2013 IEEE 37th Annual Computer Software and Applications Conference

A Mashup-based Approach for Virtual SDN Management

Oscar Mauricio Caicedo Rendon
Computer Networks Group
Institute of Informatics
University Federal do Rio Grande do Sul
Email: omcrendon@inf.ufrgs.br

Felipe Estrada-Solano
Telematics Engineering Group
Telematics Department
University of Cauca
Email: festradasolano@unicauca.edu.co

Lisandro Zambenedetti Granville
Computer Networks Group
Institute of Informatics
University Federal do Rio Grande do Sul
Email: granville@inf.ufrgs.br

***Abstract*—The Software Defined Networks paradigm aided by the Network Virtualization is a key driver to cope the Internet ossification. There are different proposals to deploy this paradigm, but there is not an integrated or standardized way for the management of networks built with such proposals. In this sense, the network management becomes too complex because multiple solutions must be used by Network Administrators to perform their tasks. In this paper, we introduce a mashup-based approach that allows Network Administrators to customize and combine management solutions, in order to they build composite applications (called SDN Mashups) aiming the integrated management of Virtual Software Defined Networks in heterogeneous environments. We evaluate our approach by building a SDN Mashup for the management of a network slice that uses three distinct Network Operating Systems and by running performance tests, corroborating that the mashup built has small response time.**

***Keywords*-OpenFlow; SDN; SDN Mashup; Virtual SDN; Web-based Management;**

I. Introduction

The Internet constantly evolves to support a lot of new technologies and protocols in Application and Link Layers. However, at the Internet core (Transport and Internet Layers), the evolution has come to a standstill that is known as Internet ossification. The Software Defined Networks (SDN) and the Network Virtualization are key drivers to overcome such ossification [1]. The SDN paradigm proposes to separate data (packet forwarding) and control (decision policies) planes in order to simplify the network operation [2]. The Network Virtualization allows to share a network physical infrastructure among several virtual networks. This type of virtualization may help to deploy the SDN-based networks, because it facilitates the control plane migration from network devices (*e.g.*, routers, switches) to servers and allows to perform network experiments in an isolated way [3]. Hereinafter, we will call a SDN aided by the Network Virtualization as Virtual SDN.

There are different proposals for deploying the SDN paradigm, such as OpenFlow [4] and the Forwarding and Control Element Separation (ForCES) framework [5]. In these proposals common components are: the Network Operating System (NOS) at the control plane and the Network Services running on it. However, there is not an integrated or standardized way for managing these components, which certainly is not suitable for the whole management of SDN-based networks on heterogeneous and virtual environments. For instance, in a future scenario, if a Network Administrator needs to manage several Virtual SDN Slices from different providers, which are using distinct NOS to operate and provide Virtual SDN Resources, the SDN management will become too complex because multiple tools must be used to perform control and monitoring tasks.

Although large research efforts have been made about the SDN deployment [2] [6] [7], few investigations are found in the literature concerning the control and monitoring of non-homogeneous SDNs. In this paper, we take a step further, proposing a novel mashup-based approach that provides a suitable model of composition and abstraction to cope the heterogeneity of virtual resources on SDN. In the approach, we introduce the SDN Mashup concept that lets Network Administrators create SDN Management solutions (called SDN Mashups) to meet their own requirements. The SDN Mashups stimulate Network Administrators to customize and combine, in a high-level abstraction, their SDN management tools, aiming to facilitate the enforcement of management tasks.

In summary, the key contributions presented in this paper are: (*i*) propose a mashup-based approach aimed to manage Virtual SDNs on heterogeneous environments and allow Network Administrators to build up SDN Management solutions, (*ii*) present a SDN Mashup prototype based on the representation of Virtual SDN Resources as Services; and (*iii*) demonstrate a monitoring scenario of a Virtual SDN that uses three different NOS, confirming the small response time of the mashup built.

The remainder of this paper is organized as follows. In Section II, we present both the background and the related work. In Section III, we introduce the SDN Mashup concept. In Section IV, we present the SDN Mashup System. In Section V, we expose and analyze the case study developed to evaluate our approach. The paper concludes in Section VI.

0730-3157/13 $26.00 © 2013 IEEE
DOI 10.1109/COMPSAC.2013.22

II. Background and Related Work

In this section, first, we present a mashups background. Second, we describe the main SDN concepts. Third, we discuss the related work about the SDN management.

A. Mashups

Mashups are Web applications created through the integration of different resources (*e.g.*, data, application logic, and user interfaces) available on the Internet [8]. The Mashup technology has been considered a fundamental piece in Web 2.0 [9], allowing end-users, without advanced programming skills, to create their own and customized applications. Furthermore, mashups encourage both cooperation and reuse among end-users [10].

In accordance to the ProgrammableWeb site, the main mashup service directory, about 60% of current mashups are related to mapping services [11]. The Mashup technology has been also used in many other areas, for instance, helping to overcome an emergency situation [12] by sharing weather and rescue information among civil organizations and government entities. In the telecommunications area, the telco-mashup concept [13] was defined to provide composite services for end-users, by combining features like streaming, Quality of Service, and billing. Mashups, based on the REpresentational State Transfer (REST) architectural model and the semantic Web, were proposed to facilitate the composition of small applications by end-users [14]. In this paper, we introduce the SDN Mashup concept to extend the use areas of mashups and cover the SDN management.

B. Software Defined Networks

The SDN paradigm has emerged as an important trend that defines how future networks are architected. A SDN is formed by three architectural components [2] [6]: the packet forwarding datapath (*e.g.*, switches and routers passing packets), the NOS that controls such datapath through a vendor-independent protocol, and the Network Services (or Network Features) running on the top of NOS. The possibility to add these Network Services, in an easier way, is the key advantage of SDN to facilitate the innovation in the Internet.

