Wifi-4G-LTE). However, these opportunities are still far from being implemented globally in production. Important issues such as convergence with existing networks, scalability, performance, and security are the challenges that should be overcome to be positioned in the market.

In this piece of work, we describe SDN and its evolution in the last years and analyze the opportunities and challenges in the future for this technology. Specially, the development of new generations IoT application (Smart Environment). The work is structured as follows: in Section 2 the limitations of traditional networks is presented; next, Section 3 defines the Software-Defined Networking concept; Section 4 presents the evolution of Network Operating Systems (NOS); then, in Section 5, the first SDN applications are reviewed; in Section 6, the challenges of SDN technology are discussed; finally, Section 7 presents the conclusions.

2. Limitations of Traditional Architectures

The idea of transmitting information between two points through a network led to the design of communication protocols (TCP/IP, HTTPS, and DNS) and the creation of specialized devices in the transmission of information. These devices have evolved resulting in a variety of equipment (hub, switch, router, firewall, IDS, middlebox, and filters). This development has produced an exponential increase in the number of connected devices, in addition to the increase of the transmission rate and the emergence of online services (e-banking, e-commerce, e-mail, VoIP, etc.).

All the devices responsible for transmitting information have similar features in their design and manufacture. First, there is a specialized hardware in the packet processing (data plane), and over the hardware works an operating system (usually Linux) that receives information from the hardware and runs a software application (control plane). The software contains thousands of lines of code for determining the next hop that a packet should be taken in order to reach its destination. The program follows the rules defined by a specific protocol (there are currently about 7000 RFCs) or some proprietary technology of the vendor. Modern equipment also analyzes information packets to search malicious information or intrusions (firewalls and IDS). However, all technology or software used in the manufacturing of these devices is rigid or closed to the network administrator. The administrator is limited only to configure some parameters, usually through low level commands using a command line interface (CLI). Moreover, each node is an autonomous system which finds the next hop to be taken by a packet to reach its destination. Some protocols (OSPF, BGP) allow the nodes to share control information between them, but only with its immediate neighbors and in a limited way in order to avoid tradditional load on network. This means that there is not a global view of the network as a whole. If the users need to control and modify a particular path, the administrator has to test with parameters, priorities, or uses gadgets to achieve the expected behavior in the network. Each change in the network policy requires individual configuration directly or remotely from each of the devices. This rigidity makes the implementation of high-level network policies difficult. Moreover, the policies require to be adaptive and dynamically react according to the network conditions.

As Operating Systems (OS) evolve and adapt to the new needs and technological trends (support multi-CPU, multi-GPU, 3D, touch screen support, etc.), the network adaptability to new requirements (VLAN, IPv6, QoS, and VoIP) is implemented through protocols or RFCs. However, in the operating system the separation between hardware and software allows the continuous update of application, or even the reinstallation of a new version of an OS. In the area of networks, the design and implementation period of a new idea could take several years until it is published in a protocol and incorporated in new devices. Some services are proprietary of the vendors and require that all network infrastructure belong to the same vendor to work properly. This limitation brings on the dependence on a specific technology or vendor.

3. Software-Defined Networking SDN

The concept of Software-Defined Networking is not new and completely revolutionary; rather it arises as the result of contributions, ideas, and developments in research networking. In [6], three important states are determined in the evolution of SDN: Active Networks (mid-90s to early 2000), separation of data and control planes (2001–2007), and the OpenFlow API and NOS (2007–2010). All these aspects are discussed below.

The difficulty for researchers to test new ideas in a real infrastructure and the time, effort and resources needed to standardize these ideas on the Internet Engineering

Task Force (IETF) necessarily give some programmability to network devices. Active networks offer a programmable network interface or API that opens the individual resources of each node for the users, such as processing, memory resources, and packet processing and includes personalized features for the packets that circulate through the node. The need to use different programming models in the nodes was the first step for research in network virtualization, as well as the development of frameworks or platforms for the development of application on the node. The Architectural Framework for Active Networks v1.0 [6, 7] contains a shared Node Operating System (NodeOS), a set of execution environments (Execution Environments (EEs)), and active applications (Active Applications (AAs)). The NodeOS manages the shared resources, while the EE defines a virtual machine for the packet operations. The AA operates within an EE and provides the end-to-end service. The separation of packets to each EE depends on a pattern in the header of incoming packets to the node. This model was used in the PlanetLab [8] platform, where researchers conducted experiments in virtual execution environments and packets were demultiplexed to each virtual environment based on its header. These developments were important, especially in the investigation of architectures, platforms, and programming models in networks. However, their applicability in industry was limited and mainly criticized for its limitations in performance and safety. The work presented in [9] is an effort to provide the best performance to the active networks, and the Secure Active Network Environment Architecture [10] tried to improve their security.

