SOFTWARE-DEFINED NETWORKING MANAGEMENT BASED ON WEB 2.0 AND WEB 3.0 TECHNOLOGIES

CARLOS FELIPE ESTRADA SOLANO

Thesis of Master of Science in Telematics Engineering

Advisor:
Oscar Mauricio Caicedo Rendon
Ph.D. in Computer Science

University of Cauca
Faculty of Electronic and Telecommunications Engineering
Telematics Department
Line of Research in Advanced Services of Telecommunications
Popayan, January 2016

CARLOS FELIPE ESTRADA SOLANO

SOFTWARE-DEFINED NETWORKING MANAGEMENT BASED ON WEB 2.0 AND WEB 3.0 TECHNOLOGIES

A thesis submitted to the Faculty of Electronic
and Telecommunications Engineering of the
University of Cauca for the Degree of

Master of Science in:
Telematics Engineering

Advisor:
Oscar Mauricio Caicedo Rendon
Ph.D. in Computer Science

Popayan

2016

Contains a copy of the defense record signed by the advisor and the evaluators.

To my parents, Tina Adriana and Luis Carlos, whose dedication and sacrifice have been the base for my progress.

To my sweetheart, Silvana, whose unconditional love and support have given me the strength to work every day.

To my furry pets, Lucas and Chata, whose follies have reduced the stress and bad mood caused by research.

To the memory of my grandpa, Ismael, whose legacy have traced an education and honesty path.

"It does not matter how slowly you go as long as you do not stop"
Confucius (551–479 BC)

Acknowledgements

Special thanks to the distinguished faculty member, Professor Oscar Mauricio Caicedo Rendon. As my advisor, Ph.D. Caicedo shared a detailed guidance and encouragement throughout the course of preparing for and conducting the research. His belief that it was, indeed, possible to finish kept me going. Certainly, today I am a better researcher because his influence.

My thanks to all the members of the Computer Networks Group, Institute of Informatics, Federal University of Rio Grande do Sul, Brazil, for their support, patience, encouragement, and useful suggestions during and after my research internship. Particularly, I want to thank Professor Lisandro Zambenedetti Granville, head of such important research group, for his acceptance and insightful comments about my investigation.

I am also grateful to the professors who served as evaluators of my work: Ph.D. Gustavo Adolfo Ramirez, M.Sc. Andres Lara, and M.Sc. Eva Maya from the University of Cauca, and Ph.D. Juan Felipe Botero from the University of Antioquia. They used their valuable time and expertise to provide helpful comments on short notice and quickly continue with the review process of my research.

Thanks to Colciencias and the University of Cauca for their financial support under Ph.D. scholarship 647-2014.

And special thanks to my family for their good-natured forebearance with the process and for their pride in this accomplishment. It was a team effort.

Structured Abstract

Background. The Software-Defined Networking paradigm establishes a three-plane architecture that facilitates the deployment of network functions and simplifies traditional network management tasks. However, this architecture lacks an integrated or standardized framework for managing the virtual, dynamic, and heterogeneous Software-Defined Networking environment itself. Some investigations have addressed such shortage by proposing different solutions that tackle specific management requirements for particular technology instances. This isolated approach forces network administrators to use multiple frameworks or to build their integrated tools for managing the whole Software-Defined Networking environment. The diversity, the continuous updating, and the restricted reuse and sharing of management solutions increase the complexity and time to manage networks based on the Software-Defined Networking paradigm. Therefore, this thesis focuses on investigating an effective approach (i.e., in terms of time and network traffic) for managing the virtual, dynamic, and heterogeneous environment of Software-Defined Networks.

Goal. Propose a mechanism based on Web technologies (2.0/3.0) for carrying out the management of virtual, dynamic, and heterogeneous networks based on the Software-Defined Networking paradigm.

Methods. This thesis proposes a Management Plane as an effective approach for managing virtual, dynamic, and heterogeneous networks based on the Software-Defined Networking paradigm. The proposed Management Plane defines a reference architecture based on the Open System Interconnection management model to achieve a proper integrated framework. In addition, such Management Plane introduces an Information Model that provides a technology-independent and consistent abstraction of the whole

Software-Defined Networking environment across distinct vendors and instances. Finally, this Management Plane presents a framework based on Web technologies (2.0/3.0) to support the simple, cooperative, and dynamic creation of management tools for Software-Defined Networks.

Results. A Management Plane reference architecture for integrated management of Software-Defined Networks. Such Management Plane details an Information Model that relies on the Common Information Model to characterize the entire Software-Defined Networking environment from a management perspective. Furthermore, this Management Plane proposes a mashup-based and event-driven framework to encourage network administrators to dynamically customize, in a high-level abstraction, tools for managing networks based on the Software-Defined Networking paradigm.

Conclusions. The different case studies raised to evaluate and analyze this thesis corroborate the effectivity, in terms of time and network traffic, of the proposed approach for carrying out management tasks in realistic scenarios based on the virtual, dynamic, and heterogeneous environment of Software-Defined Networks. Both the Information Model and the Web-based framework reduce the time for dynamically managing a network deployed with virtual and heterogeneous technologies. In addition, both aforementioned solutions demonstrate good behavior on the response time as well as on the network traffic. As future research, there is still the need to extend the proposed Management Plane in order to afford a complete, well-defined Software-Defined Networking management architecture. Furthermore, there is an opportunity to investigate the feasibility of using Big Data technologies to capture, process, and analyze the enormous sets of information generated by Software-Defined Networks.

Keywords: Software-Defined Networking, Integrated network management, Common Information Model, Web-based network management, Mashups, Event-driven.

Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

AJAX	Asynchronous Javascript And XML
API	Application Programming Interface
AS	Autonomous Systems
BGP	Border Gateway Protocol
CE	Control Element
CEP	Complex Event Processing
CIM	Common Information Model
CLI	Command Line Interface
CSS	Cascading Style Sheets
DRL	Drools Rule Language
EDA	Event-Driven Architecture
EWBI	East/Westbound Interface
FCAPS+P	Fault, Configuration, Accounting, Performance, Security, and Programming
FE	Forwarding Elements
ForCES	Forwarding and Control Element Separation
GDMO	Guidelines for the Definition of Managed Objects
GUI	Graphical User Interface
HTML	HyperText Markup Language

HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
IP	Internet Protocol
JRMP	Java Remote Method Protocol
JSON	JavaScript Object Notation
KLM	Keystroke-Level Model
LFB	Logical Functional Block
LPM	Longest Prefix Match
M&E	Mashup-based and Event-driven
MOF	Managed Object Format
MRTG	Multi Router Traffic Grapher
MWRE	Mobile Web Runtime Environment
NAT	Network Address Translation
NBI	Northbound Interfaces
NetApp	Network Application
NetDev	Network Device
NetSlicer	Network Slicer
NOS	Network Operative System
OAM	Operation, Administration and Maintenance
ONF	Open Network Foundation
OSI	Open System Interconnection
POF	Protocol-Oblivious Forwarding
REST	Representational State Transfer

RMI	Remote Method Invocation
RRDtool	Round Robin Database tool
SBI	Southbound Interface
SDN	Software-Defined Networking
SME	Small and Medium Enterprise
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
UML	Unified Modeling Language
URI	Uniform Resource Identifier
VNP	Virtual Network Providers
WRE	Web Runtime Environment
WWW	World Wide Web
XML	eXtensible Markup Language
YANC	Yet Another Network Controller
YUI	Yahoo User Interface

Chapter 1

Introduction

1.1 Problem statement

The Internet has continually evolved to support a lot of new technologies and protocols in the Application and Link layers of the TCP/IP Model[1]. However, at the Transport and Network Layers (the core of the Internet) the evolution has come to a standstill that is known as the Internet Ossification. The Internet Ossification states that the current Internet architecture hinders network management, provides limited experimentation in real conditions, presents brittle scalability and security, and lacks of stimulus for network providers to invest and implement new technology [1, 2].

Over the past 20 years, the programmable networks and the network virtualization have spread as key technologies to cope with the Internet Ossification [3]. Nowadays, the Software-Defined Networking (SDN) paradigm is an attractive and accepted trend to program networks in both research and industry [4]. From a high-level point of view, SDN separates Control (*i.e.,* decision policies) and Data (*i.e.,* packet forwarding) planes, allowing to operate networks in a simpler way from a logically centralized software program [5]. The network virtualization aims to build open and flexible architectures, enabling several logical networks to share a single physical infrastructure [6, 7]. SDN and the network virtualization are not mutually inclusive, but their symbiosis promotes deploying, testing, and evaluating both technologies [8, 9]. For example, the network virtualization facilitates to migrate the Control Plane from network devices

[1]Model that describes the Internet protocol suite, commonly known as TCP/IP in reference to the two most important protocols: the Transmission Control Protocol (TCP) and the Internet Protocol (IP). It is also called as the Internet Model—http://tools.ietf.org/html/rfc1122.

(*e.g.*, switches and routers) to servers and allows to perform multiple experiments in an isolated way on a same underlying network.

Standardization bodies [10, 11], networking vendors [12, 13], and research surveys [14, 15] describe a typical SDN architecture, composed of three horizontal planes (*i.e.,* Data, Control, and Application) and three interfaces (*i.e.,* Southbound, Northbound, and East/Westbound), that defines functional and communication requirements to deploy SDN-based networks. However, this SDN architecture lacks an integrated or standardized framework for managing the virtual[2], dynamic[3], and heterogeneous[4] environment of SDN itself. Recent proposals have considered a Management Plane in the SDN architecture for carrying out Operation, Administration, and Maintenance (OAM) functions [16, 17]. Still, these approaches just expose a very high-level view of their management component.

At the top of the SDN architecture, a lot of research has proposed services and applications to simplify traditional network management tasks, such as load-balancing [18], efficient energy usage [19], and access control [20, 21]. Furthermore, some studies have addressed the need to manage SDN itself, for example, frameworks to configure the Data Plane [22, 23], to deploy [24, 25] and monitor [26, 27] the Control Plane, to virtualize SDNs [28, 29], and to develop the Application Plane [30, 31]. Nevertheless, no integrated solution exists to manage SDN as a whole by employing well-defined interfaces and supporting different deployed technologies. As a fundamental step, an integrated solution needs a joint understanding that entirely characterizes the virtual, dynamic, and heterogeneous environment of SDN from the management perspective. Few approaches have addressed the formal representation of SDN [22, 32, 33]; still they are focused exclusively on specific technologies or fall short in modeling the SDN environment. This hinders and restricts reuse and sharing of SDN management solutions for different technology instances and domains.

The lack of frameworks for integrated network management forces network administrators to handle several isolated solutions to manage resources from distinct planes of the SDN architecture as well as various technology instances. Thus, SDN management remains complex and time-consuming because of the diversity of solutions. Alternatively, although network administrators do not have deep programming skills,

[2]Capability for sharing network resources from a same physical infrastructure among several virtual network instances.

[3]Flexibility for adding, modifying, migrating and removing network resources.

[4]Independence of the technology deployed by network resources.

they build their SDN management tools (usually, fragile and static scripts[5]) by integrating and interoperating multiple isolated solutions. However, the continuous updating (*i.e.*, arising and perishing) and the restricted reuse and sharing of SDN management solutions increase the complexity and time to build and maintain such management tools. For example, a network administrator usually must modify a script for reusing it in another network domain.

Summarizing, due to the integration and interoperation needs, the continuous arising and perishing, and the restricted sharing and reusing of SDN management solutions, managing the SDN environment as a whole remains complex and time-consuming. Thus, this thesis focuses on investigating an effective approach (*i.e.*, in terms of time and network traffic) for managing the virtual, dynamic, and heterogeneous environment of SDN. The following research question defines the aforementioned problem:

How to carry out an effective approach for managing virtual, dynamic, and heterogeneous networks based on the SDN paradigm?