The SDN deployment proposals define aforementioned components in a different way. For instance, in the ForCES framework [5], the ForCES protocol is used to communicate Control Elements (*i.e.*, the NOS) and Forwarding Elements (*i.e.*, the datapath). In such framework, Network Services can be developed as distributed features in Control Elements. In an OpenFlow-based SDN [4], a Controller (*i.e.*, the NOS), such as POX [15], Beacon [16], and FloodLight [17], uses the OpenFlow protocol to control OpenFlow-capable network devices (*i.e.*, the datapath). The Controller is also used for deploying new-centralized Network Services (*e.g.*, a new routing protocol) that are known as Network Applications.

C. Management of Software Defined Networks

Although, in previous researches, the problems about the management of heterogeneous SDNs by using high-level tools have not been directly addressed. Below, we review some of the most important OpenFlow management solutions found in the literature.

The Stanford University introduced a graphical tool, called OpenRoads [18], to facilitate the management of IP addresses in OpenFlow networks and to show monitoring information of switches on the datapath. The OpenFlow MaNagement Infrastructure (OMNI) [19] is a solution aimed to control and monitor OpenFlow networks. This solution is based on a multi-agent system that can be accessed by Network Administrators from a Web user interface. The NetOpen [20] uses a Service Oriented Architecture (SOA) to support the creation of Network Services by combinig basic SOA services that are named networking primitives. The NetOpen considered, among others, the following Network Services: to retrieve information of switches, link states, and flow tables, and to configure the network device capabilities.

It is worth noting that the described solutions were not devised to be extended and enhanced by Network Administrators themselves. Such solutions can be solely improved by network programmers in a low-level abstraction. Moreover, up to now, OpenRoad and OMNI were just tested in network slices controlled by NOX that is an OpenFlow-based NOS implemented in the C language. In turn, NetOpen can be considered as a specialization of NOX. Consequently, OpenRoad, OMNI, and NetOpen cannot manage a Virtual SDN that uses more than one type of NOS. Thus, regarding the NOS, these solutions are constrained to homogeneous environments.

III. SDN Mashups

In order to better explain our approach, first, we present the global vision of SDN Mashups. Second, we describe a network management scenario in which is necessary to use such type of mashup.

A. Global Vision

Before defining what is a SDN Mashup, we present the main concepts used in our approach. A Virtual Network Provider (VNP) is a company in charge of operating Virtual SDN Resources and providing them to distinct Virtual Network Operators (VNOs). A VNO is a company responsible for supplying the Virtual SDN Slices requested by customers and/or applications [1]. A Virtual SDN is a subset of the underlying physical network and, usually, can be formed by several Virtual SDN Resources [3]. One or more Virtual SDN form a Virtual SDN Slice.

Every Virtual Network Element (VNE), Network APplication (NAP), and NOS is a Virtual SDN Resource. A VNE is located at the bottom of the SDN architecture. Virtual network devices (*e.g.*, the Vyatta Router and the

Open vSwitch), virtual nodes and hosts (using, for instance, VMWare, Xen, or VirtualBox), links, and flows are types of VNE. A NAP is a program that handles the control software of VNE, through interfaces and protocols provided by NOS. Rendezvouz services and applications to path selection are examples of NAP. A NOS is in charge of monitoring and handling the resources and the entire state of Virtual SDN.

The SDN Mashups are composite Web applications aimed to manage any SDN that has been deployed using Network Virtualization. In our approach, Network Administrators are able to create SDN Mashups by using wiring and drag-and-drop mechanisms. Thus, Network Administrators do not require intimate knowledge about the Application Program Interfaces (APIs) of NAP, NOS, and VNE, or concerning the data mapping among these APIs. It is important to highlight that an end-user programming approach, as the used by SDN Mashups, provides flexibility for Network Administrators to build their solutions by themselves, and promotes the innovation in SDN management solutions.

A SDN Mashup poses some features that existing mashups do not support. First, it combines information, on the fly, from multiple resources, such as NAP, NOS, and VNE. Second, it hides the heterogeneity and complexity of Virtual SDN Resources in order to facilitate the carrying out of management tasks. Third, it blends local and external visualization APIs to generate integrated Graphical User Interfaces (GUIs). Fourth, it provides access to multiple end-users to enable communication and collaboration among them by sharing and reusing SDN Mashups. It is worth noting that, by the SDN Mashup concept and the aforementioned features, we lead the Network Management towards an end-user centric environment, where millions of Network Administrators are able to participate and collaborate in order to cope their own needs, and even obtain profits.

In a general way, a SDN Mashup is formed by combining Virtual SDN Resources represented as Services, Mashup Operations, and GUIs. The representation of Resources as a service consists on defining and providing a common data-format to interchange information of resources, a well-known interface to each resource, and a common protocol to communicate with every interface. Specifically, we define the following Virtual SDN Resources as a service (see details in the section IV): Network Operating System as a service (NOSS), Network APplication as a service (NAPS), and Virtual Network Element as a service (VNES). The Mashup Operations are structures of composition, such as *Sequential*, *Split*, and *Merge*. A Mashup Operation can be used, for instance, to sort, filter, and aggregate the information retrieved from one or more NOSS. The GUIs represent visualization and presentation libraries used to generate the integrated user interfaces of SDN Mashups.

The Figure 1 presents the global vision of SDN Mashups, in which we mainly propose the creation of SDN management solutions by end-users: *(1)* The mediation process