The exponential growth in the volume of traffic over the network produces the necessity to improve the supervision process and uses best management functions such as the management of paths or links circulating the network (traffic engineering), prediction traffic, reaction, and fast recovery if there are network problems, among others. However, the development of these technologies has been strongly limited by the close connection between the hardware and software of networking devices. Besides, the continuous increase in link rates (backbones) means that the whole transmission mechanism of packets (packet forwarding) is focused on the hardware, separating control, or network management to an application of software. These applications work best on a server, because it has higher processing and memory resources compared with a single network device. In this sense, the project ForCES (Forwarding and Control Element Separation) [11] standardized by the IETF (RFC 3746) established an interface between data and control plane in the network nodes. The SoftRouter [12] used this software interface to install forwarding tables in the data plane of routers. Additionally, the Routing Control Platform (RCP) [13] project proposed logical centralized control of the network, thus facilitating the management, the innovation capacity, and programming of network. RCP had an immediate applicability because it uses an existing control protocol BGP (Border Gateway Protocol) to install entries in the routing tables of the routers.

The separation of data plane and the control plane allows the development of "clean-slate" architectures, such as the 4D project [14] and Ethanet [15]. 4D architecture proposes architecture of four layers based on functionality: data plane, discovery plane, dissemination plane, and decision plane. Moreover, the Ethanet project [15] proposes a centralized control system of links to business networks. However, the need for custom switches based on Linux, OpenWrt, NetFPGA with support for Ethane protocol made it difficult the applicability of this project. At the present time, the OpenFlow protocol [16] is the most widely used in the research community and it has been the basis of different projects. Companies like Cisco have also submitted a proposal for a new architecture called Cisco Open Network Environment (Cisco ONE).

Simplifying the previous analysis, the term Software-Defined Networking proposes some changes to the networks of today. First, the separation or decoupling of the data plane and control plane, allowing evolution and development independently. Secondly, it proposes a centralized control plane, thus having a global view of the network. Finally, SDN establishes open interfaces between the control plane and data plane. The differences between these architectures are shown in Figure 1.

The programmability of the network provided by SDN can be compared with the mobile applications running on an Operating System (Android and Windows Mobile). These applications use the resources of the mobile (GPS, accelerometer, and memory) through the API provided by the OS. Likewise, the network administrator can manage and program resources in the network, according to user needs, through available APIs (proprietary or open) on the controller.

3.1. OpenFlow. OpenFlow [16] was originally proposed as an alternative for the development of experimental protocols on university campus, where it is possible to test new algorithms without disrupt or interfere with the normal operation of traffic of other users. Nowadays, the Open Networking Foundation (ONF) [6] is the organization responsible for the publication of the OpenFlow protocol and other protocols for SDN, such as OF-Config [17].

The advantage of OpenFlow, compared with previous SDN protocols, is the use of elements and features of hardware available in most network devices. These elements are the routing tables and the common functions are as follows: read the header, send the packet to a port, and drop a packet, among others. OpenFlow opens up these elements and functions; so these can be controlled externally. This implies that, with a firmware update, the actual hardware could potentially support OpenFlow. The companies do not need a complete change of their hardware to implement SDN in their products and services.

The OpenFlow architecture proposes the existence of a controller, a switch OpenFlow, and a secure protocol of communication. These elements are shown in Figure 2. Each OpenFlow switch consists of flow tables that are managed by the controller. Each flow table has three elements: packet header, actions, and statistics. The packet header is like a mask that select the packets which will be processed by the switch. The fields used for comparison can be from

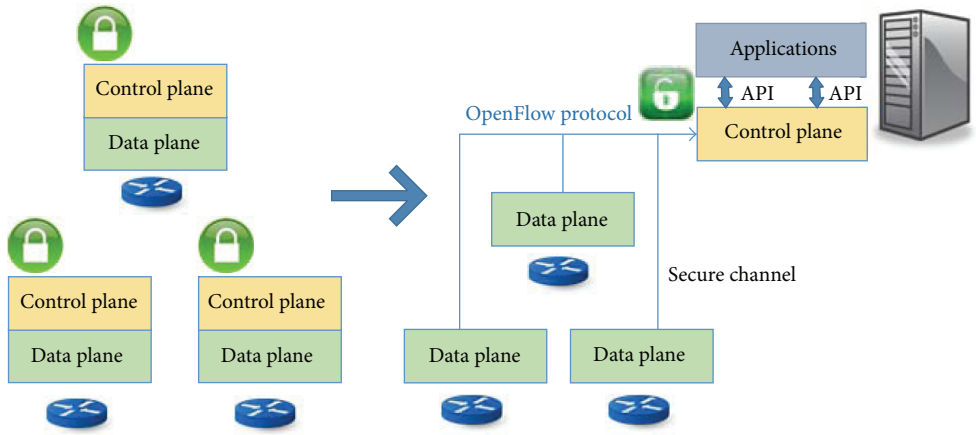

Figure 1: Comparison between traditional and SDN architectures.