1.2 Hypothesis

To address the above research question, this thesis raises the following hypothesis: **a Management Plane that establishes a common Information Model and employs Web technologies (2.0/3.0) would provide an effective approach for managing virtual, dynamic, and heterogeneous networks based on the SDN paradigm.**

This hypothesis focuses on the need of a Management Plane that facilitates integrated control and monitoring of SDN. The Open System Interconnection (OSI) management model [34] affords a proper integrated framework to define a reference architecture for this Management Plane. In such Management Plane, a common Information Model would provide a technology-independent and consistent abstraction of SDN across distinct vendors and instances, encouraging integrated management of the virtual, dynamic, and heterogeneous environment of SDN. Furthermore, employing Web technologies (2.0/3.0) to deploy this Management Plane would enable to build platforms that support the simple, cooperative, and dynamic creation of management tools. Thus, the Management Plane would introduce an effective approach for managing the virtual, dynamic, and heterogeneous SDN environment as a whole.

[5]Set of instructions commonly stored in a text file, interpreted line by line in real-time for its execution.

The below fundamental questions, associated with the afore raised hypothesis, guide the investigation conducted in this thesis.

- What is the performance, in terms of time and network traffic, of solutions based on a technology-agnostic and consistent Information Model for managing virtual, dynamic, and heterogeneous networks based on the SDN paradigm?
- What is the performance, in terms of time and network traffic, of solutions that use Web 2.0 and Web 3.0 technologies for managing virtual, dynamic, and heterogeneous networks based on the SDN paradigm?

1.3 Goals

1.3.1 Main goal

Propose a mechanism based on Web technologies (2.0/3.0) for carrying out the management of virtual, dynamic, and heterogeneous networks based on the SDN paradigm.

1.3.2 Specific goals

1. Define a formal data and interaction model for describing network management in the SDN environment.
2. Design a reference architecture[6] supported by Web technologies (2.0/3.0) for managing virtual, dynamic, and heterogeneous networks based on the SDN paradigm.
3. Present and evaluate the architecture and the formal data model through an SDN management case study in an emulated environment of a virtual, dynamic, and heterogeneous network.

1.4 Contributions

The investigation about the feasibility of deploying a Management Plane that establishes a common Information Model and employs Web technologies (2.0/3.0) for

[6]The term *reference architecture* indicates an architecture in a high-level of abstraction.

effectively managing virtual, dynamic, and heterogeneous networks based on the SDN paradigm, led to the following major contributions.

- A Management Plane that employs the OSI network management submodels (*i.e.*, Information, Organizational, Communication, and Functional) to define a reference architecture for integrated management of the virtual, dynamic, and heterogeneous SDN environment.
- A technology-agnostic and consistent Information Model that characterizes the SDN management environment as a Common Information Model (CIM) conceptual framework.
- A mashup-based concept (*i.e.*, SDN Mashup) that encourages network administrators to build and customize, in a high-level abstraction, management tools in the SDN environment.
- A mashup-based and event-driven framework that incorporates the carrying out of static and dynamic SDN Mashups into the Manager component of the Management Plane.

The above-mentioned contributions were reported to the scientific community through paper submissions to renowned conferences and journals (see Appendix A).

- A paper submitted to the journal Computer Communications. Colciencias index: A1. Contribution: the reference architecture for the Management Plane and the CIM-based Information Model.
- A paper published in the journal Computer Networks. Colciencias index: A1. Contribution: the mashup-based and event-driven framework.
- A paper published in the proceedings of The 28th IEEE International Conference on Advanced Information Networking and Applications (AINA) 2014. Contribution: a case study for the SDN Mashup.
- A paper published in the proceedings of The IEEE Global Communications Conference (GLOBECOM) 2013. Contribution: a case study for the SDN Mashup.
- A paper published in the proceedings of The 37th IEEE Annual International Computer Software & Applications Conference (COMPSAC) 2013. Contribution: the SDN Mashup concept.

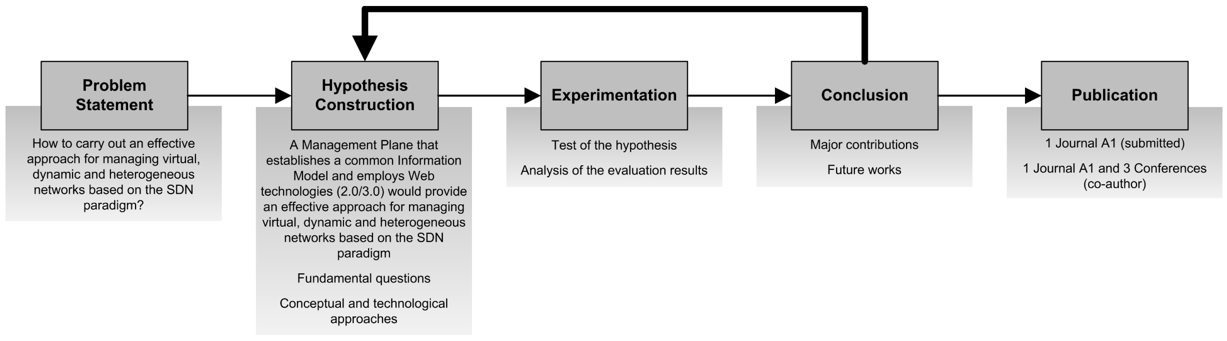

Problem Statement
How to carry out an effective approach for managing virtual, dynamic and heterogeneous networks based on the SDN paradigm?
Hypothesis Construction
A Management Plane that establishes a common Information Model and employs Web technologies (2.0/3.0) would provide an effective approach for managing virtual, dynamic and heterogeneous networks based on the SDN paradigm
Fundamental questions
Conceptual and technological approaches
Experimentation
Test of the hypothesis
Analysis of the evaluation results
Conclusion
Major contributions
Future works
Publication
1 Journal A1 (submitted)
1 Journal A1 and 3 Conferences (co-author)

Figure 1.1. Thesis phases

1.5 Methodology and organization

Figure 1.1 depicts the phases of the scientific research process followed in this thesis: Problem Statement, Hypothesis Construction, Experimentation, Conclusion, and Publication. Problem Statement, for identifying and establishing the research question. Hypothesis Construction, for formulating the hypothesis and the associated fundamental questions. In addition, this phase aimed to define and carry out the conceptual and technological approaches. Experimentation, for testing the hypothesis and analyzing the evaluation results. Conclusion, for outlining conclusions and future works. Note that Hypothesis Construction had feedback from Experimentation and Conclusion. Publication, for submitting and publishing papers for renowned conferences and journals. The writing of this document also belongs to this last phase.