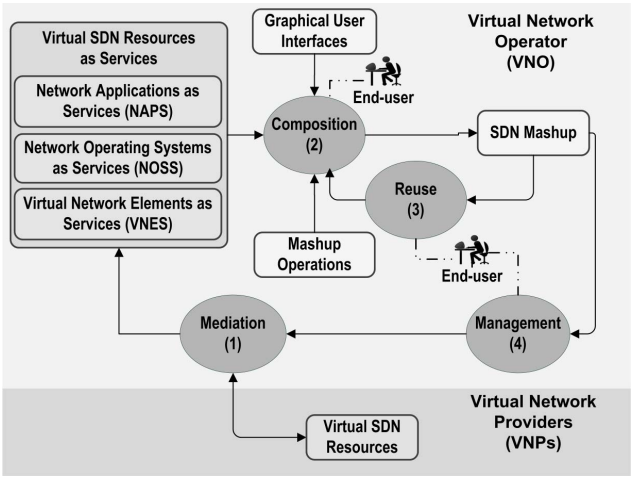

Figure 1. Global Vision of SDN Mashups

is responsible for offering the Virtual SDN Resources as Services. This mediation is necessary because there is not a standardized interface/protocol to access the data, the application logic, and the user interfaces provided by different types of resources. *(2)* The end-user (*e.g.*, a SDN Administrator: the Network Administrator of a VNO), in the composition process, defines the Mashup Operations that act on NOSS, NAPS, and/or VNES. The results of these Operations are shown by Web 2.0 GUIs, that are also defined and customized by the end-user in the composition process. *(3)* In the reuse process, a SDN Mashup can be used to create another one. Different end-users may use the same or similar candidate resources/services/mashups and glue them to compose a new complex SDN Mashup. *(4)* The end-user, by executing SDN Mashups, is able to manage one or several Virtual SDNs that are formed by Virtual SDN Resources belonging to VNPs. A SDN Mashup carries out their management tasks through the mediation process that is always hidden for the end-user.

B. Motivating Scenario

Management of Virtual SDN. Let's suppose that a Network Administrator, here called SDN Administrator, requires to purchase new Virtual SDN Slices to satisfy the demand of its customers, for instance, Internet Service Providers and small companies. Usually, VNP_a is the choosen option to meet such requirement. However, at this time, the SDN Administrator decides to buy required Slices from VNP_b because of economic profits. As a result, the SDN Administrator will need to control and monitor the Slices provided by VNP_a and VNP_b.

Considering that VNP_a uses a different NOS than VNP_b, the SDN Administrator will have to manage each type of Virtual SDN Slice by using disparate management solutions, such as proprietary command line interfaces to execute

specific commands on each NOS or dissimilar Web user interfaces to administrate virtual routers. Instead, if the SDN Administrator uses our approach, he/she will be able to build by him/herself a SDN Mashup devoted to manage the Virtual SDN Slices, in an integrated way. This SDN Mashup will hide the NOS heterogeneity from VNP_a and VNP_b. Thus, the complexity of SDN management tasks carried out by the SDN Administrator will be also mitigated.

IV. SDN Mashup System

Usually generic mashup systems (in the literature, they are also known as mashup makers) provide good basis for developing small composite applications, named mashups. However, these systems do not address, in a native way, special concerns of the SDN management. In particular, the complexity, heterogeneity, and high-level interaction of SDN Resources must be driven to enable the control and monitoring of SDN Slices in virtual environments. Therefore, there is a gap in the mashup-and-SDN related research and, consequently, there is a chance for innovation. In next paragraphs, we describe how a system based on the abstraction and composition models of the mashup technology, called SDN Mashup System, can be targeted to resolve the shortcomings of the SDN management in non-homogeneous and virtual surroundings.

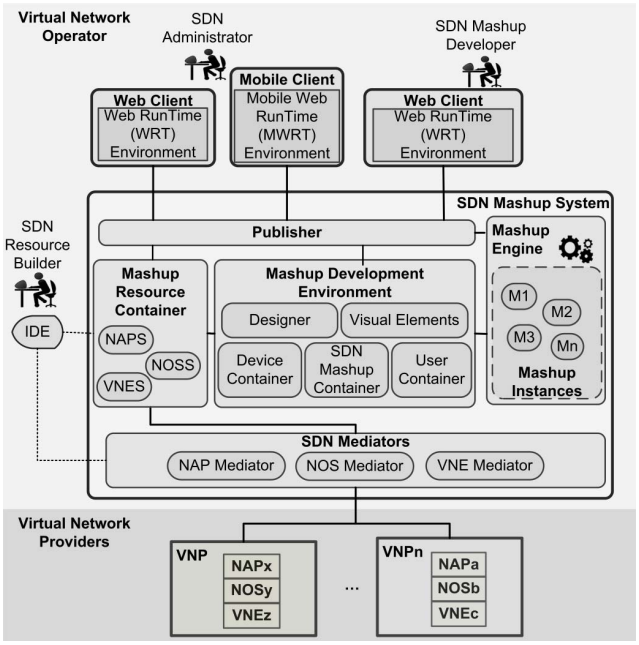

Figure 2. SDN Mashup System

The Figure 2 depicts the SDN Mashup System that enables to carry out the SDN Mashup concept, the System users, and the Virtual SDN Resources (*i.e.*, NAP, NOS, and VNE) to be managed by SDN Mashups. The SDN Mashup System is made up by the SDN Mediators, the Mashup Resource Container, the Mashup Development Environment, the Publisher, and the Mashup Engine. The users that interact with our System by using a Web Client, a Mobile Client, and/or an Integrated Development Environment (IDE) are the SDN Administrator, the SDN Mashup Developer, and the SDN Resource Builder.

The Virtual SDN Resources, provided by VNPs, are heterogeneous. Therefore, in the SDN Mashup System, these resources are accessed and handled through SDN Mediators. A Mediator hides the complexity of one or more resources in two ways: (*i*) accessing and retrieving the information from Virtual SDN Resources, and presenting it to the SDN Mashup System in a standardized data-format (*e.g*, XML and JSON); and (*ii*) providing a two-way communication between the Virtual SDN Resources and the SDN Mashup System via gateways (*e.g*, SNMP/HTTP and Proprietary/HTTP). This communication allows complete interaction among any VNO and its VNPs.

In the SDN Mashup System, SDN Mediators were defined for NAP, NOS, and VNE. A new Mediator must be developed every time a new kind of Virtual SDN Resource arises. In our approach, we propose that Mediators must be developed and extended by the SDN Resource Builder through the use of a conventional IDE. For example, if the NOX is integrated into a VNP, the SDN Resource Builder will be in charge of developing the corresponding Mediator (*e.g.*, NOX Mediator) to adapt such NOS into the SDN Mashup System.