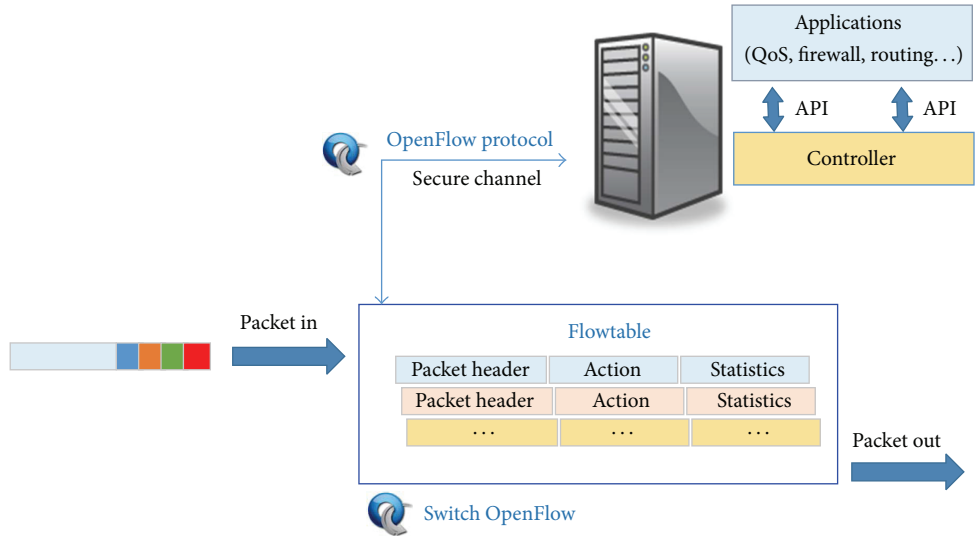

Figure 2: Elements of the OpenFlow architecture.

layer 2, 3, or 4 of the TCP/IP architecture. That means that there is not a separation between layers as in current architectures. All packets processed by the switch are filtered through this method. The number of fields that the switch can process depends on the version of the OpenFlow protocol. In OpenFlow v1.0 [18] (the most used version), there are 12 fields, while the latest version OpenFlow v1.3 defines the existence of 40 fields including support for IPv6.

Once the header of an incoming packet matches the packet header of the flow table, the corresponding actions for that mask are performed by the switch. There are main and optional actions. The main actions are as follows: forward the packet to a particular port, encapsulate the packet and send it to the controller, and drop the packet. Some optional actions are as follows: forward a packet through a queue attached to a port (enqueue action) or 802.1D processing capabilities. If the header of an incoming packet does not match with the packet header of the flow table, the switch (according to its configuration) sends the packet to the controller for its analysis and treatment. Finally, the statistics field uses counters to collect statistic information for administration purposes.

The OpenFlow protocol defines the following types of messages between the switch and the controller: controller to switch, symmetric, and asynchronous. The messages type controller to switch manage the state of the switch. Symmetric messages are sent by the controller or switch to initiate the connection or interchange of messages. The asynchronous messages update the control of the network events and the changes of state switch. Similarly, OpenFlow establishes two types of switches: OpenFlow-only and OpenFlow-enabled. OpenFlow-only switches use only OpenFlow protocol to process packets. On the other side, OpenFlow-enabled switches can additionally process the packet using traditional algorithms of switching or routing.

The controller receives the information from the various switches and remotely configures the flow tables of the switch. Here, the user can literally program the behavior of

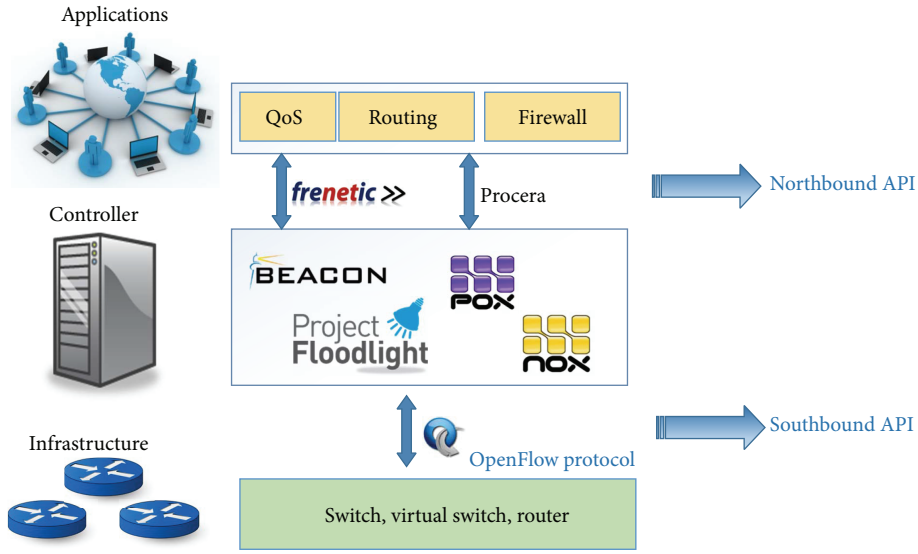

Figure 3: NOS, southbound, and northbound interfaces.

the network. Unlike active networks, which proposed a "Node Operating System." OpenFlow opens the notion of a Network Operating System (NOS). In this respect, in [19], the NOS is defined as the software that abstracts the installation of the state in the switches of network of the logic and applications that control the behavior of the network. In recent years, the NOS has evolved according to the needs and applications for researchers and network administrators.