The organization of this document reflects the phases outlined above.

- **This introductory chapter** presents the problem statement, raises the hypothesis, exposes the goals, summarizes the contributions, and describes the overall structure of this thesis.

- **Chapter 2** reviews research about SDN architecture, SDN management solutions, and Web technologies (2.0/3.0) for network management.

- **Chapter 3** details the major contributions accomplished with this thesis. Section 3.1 describes the Management Plane. Section 3.2 introduces the CIM-based Information Model. Section 3.3 depicts the mashup-based and event-driven framework.

- **Chapter 4** describes the experiments conducted to test the hypothesis, discusses the corresponding results, and presents implementations highlights.

- **Chapter 5** presents conclusions about the hypothesis and the fundamental questions as well as opportunities for future works.

Chapter 2

State of the art

This chapter presents the background of the main research topics encompassed in this thesis. In this way, the first section starts introducing a bottom-up description of the typical SDN architecture. Next, it reviews the more relevant Web 2.0 and Web 3.0 technologies. Finally, this chapter discusses some research works focused on addressing SDN management requirements and on using Web technologies (2.0/3.0) for network management.

2.1 Background

2.1.1 SDN

The SDN paradigm has emerged as an important trend that defines how future networks are architected [4]. Multiple standardization bodies, such as Linux Foundation [35] and Open Network Foundation (ONF) [11], focus on encouraging and normalizing open SDN frameworks. Also, various private networking vendors, such as Cisco [12] and Juniper [13], offer proprietary SDN deployments. In turn, several research surveys [14, 15] work on improving architectural aspects of SDN. These open, proprietary and research proposals establish a typical SDN architecture composed of three horizontal planes (*i.e.*, Data, Control, and Application) and three interfaces (*i.e.*, Southbound, Northbound, and East/Westbound), as depicted in Figure 2.1.

At the bottom of the SDN architecture, the Data Plane deploys the network infrastructure composed by interconnected Network Devices (NetDev) that perform forwarding operations. A NetDev consists of a physical part and a functional part. The for-

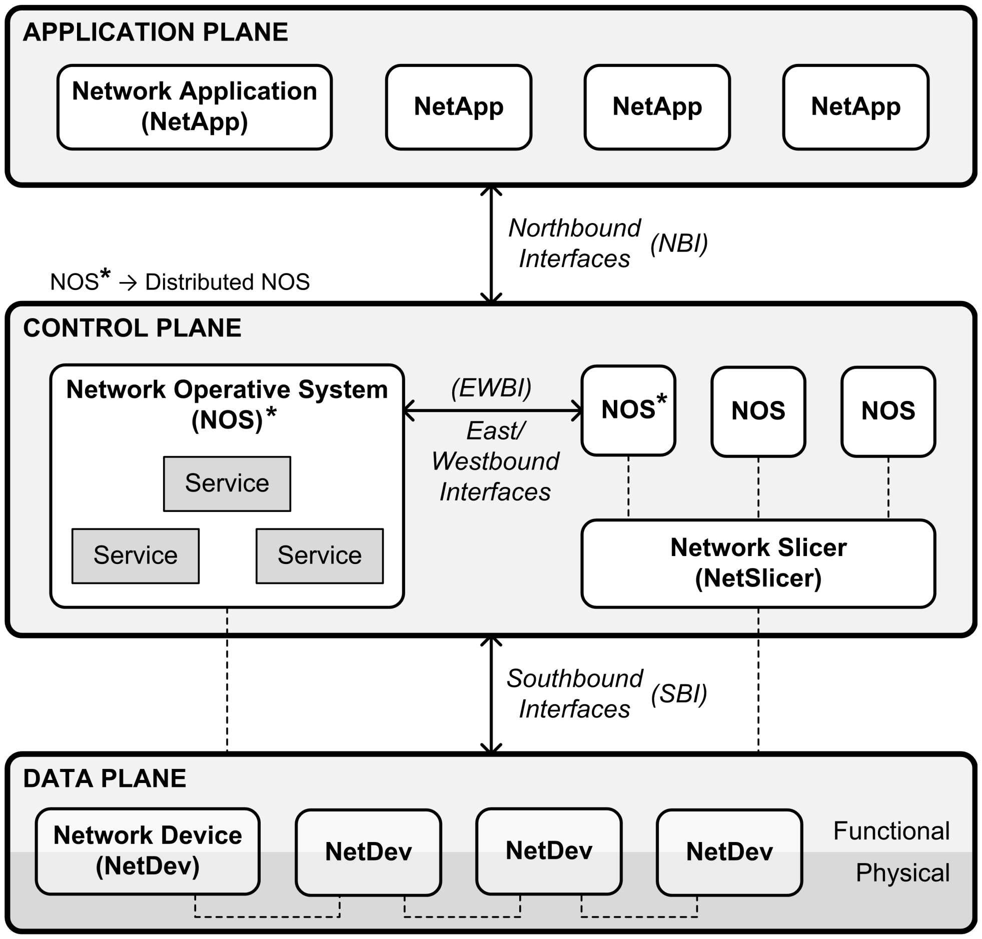


Figure 2.1. High-level SDN architecture

mer comprises hardware elements, such as ports, storage, processor, and memory. The latter defines a collection of software-based forwarding functions executed by NetDevs. Regarding the functional part, a NetDev ranges from dumb switches to custom switches. A dumb switch merely carries out simple forwarding functions, such as Longest Prefix Match (LPM). For example, OpenFlow-Only switches [36] just forward packets regarding their flow tables that are updated by the Control Plane. A custom switch relies on programmable platforms (*e.g.*, OpenWrt and NetFPGA) to integrate more complex forwarding functions, such as Network Address Translation (NAT) and firewall. For example, Forwarding Elements (FE) in ForCES [37] include multiple associated Logical Functional Blocks (LFB) to carry out such forwarding functions. An LFB defines either a punctual action for handling packets or a configuration entity operated by the Control Plane.