The Mashup Resource Container stores services that represent the Virtual SDN Resources in the SDN Mashup System. We define three types of services: (*i*) NAPS that offers the functionalities provided by Network Applications (*e.g.*, a Video Multicasting solution) running on the top of a specific NOS, (*ii*) NOSS, in turn, provides the management facilities (*e.g.*, slice topology discovery) supplied by a determined NOS; and (*iii*) VNES that offers the information about one or a set of virtual network elements, for instance, quantity of sent/lost packets in a Vyatta virtual router. The communication between the Mashup Resource Container and every SDN Mediator is made via a standardized protocol (*e.g.*, HTTP and SOAP). NAPS, NOSS, and VNES interact with corresponding Virtual SDN Resources through SDN Mediators. Similarly to Mediators, services in the Resource Container must be implemented by the SDN Resource Builder.

In very general terms, SDN Administrators and SDN Mashup Developers use the Mashup Development Environment to compose and execute SDN Mashups (SDN management solutions). The Mashup Development Environment provides flexibility to the SDN Mashup System through a high-level abstraction of Virtual SDN Resources, GUIs, and Mashup Operations used in the composition process. In this sense, it is important to point out that we propose a Mashup Development Environment in which, during the

building of SDN Mashups, it is not necessary to work with data mapping. The data mapping is one of the least intuitive tasks in current mashup makers because non-programmers (as the SDN Administrators) are usually not able to specify it correctly.

The Mashup Development Environment is formed by the Visual Elements, the Designer, the SDN Mashup Container, the Device Container, and the User Container. The Visual Elements are graphical representations of the Mashup Operations, the services stored into the Mashup Resource Container, and the SDN Mashups. In addition to SDN Mashups, we define four types of Visual Elements: Visual_NAP, Visual_NOS, Visual_VNE, and Visual_MashupOperation. An instance of Visual_NAP is a box symbolizing a new tunneling algorithm to be executed on the top of a NOS. An example of Visual_NOS is a box representing a particular NOS as Beacon, NOX, FloodLight, or POX. A type of Visual_VNE is a box symbolizing a virtual switch as the Open vSwitch. A visual filter to be applied to the information collected from NOSS invocations is an example of Visual_MashupOperation.

The Designer is an user interface based on drag-and-drop and wiring mechanisms. Using these mechanisms, SDN Mashup Developers and SDN Administrators can blend, in an easy way, different Visual Elements to create SDN Mashups. By considering, the Visual Elements, the Mediators, and the Designer, the Mashup Development Environment becomes technology-agnostic. Here, technology-agnostic means that the Mashup Development Environment allows to combine Resources/Services regardless the corresponding underlying protocols (*e.g.*, OpenFlow/ForCES), controller libraries (*e.g.*, Beacon/POX API), and so on.

In the Designer, the SDN Mashups can be used to develop new and complex ones, which promotes: (*i*) the reuse of SDN Mashups, (*ii*) the extension and improvement of SDN Mashups and the Mashup Development Environment; and *(iii)* the fast development of SDN Mashups. In brief, the SDN Administrator and the SDN Mashup Developer can use the Designer to extend and enhance their SDN Management solutions and the own Designer. Likewise, the SDN Resource Builder can also improve the Mashup Development Environment (including the Designer) by adding new Visual Elements using an IDE.

The SDN Mashup Container stores the metadata of all SDN Mashups built in the Mashup Development Environment. This metadata is used on design time to present each SDN Mashup as a Visual Element. Thus, the SDN Mashup Container is also a key module that enables the reuse in our approach. On runtime the metadata is read to execute every SDN Mashup. The User Container stores the user profiles metadata, that is used to control the access to SDN Mashups. The Device Container hosts the information related to device capabilities. This information is processed to identify what type of Client device is able to run the SDN Mashups and the Mashup Development Environment.

The Publisher module is responsible for adapting the GUI of each SDN Mashup to different Client devices (*i.e.*, the Web Client and the Mobile Client). Moreover, this module controls the access to available elements in the containers of the SDN Mashup System. After that a SDN Mashup is launched, for example from the Mashup Development Environment, the Mashup Engine acts as the SDN Mashup life cycle manager in charge of creating, deleting, and caching Mashups Instances. As a result, this Engine interacts with all modules of the SDN Mashup System.

The Web Client and the Mobile Client are software entities in charge of running and showing SDN Mashups, anywhere and anytime. The former uses a Web RunTime environment and the latter a Mobile Web RunTime environment to execute client-side mashup functionalities. The SDN Mashups can be executed on both types of Clients. Therefore, browsers, running on personal computers, notebooks, and smartphones, are enabled to be used as front-end of SDN management solutions based on mashups. The Mashup Development Environment only can be executed on Web Clients, which means that SDN Mashups are programmed on Web and not on Mobile environments. Since SDN Mashups and the Mashup Development Environment are Web 2.0 solutions, Client devices must support Javascript, Asynchronous Javascript And XML (AJAX), Cascading Style Sheets (CSS), and HyperText Markup Language (HTML) version 5, among other Web 2.0 technologies.

Regarding the users of the SDN Mashup System, we consider: first, the SDN Resource Builder is an information technology developer responsible for programming and providing SDN Resources as Services, Visual Elements, and Mediators. The Resource Builder interacts with our Mashup System by means of a conventional IDE. Second, the SDN Mashup Developer is in charge of combining Visual Elements and even existing SDN Mashups in order to develop new mashups. The Mashup Developer interacts with our Mashup System via the Mashup Development Environment running on a Web Client. Third, the SDN Administrator is responsible for the management of virtual and heterogeneous SDNs through mashups running on the Web Client and/or the Mobile Client. Using the Mashup Development Environment, the SDN Administrator is also able to perform two actions: (*i*) create, customize, and improve composite applications addressed to manage Virtual SDNs; and (*ii*) as a result of developing mashups, extend, and enhance his/her workspace.

V. Case Study

In order to evaluate our approach, first, we created an infrastructure of OpenFlow-based Virtual SDNs. Second, we developed a SDN Mashup for monitoring such infrastructure. Third, we conducted experiments to measure the response time of the SDN Mashup built.