4. Network Operating Systems Evolution (NOS)

The concept of Network Operating Systems (NOS) is based on the function of an operating system in computing. That is, the Operating System allows user to create applications using high-level abstraction of information, resources, and hardware. In SDN, some authors [20–22] have classified the abstractions of network resources as southbound and northbound interfaces (Figure 3). The function of the southbound interfaces is to abstract the functionality of the programmable switch and connect it to the controller software. A clear example of southbound interface is OpenFlow. On the southbound interfaces, you run a Network Operating Systems. An example of NOS is NOX [23–25], among others. On the other hand, the northbound interfaces allow applications or high-level network policies to be easily created and they transmit these tasks to Network Operating System (NOS). Examples of these interfaces are Frenetic [26, 27], Procera [21, 28], Netcore [29], and McNettle [30]. Then, they are analyzed in the main NOS and northbound interfaces.

The NOX software [23] is the first NOS for OpenFlow and consists of 2 elements: processes of controller (controller) and a global view of the network (network view). Depending on the current state of the network, the user can make decisions and set the network behavior through these processes. In NOX, traffic is handled at the level of flows (flow-based granularity); that is, all packets with the same header are treated similarly. The controller inserts, deletes entries, and reads the counters found in the flow tables of the switches. Furthermore, due to the dynamic nature of traffic, NOX uses events (event handlers) that are registered with different priorities to be executed when a specific event occurs in the network. The most used events are switch join, switch leave, packet received, and switch statistics received. Additionally, NOX includes "system libraries" implementations and common network services. Finally, NOX is implemented in C++ providing high performance. Moreover, there is an implementation entirely in Python denominated POX, which provides a more friendly developed language.

Beacon [24] is a Java-based OpenFlow controller. Its interface is simple and unrestricted; that is, the user can freely use the constructors available in Java (threads, timers, sockets, etc.). Furthermore, Beacon is a NOS based on events; that is, the user sets the events that the controller listens to. The interaction with OpenFlow messages of the switch is done by the library OpenFlowJ, an implementation of the OpenFlow 1.0 [18] protocol and IBeaconProvider interface that contains the following listeners: IOFSwitchListener, IOFInitializerListener, and IOFMessageListener. Additionally, Beacon has multithreading support and provides important APIs implementations (Device Manager, Topology, Routing, and Web UI) as well as the ability to start, add, and complete applications without completely terminating a process in Beacon (runtime modularity).

Although a NOS can handle the flow tables of the switches, there are some problems that can cause malfunction of the network [20, 22, 31]. For example, the controller receives the first packet that arrives at the switch and has not matched a header in the flow table. Then the controller analyzes it, assigns actions, and forwards these instructions to the switch so that the other similar packages follow the same route. However, during this time, the second, third, or fourth similar packets can be received by the controller and cause an erratic operation. In other words, there are

virtually two processes running, one on the controller and another on the switch, and these processes are not fully synchronized.

Another limitation is the composition; that is, if the user wants to configure two different services on the same switch (e.g., routing and monitoring), it is necessary to manually combine the two actions on the switch, prioritize, and keep the semantics of each element of the network. This makes the design, coordination, and reuse of the libraries very difficult. Additionally, the switch has to handle two types of messages simultaneously: packets and control messages. Any mismatch can cause a packet to be processed with an invalid policy and thereby causing major security problem on the network. For example, if there are two entries in a flow table with the same priority, the switch behavior might be nondeterministic, because the execution would depend on the design of the switch hardware. For this reason, the research community has worked on secure interfaces that automatically interact and coordinate the correct behavior of the switch (northbound).

Procera [21, 28] is a framework that allows politics or high-level network configurations to be expressed. This architecture provides different actions and control domains to program the behavior of the network. The main domains of control are as follows: time, data usage, flow, and status. With these domains, the user can determine a behavior depending, for example, on the time of day, amount of data transmitted, privileges or groups of users, type of transmitted traffic, and so forth. Actions can be temporal or reactive and are expressed on a high-level language based on Functional Reactive Programming (FRP) and Haskell. In [21] are the details of this language as well as examples of using Procera in monitoring applications and users control on a college campus.

Frenetic [26, 27] is a high-level language dedicated to SDN networks developed in Python. It is structured by 2 sublanguages: a Network Query Language and a Reactive Network Policy Management Library. The Network Query Language allows the user to read the status of the network. This task is performed by installing rules (low-levels rules) on the switch which does not affect the normal operation of the network. In addition, the Network Policy Management Library is designed based on a language for robots, Yampa, [32] and web programming libraries in Flapjax [33]. The actions use a constructor type rule containing a pattern or filters and action list as arguments. The main actions are as follows: sending to a particular port, sending packet to the controller, modification the packet header, and blank action that is interpreted as discard the packet. The installation of these policies is performed by generating policy events (queries), primitive events (Seconds, SwitchJoin SwitchExit, and PortChange), and listener (Print and Register). The results of experiments [26] show that Frenetic provides simplicity and a significant savings in code and lower consumption of network resources compared to NOX.