In the middle, the Control Plane compiles the network logic and enforces decision policies on the Data Plane through Southbound Interfaces (SBI). Each SBI defines the set of instructions and the communication protocols to allow the interaction between components in the Control Plane and in the Data Plane. The OpenFlow protocol [38] is the most well-known open standard SBI because its widespread use by vendors and research [4]. Other SBI proposals are ForCES [37] and Protocol-Oblivious Forwarding (POF) [39].

The Control Plane comprises Network Slicers (NetSlicer) and Network Operative Systems (NOS). A NetSlicer divides the underlying network infrastructure into several isolated logical network instances (*a.k.a.* slices), assigning their control to specific NOSs. NetSlicers may employ SBIs to communicate with NOSs. For example, FlowVisor [40] acts as an OpenFlow proxy between switches and controllers, redirecting messages according to flow parameters, such as TCP ports and IP addresses. An NOS instructs the underlying Data Plane and provides generic services (*e.g.*, topology discovering and host tracking) and Northbound Interfaces (NBI) to the Application Plane, facilitating to integrate custom Network Applications (NetApp). The possibility to add these NetApps in an easier way is the key advantage of SDN to encourage innovation on the Internet. OpenFlow Controllers [36] and ForCES Control Elements (CE) [37] are NOS instances. It is important to highlight that a lot of frameworks exist to develop and deploy OpenFlow Controllers, including open source projects like NOX for C++ [41], POX for Python [42], Floodlight [43] and OpenDaylight [10] for Java, and Trema [44] for Ruby. Also, the Control Plane defines East/Westbound Interfaces (EWBI) to deploy distributed NOSs. For example, SDNi [45] and ForCES CE-CE interface [37].

At the top of the SDN architecture, the Application Plane contains NetApps that deploy and orchestrate business logic and high-level network functions, such as routing policies and access control. As aforementioned, NetApps communicate with the Control Plane through NBIs provided by NOSs. NBIs encompass common Application Programming Interfaces (API) based on protocols (*e.g.*, Floodlight REST API [46]), programming languages (*e.g.*, ad-hoc, Pyretic [30], and Procera [31]), file systems (*e.g.*, YANC [47]), among others. NetApps run either locally or remotely regarding NOSs. Local NetApps prefer NBIs based on programming languages. Remote NetApps usually employ protocol-based APIs.

2.1.2 Web 2.0

Web 2.0 does not define a specific technical update, but a different way how people interact with the World Wide Web (WWW). Web 2.0 exposes a platform constituted by a new kind of technologies that encourage conventional Web users to actively develop services and dynamically create and modify contents. This maximizes collective intelligence by means of a cooperative and a participative architecture [48, 49].

For the development of this thesis, the **mashup** technology excels as a fundamental Web 2.0 solution. Mashups are composite Web applications centered on end-users and created by combining resources available along the Web, such as data, application logic, and user interfaces [50]. End-user centric means that conventional users with poor programming skills may develop mashups [51]. This Web 2.0 technology deploys a simple composition model that provides easy development and flexible execution of mashup-based applications, encouraging cooperation among end-users and reuse of existing Web resources [52, 53].

2.1.3 Web 3.0

Web 3.0 defines a data and link structure that enables a more effective discovering, automation, integration, and reusing of information among multiple applications [54, 55]. This facilitates searching, retrieving, processing, sharing, and integrating data across the Web [56]. The main concept of Web 3.0 emerged from the Semantic Web approach, which aims to deploy a Web that machines—and not only humans—can understand [57]. To provide the aforementioned capabilities, Web 3.0 comprises several technologies and tools. Following, the Web 3.0 technologies considered as more relevant for developing this thesis.

Complex Event Processing (CEP) defines a general concept that describes the capacity of inferring high-level knowledge by identifying significant patterns from multiple streams of simple events [58]. This technology uses techniques for detecting, filtering, causal and time correlation, abstraction, aggregation, and real-time computing that enable building applications aimed to situational intelligence [59].

Rule-based Systems technologies facilitate knowledge representation and reasoning. From a general perspective, Rule-based Systems consist of *(i)* ontologies, for symbolically describing the knowledge, *(ii)* rules, for decision making according to the modeled knowledge; and *(iii)* data, for applying the inference principle in order to derive new information [60, 61].

JavaScript Object Notation (JSON) provides a simple and lightweight data-interchange format. Humans comfortably read and write data using JSON, and machines easily parse and generate JSON data [62]. Web 3.0 approaches use JSON to define their data and link structure [63].

2.2 Related work

This section reviews the most relevant approaches that cope with SDN management requirements and that employ Web 2.0 and Web 3.0 technologies for managing networks.

2.2.1 SDN management

Most SDN proposals have tackled traditional network management tasks by carrying out managing *functions in NetApps* at the Application Plane. For example, wildcard-based algorithms [18] to better redistribute traffic in SDN networks, ElasticTree [19] to efficiently provide energy for SDN components, and Resonance [20] and OpenRoads [21] to control access for SDN resources. However, *functions in NetApps* lack of mechanisms to deal with several management requirements from distinct SDN architectural planes, such as: *(i)* in the Data Plane, configure certain NetDevs to communicate with a preferred NOS, *(ii)* in the Control Plane, set up a NetSlicer to link NOSs to their corresponding virtual network instances; and *(iii)* in the Application Plane, modify business parameters to customize NetApps logic.