A. OpenFlow Virtual SDN

The Figure 3 presents the test environment of our case study. VNP_a has an OpenFlow-based network where virtual switches (Open vSwitches), links and flows are monitored via Beacon version 1.0.2. The Beacon is an OpenFlow Controller developed in the Java language. VNP_b has an OpenFlow-based network that is monitored by POX version 1.0.0. The POX is a Controller implemented in the Phyton language. VNP_c has an OpenFlow-based network that is monitored by FloodLight version 0.9.0. The FloodLight is a Controller developed in the Java language. All OpenFlow-based Virtual SDNs were deployed on Mininet version 1.0.0 [6] that, in turn, was executed on Oracle VM VirtualBox version 4.2.6. The Mininet is a software for emulating OpenFlow networks. Here, we use OpenFlow because of its commercial and research significance [4].

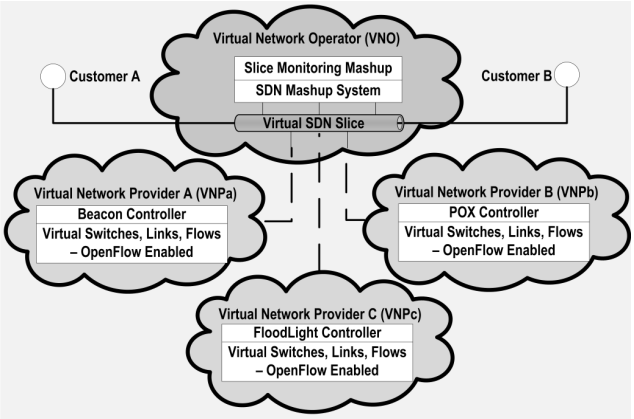

Figure 3. Test Environment

VNO provides network services to Customers A and B by means a Virtual SDN Slice made up of OpenFlow Controllers, virtual switches, links and flows from VNP_a, VNP_b, and VNP_c (see Figure 3). In this sense, the SDN Administrator of VNO requires to monitor Virtual SDN Resources, in an integrated way, regardless of controllers, network topologies, and implementation technologies. The previous requirement is met by the *Slice Monitoring Mashup* that is an instantiation of our SDN Mashup concept.

B. Slice Monitoring Mashup

The Slice Monitoring Mashup was composed in the Designer of the SDN Mashup System by connecting the VisualBeacon (*i.e.*, a Visual_NOS), the VisualFloodLight, and the VisualPOX to the MonitorSDN (*i.e.*, a Visual_MashupOperation). The VisualBeacon, the VisualPOX, and the VisualFloodLight are boxes built to represent, respectively, Beacon, POX, and FloodLight. In turn, the MonitorSDN is a visual box created to encapsulate the next monitoring operations: `SwitchesList`, `LinksList`, and `FlowsList`. These operations are applied to the VisualBeacon, the VisualPOX, and/or the VisualFloodLight to retrieve the list of virtual switches, links, and flows. The Slice Monitoring Information results from executing one or more of the above mentioned operations.

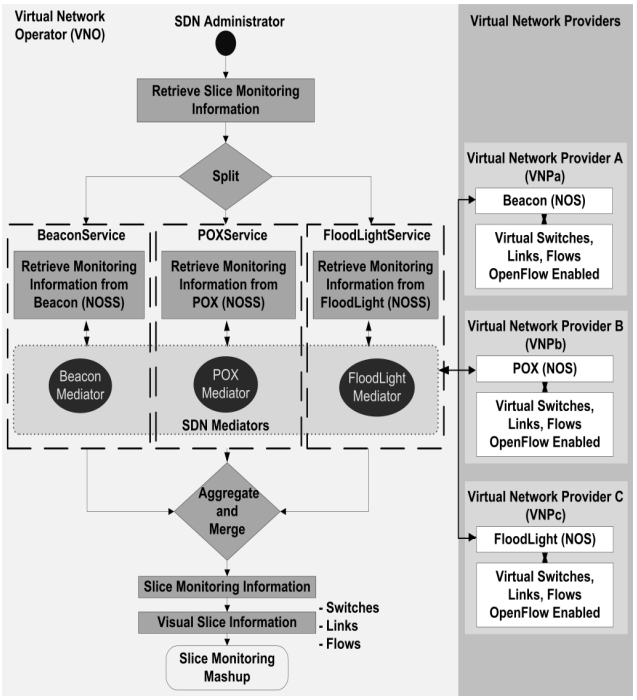

Figure 4. Internal Operation of Slice Monitoring Mashup

In a general way, the *Slice Monitoring Mashup* is in charge of splitting, aggregating, and merging the information collected from different Virtual SDN Resources in VNP_a, VNP_b, and VNP_c. The Figure 4 depicts the internal operation of the mashup developed to the raised case study: (*i*) in the Mashup Development Environment running on a Firefox Web Client, the SDN Administrator sends a request to execute the *Slice Monitoring Mashup*, (*ii*) this request is *Splitted* in three solicitations, the first solicitation targeted to the BeaconService, the second to the POXService, and the third to the FloodLightService, (*iii*) each Service invokes NOSS operations (`SwitchesList`, `LinksList`, and `FlowsList`) to collect information from a particular Controller, (*iv*) each Service carries out the corresponding mediation process to interact with its specific OpenFlow-based Virtual SDN, (*v*) the information retrieved by FloodLight/POX/Beacon Services is *Aggregated* and *Merged* to generate the Slice Monitoring Information; and (*vi*) finally, such information is shown, in an integrated way, in the Web GUI (see Figure 5) of the *Slice Monitoring Mashup*.