One of the additional advantages of this language is the composition; that is, independent functional modules can be written and the runtime system coordinates its proper function in the controller and the switch. There are 2 types of composition: sequential and parallel. In sequential composition, the output of one module is the input of the next, for example, a load balancer that first modifies the IP destination of a packet and then searches the output port according to the new IP header. In parallel composition, both modules are executed virtually simultaneously in the controller; for example, if the balancer sends a packet with destination IP A to port 1, and packet B IP destined to port 2, this composition would result in a function that sends incoming packets for ports 1 and 2.

McNettle [30] is a controller specially designed to offer high scalability at the SDN network. This is achieved using a set of message handlers (one for each switch) having a function that handles the switch-local and network-state variables and manages the supply actions from the network flows. The idea is that the messages from the same switch are handled sequentially, while messages from different switches are handled concurrently. Similarly, each message is processed in a single core CPU to minimize the number of connections and synchronizations inter-cores among other performance improvements. The tests performed in [30] show that McNettle have a higher multicore performance compared to NOX or Beacon.

The controller proposed in [31] is based on the verification of the established politics, instead of searching bugs monitoring the controller operation. To perform the verification, the first step is to make use of the high-level language Netcore [29] to describe only the network behavior. Then, the Netcore Compiler translates the politics to network configurations as flow table entries. The flow tables information is analyzed by the Verifier Run-time System which transforms the network configuration into a lower abstraction level named Featherweght OpenFlow. Featherweght Openflow is a model that use synchronization primitives to guarantee the coherent behavior of the flowtables. Additionally, the Kinetic tool is described in [22]; this tool allows performing consistent updates in the network using two mechanisms: per-packet consistency and per-flow consistency. The per-packet consistency mechanism ensures that a packet that is transmitted across the network is processed with the same configuration when an update occurs. The per-flow consistency mechanism ensures that every packet that belongs to the same flow (e.g., a TCP connection) will be processed in the same way by every switch in the network.

5. First SDN Applications

Software-Defined Networking provides the ability to modify the network behavior according to user needs. In other words, SDN itself doesnt solve any particular problem, but provides a more flexible tool to improve the network management. In order to test the advantages of this architecture, the research community has presented multiple projects of interest. Next, some of these applications are described.

5.1. Home Networking. In the emerging topic of Internet of Things (IoT), the management of devices and network

resources in home networks is a big challenge due to the number of users and devices connected to the same point (usually an access point). In [21, 34], the authors present an implementation of an OpenFlow-based system that allows the monitoring and management of user and control of the Internet access based on "usage caps" or a limited data capacity for each user or device. The system provides visibility of the network resources and management of access based on user, group, device, application, or time of day and even enables the ability to exchange data capacity with another user. The system of control and network monitoring uses the friendly interface Kermit. The capacity management and network policies are based on the Resonance language [35].

5.2. Security. The global vision of the network can improve the security of the systems. This security cannot be based only in the host-security, because such defenses are ineffective when the host is compromised. In [36], the Pedigree system is presented as an alternative to provide security in the traffic moving in an enterprise network. This OpenFlow-based system allows to the controller the analysis and the approval of connections and traffic flows in the network. The host has a security module in the kernel (tagger) that is not under users control. This module labels the connections request to send information through the network (processes, files, etc.). This label is sent to the controller (arbiter) in the start of the communication. The controller analyzes the tagger and accepts or rejects the connection according to its policies. Once the connection is authorized, the corresponding flow tables are installed in the switch. Pedigree increases the tolerance to a variety of attacks, such as polymorphic worms. The systems increase the load in the network traffic and the host. However, this load is not higher than common antivirus software.

5.3. Virtualization. The concept of virtualization in networks is similar to OS-virtualization, where different Operating Systems can share hardware resources. That is, in network virtualization, it is intended that multiple virtual networks can operate on the same infrastructure, each with its own topology and routing logic. Initially, VLAN technologies and private virtual networks allow the different users to share network resources. However, the separation is controlled only by the network administration and with limited parameters (port number) and just work with known network protocols. With the SDN data-control separation, the possibilities to create new advanced virtual networks are promising. For example, Flowvisor [37, 38] is an OpenFlow-based project that allows creating slices based on multiple parameters, such as bandwidth, flowspace (src/dst MAC, src/dst IP, and src/dst TCP ports), or CPU switch load. Each slice is independent; that means that it does not affect the traffic of the others slices. Additionally, it is possible to subdivide slices in order to create hierarchical models. A network service that takes advantage of virtualization of network resources is the migration of virtual networks. In [39], a system that enables the migration of the switch configuration to another device into the network is proposed, but without disrupting the active network traffic. The controller copies the flow tables configuration from the old to the new switch and modifies the paths automatically. This service enables the possibility to replace a network device avoiding the disruption or packet loss. This advantage can be used to dynamically modify the resources used in the network (green networks). In other words, the network can turn off or disable the unnecessary devices (nights or weekends) and automatically enable them in function of the traffic demand (peak hours).