Some investigations have tackled the above gap by providing *isolated tools* that address specific management require- ments for particular SDN technology instances. For example: For example: *(i)* OF-Config [22] and Open vSwitch Database (OVSDB) [23] that define protocols to configure NetDevs, *(ii)* Kandoo [24] and HyperFlow [25] to scale and distribute NOSs, *(iii)* OpenFlow Management Infrastructure (OMNI) [26] and ROVIZ [27] that provide graphic interfaces to monitor NOSs, *(iv)* VeRTIGO [28] and ADVisor [29] to configure NetSlicers; and *(v)* Pyretic [30] and Procera [31] that supply development tools to build NetApps. Considering that heterogeneous SDNs deploy a variety of resources from multiple vendors and distinct technologies, it is to highlight

Table 2.1. Research on approaches for SDN management

		Requirements for SDN management			
Research work	**Approach**	**Managing the whole SDN**	**Integrated management**	**Heterogeneous environment**	**Information Model**
[18]–[21]	Functions in NetApps	✓			
[22]–[31]	Isolated tools	✓			
[22, 32, 33]	Information Models		✓		✓

that a classic solution based on using *isolated tools* to accomplish a complete SDN management is complex and time-consuming.

As an essential step, an SDN management solution requires an Information Model that establishes a shared characterization of the entire SDN to enable integrated management in heterogeneous environments. Few approaches have defined *Information Models* to characterize the SDN managed environment: *(i)* OF-Config Data Model [22] that uses UML and XML to structure the configuration of an OpenFlow Capable Switch, *(ii)* ForCES Network Abstraction Model [32] that employs a building block approach to represent ForCES FEs; and *(iii)* CIM-SDN [33] that attempts to model SDN resources by proposing a CIM extension schema. It is worth noting that these *Information Models* fall short in representing the whole SDN managed environment and are tied to specific SDN technology instances. OF-Config Data Model and ForCES Abstraction Model describes only the Data Plane. The former was designed for OpenFlow and the latter for ForCES. CIM-SDN merely includes the main elements from the Data and the Control Planes. Although CIM-SDN is based on a technology-neutral model (*i.e.*, CIM), the extended schema is highly attached to the OpenFlow architecture.

Table summarizes the targets and gaps in SDN management of the above reviewed proposals. Unlike these proposals, we consider an SDN management approach based on a reference, technology-agnostic model of the whole SDN in order to achieve integrated management in heterogeneous SDN environments.

2.2.2 Network management using Web technologies

So far, in the literature, no investigation features an approach for SDN management based on Web 2.0 and Web 3.0 technologies. However, there are few research works that use either Web 2.0 or Web 3.0 mechanisms for carrying out management tasks in other network architecture domains. The next paragraphs describe the most relevant of such works.

In the domain of conventional networks, an approach based on mashups—a Web 2.0 technology—copes with the security problem of botnets[1] in an more flexible, extensible, and usable way [64]. The approach deployed a mashup application that dinamically integrated botnet information collected from existing mitigation tools (*e.g.*, sandboxes and antiviruses) with geographic location retrieved from online mapping and geolocation APIs.

In the same direction of the solution for mitigating botnets, a research work qualitatively evaluate the use of mashups as a feasible approach for managing conventional networks [65]. Such work carried out a mashup application that monitors the traffic of the Border Gateway Protocol (BGP) among two autonomous systems by integrating traffic router information. The application collected data by using the Simple Network Management Protocol (SNMP) and presented the integrated information by combining images retrieved from a traffic graph generator and a mapping service.

Continuing the latter two works, the authors propose a generic architecture for composing network management applications based on mashups [66]. The architecture enables network administrators to build and customize their management applications through high-level abstraction and user-oriented interfaces. The qualitative evaluation of the architecture carried out a mashup system prototype that deployed the BGP peering[2] mashup and the botnet mashup previously referred.

As an extension of the aforecited architecture, Maestro [67] provides a data confidentiality framework for mashups that assist daily activities of network administrators. For the proof-of-concept, an instance of Maestro deployed a prototype of a network traffic monitoring mashup. This mashup enabled to measure the time overhead caused by incorporating the modules of Maestro for achieving data confidentiality. It is to highlight that such quantitative evaluation lacks measuring the overall response time of the

[1]Network of infected machines (*i.e.*, bots) remotely controlled by human operators, usually for illegal purposes.

[2]Interconnection between two independently managed network domains.

mashup.

The Monitoring and Measurement in the Next Generation Technologies (MOMENT) project [68] deploys a framework that uses ontologies—a Web 3.0 technology—for semantically integrating the measurement information provided by distinct network monitoring tools and platforms. This framework enables some degree of inference and automatic reasoning over the retrieved measurement data. MOMENT relies on an ontology-based architecture [69] that supports semantic management of networks. This architecture focuses on semantically solving the interoperability problems among different existing information models for network management.

In the domain of Grid[3] networks, GEMINI2 [70] provides an architecture that employs CEP mechanisms—a Web 3.0 technology—to expose monitoring data as event streams. This monitoring approach provides access to such event streams by means of advanced, high-performance, and real-time queries that facilitate management operations, such as dynamic resource allocation. GEMINI2 conducted a test to quantitatively evaluate the network traffic added by its components, demonstrating that the proper use of CEP-based mechanisms allows to considerably reduce the intrusiveness caused by network monitoring.

Table 2.2 reveals several facts, first, none of the above-mentioned research works focused on SDN management by jointly using Web 2.0 and Web 3.0 technologies. Actually, research on using either mashups, ontologies, or CEP-based mechanisms for network management have not considered the virtual, dynamic, and heterogeneous environment of SDN. Second, the aforecited works lack evaluating the consuming of time for carrying out management tasks on the deployed scenarios. Third, there is the need to deeply analyze the response time and the network traffic of network management approaches that use Web 2.0 and Web 3.0 technologies; the formerly described research just conducted a concise evaluation of overhead for these metrics. It is to point out that the above facts externalize a research gap located at the intersection of the SDN environment and the use of Web technologies (2.0/3.0) for network management. The present thesis leverages such gap to introduce a groundbreaking framework aimed to carry out SDN management by using mechanisms from the Web 2.0 and the Web 3.0.

[3]A distributed computing system that allows to share not geographically-centric resources in order to solve big scale problems.