As result of composing and executing the *Slice Monitoring Mashup*, the SDN Administrator is able to observe, in an unique GUI, the detailed information of resources

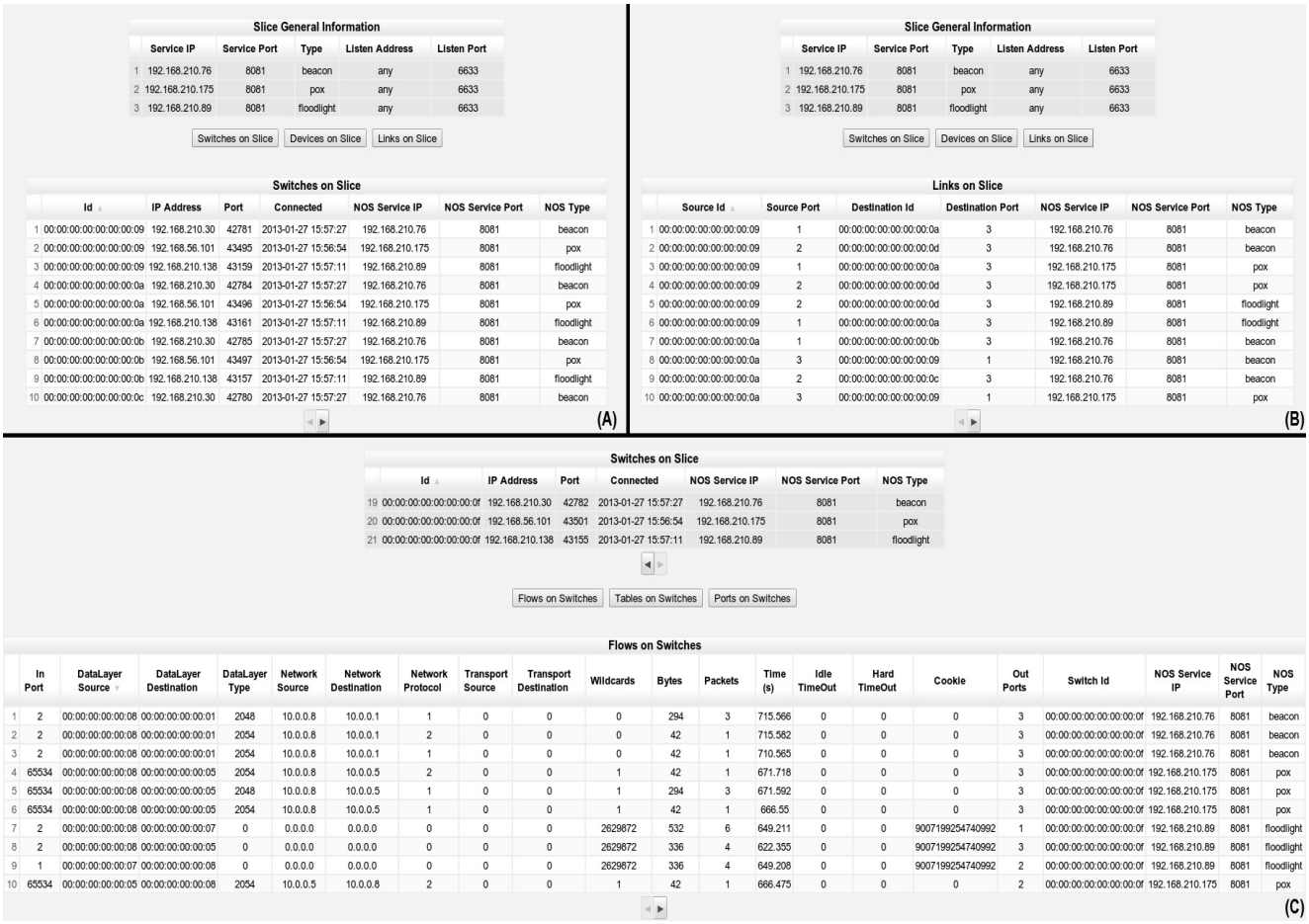

Figure 5. Slice Monitoring Mashup

forming the Virtual SDN Slice. For instance, (*i*) the Figure 5 (A) depicts the information about several virtual switches, monitored by either Beacon, POX, or FloodLight, (*ii*) the Figure 5 (B) presents details of links located in three OpenFlow-based SDNs; (*iii*) and the Figure 5 (C) illustrates flows in three virtual switches, each one monitored by a different OpenFlow Controller.

C. Implementation Highlights

Regarding the communication details, it is important to highlight that, first, the interaction between each Controller (Beacon, POX, and FloodLight) and its corresponding virtual switches (Open vSwitches) is made by the OpenFlow protocol. Second, the interaction between mediation processes and Controllers is based on Remote Procedure Calls (Beacon and POX do not expose their monitoring operations through interfaces of services) and HTTP (FloodLight exposes its monitoring operations as Web services). In this sense, the interactions BeaconController/BeaconService, POXController/POXService, and FloodLightController/FloodLightService were implemented by using the Java Remote Method Invocation API, the PYthon Remote Objects Library, and the FloodLight REST API, respectively.

The POXService, the BeaconService, and the FloodLightService are based on the REST architectural model [21]. We have used REST because it is becoming in the defacto model for developing mashups. Furthermore, REST-based solutions are suitable for heterogeneous environments (as in our case study) because their HTTP interaction is independent of programming languages. Specifically, the POXService was developed by using the Phyton Flask API. In turn, the BeaconService and the FloodLightService were implemented by using the Java Jersey API.

The GUIs of the *Slice Monitoring Mashup* and the Mashup Development Environment were built using the Yahoo User Interface (YUI) API, that provides a lot of high-level widgets, and the Google Chart API, that allows to create advanced graphics for websites. It is to point out that both APIs are based on AJAX. AJAX granted us two aspects: dynamic and interactive SDN Mashups, and asynchronous interaction of GUIs with POX/Beacon/FloodLightServices.

D. Evaluation and Analysis

The test environment (see Figure 3) was deployed on servers and personal computers running the Linux Ubuntu O.S. 11.10 (64 bits). The SDN Mashup System was executed on a server with 4 GBytes RAM and 2.53 GHz core 2 duo processor. Each virtual OpenFlow-based network, in a tree or linear topology, was deployed on a server with 8 GBytes RAM and 3.4 GHz core i7 processor. The *Slice Monitoring Mashup* was ran on a personal computer with 2 GBytes RAM and 2.53 GHz core 2 duo processor.