5.4. Mobile Networks. The devices in the infrastructure on mobile carrier networks share similar limitations as computer networks. Likewise, the carrier networks execute standards and protocols, for example, the Third Generation Partnership Project (3GPP) as well as the private vendors implementations. At this point, the SDN paradigm and its flow-based model can be applied on this kind of infrastructure offering better tools. Software-Defined Mobile Network (SDMN) [40] is an architecture that enables openness, innovation, and programmability to operators, without depending on exclusive vendors or over the top (OTT) service providers. This model consists of two elements: MobileFlow Forwarding Engine (MFFE) and the MobileFlow Controller (MFC). MFFE is a simple and stable data plane and with high performance. It has a more complex structure than an OpenFlow switch, because it must support additional carrier functions, such as layer 3 tunneling (i.e., GTP-U and GRE), access network nodes functions, and flexible charging. The MFC is the high performance control plane, where the mobile networks applications can be developed. Additionally, MFC has 3GPP interfaces to interconnect with different Mobile Management Entities (MMEs), Serving Gateways (SGWs), or Packet Data Network Gateways (PGWs).

5.5. Multimedia. The multiple online multimedia services, for example, the real time transmissions, require high levels of efficiency and availability of the network infrastructure. According to studies presented by CISCO, the IP video traffic will grow from 60% in 2012 to 73% by 2017 [41]. Moreover, in the last years, the concept of Quality of Experience (QoE) [42] gained particular strength, which attempts to redefine the Quality of Service (QoS) considering the level of user acceptance to a particular service or multimedia application. Therefore, SDN allows the optimization of the multimedia management tasks. For example, in [43] is improved the QoE experience through the path optimization. This architecture consists of two elements: the QoS Matching and Optimization Function (QMOF) that reads the different multimedia parameters and establishes the appropriate configuration for this path, and the Path Assignment Function (PAF) that regularly updates the network topology. In case of degradation of the quality on the links, the system automatically modifies the path parameters taking in count the priorities of the users. Similarly, the project OpenFlow-assisted QoE Fairness Framework (QFF) [44] analyzes the traffic in the network and identifies the multimedia transmissions in order to optimize

them in function of the terminal devices and the network requirements.

5.6. Reliability and Recovery. One of the most common problems in the traditional networks is the hardness to recover a link failure. The convergence time is affected by the limited information of the node to recalculate the route. In some cases, it is necessary the intervention of the network administrator to reestablish the network datapath. At this point, the global vision of SDN enables the customizing of recovery algorithms. [45] proposed an OpenFlow-based system that uses the mechanism of restoration and protection to calculate an alternative path. In restoration mechanism, the controller looks for an alternative path when the fail signal is received. Meanwhile, in protection the system anticipates a failure and previously calculates an alternative path. Similar to a failure on switch or routers, the malfunction of the SDN controller (NOS failure, DDoS attack, and application error) can cause a collapse of the whole network. Therefore, the reliability of the network can be ensured through backup controllers. However, it is necessary to coordinate and update the information of control and configuration between principal and backup controllers. The CPRecovery [46] component is a primary backup mechanism that enables the replication of information between primary and backup controller. The system uses the replication phase to maintain the updated backup controller and the phase of recovery that starts the controller backup at the moment it detects a failure of the principal controller.

6. Challenges of SDN Technology

The SDN advantages as applied technology in production networks are still close but not immediate. Furthermore, there are some challenges in terms of security, scalability, and reliability, among other aspects, which must be overcome in order to be considered acceptable for commercial users. Next, these aspects are analyzed.

As it was previously explained, the separation between data and control plane enables their independent development and evolution. In data plane, the rate of packet processing depends on the used hardware technology, such as *Application-Specific Integrated Circuits (ASIC), Application-Specific Standard Products (ASSP), Field Programmable Gate Array (FPGA), or multicore CPU/GPP*. Meanwhile, in control plane the performance also depends principally on the hardware and the NOS (Beacon, POX, Floodlight). However, a poor performance of one of the two levels can cause significant problems, such as packet loss or delay and incorrect behavior of the network of denial of service DDoS. For this reason, a balance in performance, cost, and facility of development for the hardware and software of SDN components is necessary.

Moreover, OpenFlow uses the common hardware resources of actual networks, such as the flow tables. However, SDN can be extended beyond flow tables and use additional resources offered by actual hardware [17]. The integration and research of new features between control and data plane is a recently open topic. Applications like encryption, analysis, and traffic classification and devices such as middleboxes and custom packet processors can be integrated and efficiently used by the SDN technology. On the other hand, the number and the position of the controllers in a network are open questions. The analysis presented in [47] exposes that the determining factors for the selection of the number and position are the topology and the expected performance of the network.

The security is another fundamental aspect that must also be taken into account. For example, not all network applications should have the same access privileges [20]. The assignment of profiles, authentication, and authorization to access the network resources are necessary. In addition, OpenFlow establishes the optional use of TLS (Transport Layer Security) as authentication tool between switch and controller. However, there are not clear specifications that provide security for multiple controller systems that interchange information among them and with the switches. Additionally, Openflow establishes that an unknown packet could be send completely (or its packet header) to the controller, it can easily be affected by DDoS attacks by the sending of multiple unknown packets to the switch.