Table 2.2. Research on using Web technologies for network management

		Web technology			Evaluated characteristic		
Research work	SDN	Mashup	Ontology	CEP	Time of task	Time of response	Network Traffic
[64]		✓					
[65]		✓					
[66]		✓					
[67]		✓				✓	
[68]			✓				
[69]			✓				
[70]				✓			✓

* A rule-based representation model.

2.3 Final remarks

Initially, this chapter detailed the typical architecture for deploying an SDN-based network. Subsequently, it described the technologies from Web 2.0 and Web 3.0 that this thesis considers more appropiate for defining the SDN management approach. Afterwards, the chapter presented several research works that aim to address management requirements of the SDN environment and that apply Web technologies (2.0/3.0) on network management. The related work analysis shows that the literature lacks an integrated approach aimed to manage the virtual, dynamic, and heterogeneous environment of SDN as a whole. Unlike such research works, this thesis aims to propose an Information Model and a Web-based (2.0/3.0) framework towards building an integrated management architecture for the entire SDN environment regardless the deployment technologies.

Chapter 3

A Management Plane for SDN based on CIM and Web technologies

This chapter provides an extensive study about defining a Management Plane approach that facilitates network administrators to address management requirements in virtual, dynamic, and heterogeneous networks based on the SDN paradigm. In this sense, the chapter starts proposing a reference architecture of the Management Plane aimed to carry out integrated management in SDN environments by referencing the OSI network management submodels: Information, Organizational, Communication, and Functional. Then, the chapter introduces a CIM-based Information Model for such Management Plane in order to establish a technology-agnostic and consistent characterization of the whole SDN environment. After, this chapter presents a mashup-based concept that defines a high-level composition of applications for carrying out management tasks in SDN-based networks. Finally, the chapter proposes a framework that supports the static and dynamic deployment of the above SDN mashup concept in the Management Plane.

3.1 Management Plane reference architecture

To define the management proposal for the SDN architecture, this thesis extends the Management Plane concept considered in early SDN approaches [16, 17]. Unlike these approaches, the proposed Management Plane aggregates components that facilitate the integrated management in the virtual, dynamic, and heterogeneous environment of

SDN.

Figure 3.1 depicts the proposed Management Plane. This plane is formed by the Data Repository, the Manager, the Adapters, the Management Interfaces, and the Agents. The Data Repository holds the Resource Representation Model (RRM) and serves the Manager to store management instance data. RRM handles metadata to provide an abstract, technology-neutral characterization of SDN resources. The Manager orchestrates and deploys the Management Services to carry out different SDN management functions. These Management Services expose user interfaces to allow interaction of Network Administrators. The Adapters afford a protocol-agnostic communication between the Manager and the Agents through well-defined Management Interfaces. Each Management Interface connects every Adapter with the corresponding Agent. The Agents situate on SDN resources to act on behalf of the Manager. The whole operation of the Management Plane is based on RRM to achieve an integrated and technology- independent SDN management.

This Management Plane defines a reference architecture based on the four OSI network management submodels [34]: *(i)* an Information Model to establish a shared abstraction of SDN resources, *(ii)* an Organizational Model to specify roles and collaboration forms, *(iii)* a Communication Model to delineate the exchange of management data; and *(iv)* a Functional Model to structure management requirements.

3.1.1 Information Model

The Management Plane incorporates a CIM-based Information Model that describes the SDN management environment at a conceptual level regardless of deploying technologies. Using the Unified Modeling Language (UML), the Information Model graphically represents SDN resources and their relationships as CIM classes and associations, respectively [71]. This object-oriented, well-understood abstract framework standardizes SDN management information across disparate vendors and SDN instances. Thus, enabling to carry out integrated management in the virtual, dynamic, and heterogeneous environment of SDN. Furthermore, network designers may extend the proposed CIM model to include customized resource behavior. In the Management Plane, the Information Model is realized by RRM in the Data Repository. This thesis focuses on the details of the proposed Information Model in Section 3.2.