To test our approach, we evaluate the operations (`SwitchesList`, `LinksList`, and `FlowsList`) of the *Slice Monitoring Mashup* when it is used to monitor, in an integrated way, the Virtual SDN Slice composed by Virtual SDN Resources in VNP_a, VNP_b, and VNP_c. In all evaluation cases, we took 30 measurements with a 95% confidence level for the average response time in milliseconds.

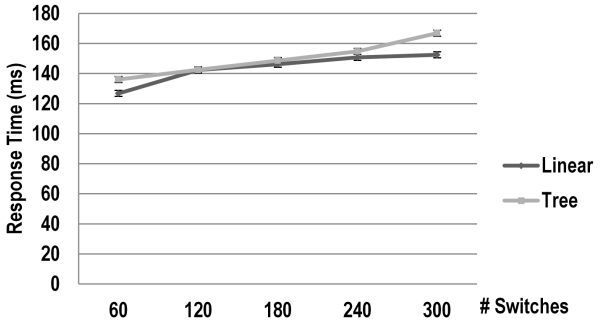

Figure 6. Slice Monitoring Mashup - SwitchesList

The Figure 6 depicts the response time results of the `SwitchesList` operation when the number of switches was increased from 20 to 100 in each Virtual SDN of VNP_a, VNP_b, and VNP_c. Thus, pertest, the total number of switches was 60, 120, 180, 240, and 300. Since the response time (r in milliseconds - ms) of Web systems can be ranked as optimal ($r \leq 100$), good ($100 < r \leq 1000$), admissible ($1000 < r \leq 10000$), and deficient ($r > 10000$) [22], we can state that the *Slice Monitoring Mashup* has a good r when executing the `SwitchesList` operation. Moreover, r has a similar behavior in tested topologies and its growth is less than 1 ms per switch.

The Figure 7 presents the response time results of the `LinksList` operation when the number of links was increased from 50 to 250 in each Virtual SDN of our case study. Thus, pertest, the total number of links was 150, 300, 450, 600, and 750. According to obtained results, we can assert that the `LinksList` operation of the *Slice Monitoring Mashup* has a good r that grows negligibly (less than 1 ms per link) when the number of links is raised in linear and tree topologies.

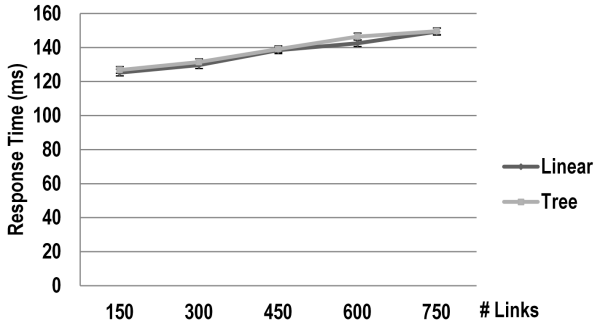

Figure 7. Slice Monitoring Mashup - LinksList

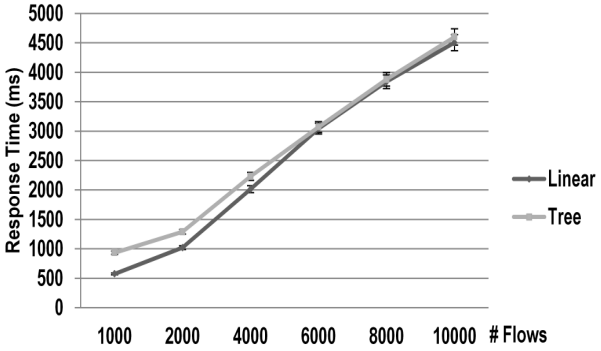

Figure 8. Slice Monitoring Mashup - FLowsList

The Figure 8 presents the response time results of the `FlowsList` operation when the number of flows is increased from 1000 to 10000 in the Virtual SDN Slice. For this operation, the *Slice Monitoring Mashup* has an admissible r that grows less than 3 ms per flow, regardless network topology. As in a network the number of flows may be large, in practice, `FlowsList` retrieves 1000 flows per block, getting so a good r. Such constraint is not relevant because the use of an unique GUI to display all flows is not a good practice of usability. Furthermore, using a mechanism of pagination, flows can be suitably retrieved and displayed to the SDN Administrator.

Summarizing, although the *Slice Monitoring Mashup* uses 11 additional software modules to integrate the monitoring information of the Virtual SDN Slice, the response time of all mashup operations is good on heterogeneous environments. In this way, we can state that the *Slice Monitoring Mashup* can be used to monitor a Virtual SDN Slice regardless of NOS, the network topology, and the number of virtual switches, links, and flows. The NOS heterogeneity is hidden by NOSS and SDN Mediators. The topologies and the number of virtual resources are handled by own centralized-nature of each NOS.

On the other hand, since the SDN Mashup System is based on visual mechanisms as dragging-and-dropping and

wiring, we can state that the SDN Administrator and the SDN Mashup Developer do not need programming skills to develop similar SDN Mashups to the *Slice Monitoring Mashup*. In the mashup composition process, the Administrator and the Developer only need to link Visual Elements and provide configuration information, such as IP address and type of NOS. Thus, if a SDN Administrator or a SDN Mashup Developer have to build SDN Mashups, he/she does not need to worry about the data mapping between Visual Elements.

VI. Conclusions and Future Work

In this paper, we introduced a mashup-based approach formed by the SDN Mashup concept and the SDN Mashup System that allows to carry it out. The concept and its instantiation are based on the abstraction and representation of any SDN Resource as a Service, and on the end-user centric development of composite applications. Thus, our approach empowers the SDN Administrator with the important ability to build, extend, and customize SDN management systems. We also presented a realistic Virtual SDN management scenario where multiple information sources and services are aggregated/merged to create a new application, namely the *Slice Monitoring Mashup* that is a SDN Mashup aimed to meet a specific purpose: integrated monitoring of a Slice formed by Virtual SDNs that use different NOS.