The transition between actual network architectures to SDN-based architectures is also an open issue. Despite the emergence of network devices with OpenFlow support (IBM and NEC) in the market, it is impossible to replace the network infrastructure completely. The transition period requires mechanism, protocols, and interfaces allowing coexistence between both architectures. Currently, there are important efforts to achieve this objective; the Open Networking Foundation (ONF) published the IF-Config Protocol [48] as a first step to the configuration of OpenFlow devices. Similarly, the European Telecommunications Standards Institute (ETSI) as well as the el IETFs Forwarding and Control Element Separation Working Group (ForCES) works on the standardization of interfaces for the appropriated development of this technology.

7. Conclusion

The exponential growth of devices and online services that exchange information over the network consolidated the concept of Internet of Things (IoT). In this new approach, the rigidity of traditional architectures is inefficient suggesting rethinking new ways to use the infrastructure and technology communications. Software-Defined Networking has emerged as an alternative to the current problems of traditional networks. It allows administrators to have a global view of the network, as well as the opportunity to control the network according to the needs of each organization.

This work presents the basis of this technology and the development of Network Operating Systems NOS, as well as some interesting projects based on this paradigm. Additionally, the problems and challenges for the implementation of SDN in production networks are analyzed. It is noteworthy that SDN provides the tools to improve the management of the network behavior. The use of this tool and the development of new SDN applications are new fields of study. In the future, this paradigm will bring new ways

of viewing and using the communication networks as well as new business models focused on offering services and network applications.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

Part of the computations of this work were performed in EOLO, the HPC of Climate Change of the International Campus of Excellence of Moncloa, funded by MECD and MICINN. This is a contribution to CEI Moncloa. Ángel Leonardo Valdivieso Caraguay and Lorena Isabel Barona López are supported by the Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación SENESCYT (Quito, Ecuador) under Convocatoria Abierta 2012 Scholarship Program no. 2543-2012. The authors would also like to thank Ana Lucila Sandoval Orozco for her valuable comments and suggestions to improve the quality of the paper.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): a vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] SENSEI, Integrated EU Project, http://www.ict-sensei.org/.

[3] European Lighthouse Integrated Project, Internet of Things Architecture, http://www.iot-a.eu/public/.

[4] P. Vlacheas, R. Giaffreda, V. Stavroulaki et al., "Enabling smart cities through a cognitive management framework for the internet of things," in *IEEE Communications Magazine*, vol. 51, pp. 102–111, 2013.

[5] L. Shi, B. Zhang, H. T. Mouftah, and J. Ma, "DDRP: an efficient data-driven routing protocol for wireless sensor networks with mobile sinks," *International Journal of Communication Systems*, vol. 26, no. 10, pp. 1341–1355, 2013.

[6] Open Networking Foundation, https://www.opennetworking.org/.

[7] K. Calvert, *Architectural Framework for Active Networks Version 1.0*, 1999.

[8] PlanetLab, https://www.planet-lab.org/.

[9] T. Wolf and J. S. Turner, "Design issues for high-performance active routers," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 3, pp. 404–409, 2001.

[10] D. Alexander, W. Arbaugh, A. Keromytis, and J. Smith, "A secure active network environment architecture: realization in SwitchWare," *IEEE Network*, vol. 12, no. 3, pp. 37–45, 1998.

[11] L. Yang, R. Dantu, T. Anderson, and R. Gopal, "Forwarding and Control Element Separation (ForCES) framework," RFC 3746, 2004.

[12] T. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, and T. Woo, "The soft- router architecture," in *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Networking*, November 2004.

[13] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. Van der Merwe, "Design and implementation of a routing control platform," in *Proceedings of the 2nd Conference on Symposium on Networked Systems Design and Implementation (NSDI '05)*, vol. 2, pp. 15–28, USENIX Association, May 2005.

[14] A. Greenberg, G. Hjaimtysson, D. Maltz et al., "A clean slate 4D approach to network control and management," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 41–54, 2005.

[15] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Mckeown, and S. Shenker, "Ethane: taking control of the enterprise," in *Proceedings of the ACM SIGCOMM 2007: Conference on Computer Communications*, vol. 37, pp. 1–12, August 2007.

[16] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 69–74, 2008.

[17] A. Valdivieso, L. Barona, and L. Villalba, "Evolution and challenges of software defined networking," in *Proceedings of the 2013 Workshop on Software Defined Networks for Future Networks and Services*, pp. 61–67, IEEE, November 2013.

[18] OpenFlow Switch Specification v.1.0.0, December 2009.

[19] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: an intellectual history of programmable networks," Technical Report, Princeton University, Princeton, NJ, USA, 2013.

[20] S. Sezer, S. Scott-Hayward, P. K. Chouhan et al., "Are we ready for SDN? Implementation challenges for software-defined networks," in *IEEE Communications Magazine*, vol. 51, pp. 36–43, 2013.

[21] H. Kim and N. Feamster, "Improving network management with software defined networking," in *IEEE Communications Magazine*, vol. 51, pp. 114–119, 2013.

[22] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *ACM SIGCOMM Computer Communication Review*, vol. 42, pp. 323–334, 2012.