The aforementioned scenario has a particular challenge: the monitoring of heterogeneous Virtual SDNs. Our approach was able to overcome such challenge, corroborating the significance of the SDN Mashup concept and the SDN Mashup System. In this sense, through a quantitative evaluation, we have confirmed, first, a good response time ($r \leq 1000$) of the *Slice Monitoring Mashup*, regardless of network topologies and Virtual SDN Resources (NOS, virtual switches, links, flows). Second, the negligible growth of this response time as the number of Virtual SDN Resources increases. The evaluation of the *Slice Monitoring Mashup* confirms the feasibility of using our approach to cope the complexity and heterogeneity of the Virtual SDN management.

From a qualitative point of view, the use of Visual Elements, drag-and-drop, and wiring facilities, provides an easy-to-use Mashup Development Environment with little compromise on usability, particularly during the SDN Mashup composition process. The Visual Elements and the Mashup Designer allow to create and reuse SDN Mashups to manage, in an integrated manner, Virtual SDNs. Additionally, the SDN Mashup System, as a whole, hides implementation details about types of NOS, network topologies, virtual switches, flows, and links. Thus, our approach overcomes the stiffness of current SDN management solutions. In this sense, we consider SDN Mashups a step forward, in both the mashup technology and the network management area. We lead the former towards a new application domain and the latter to an environment centric in the Network Administrator.

As future researches, we plan to extend the SDN Mashup System, adding new features to perform other management tasks and appending more powerful GUIs to automatically compose SDN Mashups. Also, we are interested on evaluating the decrease on the carrying out time of SDN management tasks by using our mashup-based approach. The acceptance by Network Administrators of mashups as network management solutions is a topic that we are going to explore too.

Acknowledgment

The research of PhD(c) Caicedo is supported in part by a scholarship PECPG (Agreement Program Students Graduate) of the CAPES (Brazil). The University of Cauca (Colombia) also funds the research of PhD(c) Caicedo.

References

[1] N. Chowdhury and R. Boutaba, "Network Virtualization: State of the Art and Research Challenges," *Communications Magazine, IEEE*, vol. 47, no. 7, pp. 20–26, july 2009.

[2] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an Operating System for Networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008. [Online]. Available: http://doi.acm.org/10.1145/1384609.1384625

[3] A. Khan, A. Zugenmaier, D. Jurca, and W. Kellerer, "Network virtualization: a Hypervisor for the Internet?" *Communications Magazine, IEEE*, vol. 50, no. 1, pp. 136–143, january 2012.

[4] 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 Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, march 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[5] 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, march 2010. [Online]. Available: http://datatracker.ietf.org/doc/rfc5810/

[6] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-defined Networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. New York, NY, USA: ACM, 2010, pp. 19:1–19:6. [Online]. Available: http://doi.acm.org/10.1145/1868447.1868466

[7] A. Tootoonchian and Y. Ganjali, "HyperFlow: a Distributed Control Plane for OpenFlow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, ser. INM/WREN'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 3–3. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863133.1863136

[8] D. E. Simmen, M. Altinel, V. Markl, S. Padmanabhan, and A. Singh, "Damia: Data Mashups for Intranet Applications," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 2008, pp. 1171–1182. [Online]. Available: http://doi.acm.org/10.1145/1376616.1376734

[9] C. Cappiello, F. Daniel, M. Matera, and C. Pautasso, "Information Quality in Mashups," *Internet Computing, IEEE*, vol. 14, no. 4, pp. 14–22, july-august 2010.

[10] J. Yu, B. Benatallah, F. Casati, and F. Daniel, "Understanding Mashup Development," *Internet Computing, IEEE*, vol. 12, no. 5, pp. 44–52, september-october 2008.

[11] J. J. Jung, "Collaborative browsing system based on semantic mashup with open apis," *Expert Syst. Appl.*, vol. 39, no. 8, pp. 6897–6902, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2012.01.006

[12] A. Majchrzak and P. H. B. More, "Emergency! Web 2.0 to the Rescue!" *Commun. ACM*, vol. 54, pp. 125–132, April 2011. [Online]. Available: http://doi.acm.org/10.1145/1924421.1924449

[13] H. Gebhardt, M. Gaedke, F. Daniel, S. Soi, F. Casati, C. Iglesias, and S. Wilson, "From Mashups to Telco Mashups: A Survey," *Internet Computing, IEEE*, vol. 16, no. 3, pp. 70 –76, may-june 2012.

[14] A. P. Sheth, K. Gomadam, and J. Lathem, "SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups," *IEEE Internet Computing*, vol. 11, pp. 91–94, 2007.

[15] P. Community. (2012) POX Home. [Accessed july 20, 2012]. [Online]. Available: https://github.com/noxrepo/pox

[16] D. Erickson. (2012) Beacon Home. [Accessed july 20, 2012]. [Online]. Available: https://openflow.stanford.edu/display/Beacon/Home

[17] F. Community. (2011) Floodlight Home. [Accessed july 20, 2012]. [Online]. Available: http://floodlight.openflowhub.org/

[18] K.-K. Yap, M. Kobayashi, D. Underhill, S. Seetharaman, P. Kazemian, and N. McKeown, "The Stanford OpenRoads Deployment," in *Proceedings of the 4th ACM international workshop on Experimental evaluation and characterization*. New York, NY, USA: ACM, 2009, pp. 59–66. [Online]. Available: http://doi.acm.org/10.1145/1614293.1614304

[19] D. Mattos, N. Fernandes, V. da Costa, L. Cardoso, M. Campista, L. Costa, and O. Duarte, "OMNI: OpenFlow MaNagement Infrastructure," in *Network of the Future (NOF), 2011 International Conference on the*, november 2011, pp. 52 –56.

[20] N. Kim and J. Kim, "Building NetOpen Networking Services over OpenFlow-based Programmable Networks," in *Information Networking (ICOIN), International Conference on*, jan. 2011, pp. 525 –529.

[21] R. T. Fielding and R. N. Taylor, "Principled Design of the Modern Web Architecture," *ACM Transactions on Internet Technology*, vol. 2, no. 2, pp. 115–150, may 2002. [Online]. Available: http://doi.acm.org/10.1145/514183.514185

[22] S. Joines, R. Willenborg, and K. Hygh, *Performance Analysis for Java Websites*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.