[23] N. Gude, T. Koponen, J. Pettit et al., "NOX: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[24] D. Erickson, "The beacon OpenFlow controller," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot topics in Software Defined Networking*, pp. 13–18, ACM, August 2013.

[25] Floodlight, http://www.projectfloodlight.org/.

[26] N. Foster, R. Harrison, M. J. Freedman et al., "Frenetic: a network programming language," *ACM SIGPLAN Notices*, vol. 46, no. 9, pp. 279–291, 2011.

[27] N. Foster, A. Guha, M. Reitblatt et al., "Languages for software defined networks," in *IEEE Communications Magazine*, vol. 51, pp. 128–134, 2013.

[28] A. Voellmy, H. Kim, and N. Feamster, "Procera: a language for high-level reactive network control," in *Proceedings of the 1st Workshop on Hot topics in Software Defined Networks*, pp. 43–48, ACM, August 2012.

[29] C. Monsanto, N. Foster, R. Harrison, and D. Walker, "A compiler and run-time system for network programming languages," *ACM SIGPLAN Notices*, vol. 47, no. 1, pp. 217–230, 2012.

[30] A. Voellmy and J. Wang, "Scalable software defined network controllers," in *ACM SIGCOMM Computer Communication Review*, vol. 42, pp. 289–290, ACM, 2012.

[31] A. Guha, M. Reitblatt, and N. Foster, "Machine-verified network controllers," in *ACM SIGPLAN NOTICES*, vol. 48, pp. 483–494, ACM, 2013.

[32] A. Courtney, H. Nilsson, and J. Peterson, "The Yampa arcade," in *Proceedings of the ACM SIGPLAN 2003 Haskell Workshop*, pp. 7–18, ACM, August 2003.

[33] L. A. Meyerovich, A. Guha, J. Baskin et al., "Flapjax: a programming language for ajax applications," *ACM SIGPLAN Notices*, vol. 44, no. 10, pp. 1–20, 2009.

[34] H. Kim, S. Sundaresan, M. Chetty, N. Feamster, and W. K. Edwards, "Communicating with caps: managing usage caps in home networks," in *Proceedings of the ACM SIGCOMM 2011 Conference (SIGCOMM '11)*, vol. 41, pp. 470–471, August 2011.

[35] A. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: dynamic access control for enterprise networks," in *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking (WREN '09)*, pp. 11–18, ACM, August 2009.

[36] A. Ramachandran, Y. Mundada, M. B. Tariq, and N. Feamster, *Securing Enterprise Networks Using Traffic Tainting*, 2009.

[37] R. Sherwood, G. Gibb, and M. Kobayashi, "Carving research slices out of your production networks with OpenFlow," in *ACM SIGCOMM Computer Communication Review*, vol. 40, pp. 129–130, ACM, 2010.

[38] R. Sherwood, G. Gibb, K. K. Yap et al., "Flowvisor: a network virtualization layer," in *OpenFlow Switch Consortium*, 2009.

[39] P. S. Pisa, N. C. Fernandes, H. E. Carvalho et al., "OpenFlow and Xen-based virtual network migration," in *Wireless in Developing Countries and Networks of the Future*, vol. 327, pp. 170–181, Springer, Berlin, Germany, 2010.

[40] K. Pentikousis, Y. Wang, and W. Hu, "MobileFlow: toward software-defined mobile networks," in *IEEE Communications Magazine*, vol. 51, pp. 44–53, 2013.

[41] The Zettabyte EraTrends and Analysis, May 2013.

[42] S. M. Patrick Le Callet and A. Perkis, "Qualinet white paper on definitions of quality of experience," in *Proceedings of the European Network on Quality of Experience in Multimedia Systems and Services, COST Action IC, 1003*, March 2012.

[43] A. Kassler, L. Skorin-Kapov, O. Dobrijevic, M. Matijasevic, and P. Dely, "Towards QoE-driven multimedia service negotiation and path optimization with software defined networking," in *Proceedings of the 20th International Conference on Software, Telecommunications and Computer Networks*, vol. 1, pp. 1–5, IEEE, September 2012.

[44] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race, "Towards network-wide QoE fairness using OpenFlow-assisted adaptive video streaming," in *Proceedings of the 2013 ACM SIGCOMM Workshop on Future Humancentric Multimedia Networking (FhMN '13)*, pp. 15–20, ACM, August 2013.

[45] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "A demonstration of fast failure recovery in software defined networking," in *Testbeds and Research Infrastructure. Development of Networks and Communities*, vol. 44, pp. 411–414, Springer, Berlin, Germany, 2012.

[46] P. Fonseca, R. Bennesby, E. Mota, and E. Passito, "A replication component for resilient OpenFlow-based networking," in *Proceedings of the 2012 IEEE Network Operations and Management Symposium*, pp. 933–939, IEEE, April 2012.

[47] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *ACM SIGCOMM Computer Communication Review*, vol. 42, pp. 473–478, ACM, 2012.

[48] OpenFlow Management and Configuration Protocol (OF-Config) v.1.1.1, March 2013.