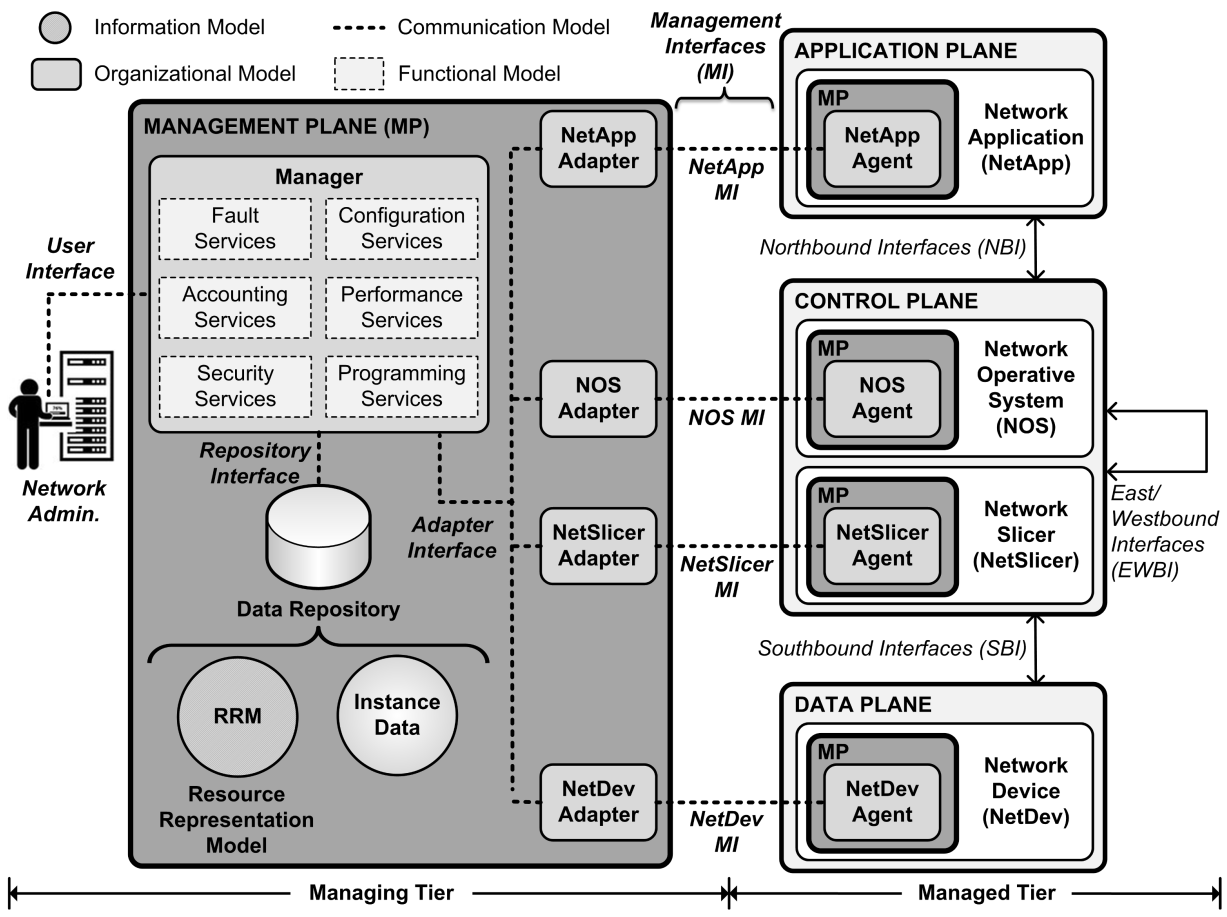


Figure 3.1. High-level SDN architecture with Management Plane

3.1.2 Organizational Model

The Management Plane depicts a two-tier like network management model that incorporates three kinds of entities (*a.k.a.* roles). A Managing Tier that encloses *manager* and *adapter* entities, and a Managed Tier that contains *agent* entities. A *manager* entity is responsible for: *(i)* housing and coordinating logic of management functions, *(ii)* providing user interaction with deployed management functions through tailored user interfaces (*e.g.*, command-line, graphical, and Web-based); and *(iii)* sending requests to and receiving replies and notifications from *agents* by means of *adapters*. An *adapter* entity allows a *manager* to interact with any specific *agent* by parsing data formats and protocols handled by their communication interfaces (*i.e.*, the Adapter Interface and the Management Interfaces). An *agent* entity resides on managed resources to carry out management requests delegated by a *manager*, such as performing an operation or responding to a query. In addition, an *agent* entity may dispatch unsolicited notifications to a *manager*.

Each organizational component in the Management Plane gets the same name as its corresponding role. The Manager acts as a *manager* entity. The NetApp Adapter, the NOS Adapter, the NetSlicer Adapter, and the NetDev Adapter play an *adapter* role. The NetApp Agent, the NOS Agent, the NetSlicer Agent, and the NetDev Agent perform *agent* tasks. The Management Plane differentiates the Adapters and the Agents regarding of SDN managed resources (*i.e.*, NetApp, NOS, NetSlicer, and NetDev) to demarcate the communication between such kind of entities located at different architectural planes.

3.1.3 Communication Model

The Management Plane defines the User Interface, the Repository Interface, the Adapter Interface, and the Management Interfaces. The User Interface enables Network Administrators to interact with the Management Services exposed by the Manager. The Repository Interface connects the Manager with the Data Repository. The Adapter Interface and the Management Interfaces transport request messages (*i.e.*, operations and queries) from the Manager to a particular Agent, passing through the related Adapter. These both kind of interfaces also transmit reply messages and unsolicited notifications sent by any Agent towards the Manager. In order to match each Agent with its respective Adapter, the Management Plane establishes a Management Interface per SDN managed resource: NetApp MI, NOS MI, NetSlicer MI, and NetDev MI.

Regarding communication support, the User Interface and the Adapter Interface must employ a consistent data format (*e.g.*, CIM, XML, and JSON) and a standardized protocol (*e.g.*, HTTP and SOAP) to exchange management data. The Repository Interface relies on technologies deployed by the Data Repository (*e.g.*, XML over JRMP or HTTP). Finally, the Management Interfaces handle data formats and protocols implemented by the Agents (*e.g.*, OVSDB, NETCONF, and SNMP).

3.1.4 Functional Model

The Management Plane references the five OSI management functional areas to classify the Management Services: Fault Services, Configuration Services, Accounting Services, Performance Services, and Security Services. The Fault Services detect, separate, and fix failures in physical and logical SDN resources. The Configuration Services modify and update behavior of SDN resources. The Accounting Services tracks and allocate usage of SDN resources. The Performance Services monitor, collect, and report information about the operation of SDN resources. The Security Services control and analyze access to SDN resources. In addition, this Management Plane includes the Programming Services to coordinate programmable software of SDN resources, such as checking and deploying versions of a particular NetApp running on a specific NOS.

By using or combining the aforementioned Management Services, the proposed approach allows network administrators to carry out different SDN management requirements, as those described in [16].

3.2 CIM-based Information Model

As aforementioned, the proposed Management Plane requires an Information Model that provides a technology-agnostic and consistent abstraction of the whole SDN environment to enable integrated management. Few approaches provide models that attempt to characterize the SDN management environment [22, 32, 33], but they are tied to specific SDN instances and expose incomplete SDN representations.

This thesis proposes a CIM-based Information Model that provides a technology-independent and consistent abstraction of SDN managed and managing resources. The Information Model represents every plane in the SDN architecture to encourage a complete SDN management regardless of deploying technologies. This thesis adopted CIM [71] because it offers higher expressiveness than other information definition languages (*e.g.*, Structure of Management Information [SMI] and Guidelines for the Definition of Managed Objects [GDMO]), affording future robust semantic integration [72]. CIM supplies several classes, associations, properties and methods to describe network resources at a conceptual level, such as Ethernet ports, LAN endpoints, and VLANs [73]. However, CIM lacks elements that represent specific SDN features for management [33]. Thus, the proposed Information Model extends the actual CIM Schema to characterize the whole SDN management environment. The resulting CIM extended schema employs a graphical visualization composed by UML classes and associations to model the SDN managed and managing resources and their relationships.

Following paragraphs and figures describe a simple version of the proposed Information Model. Specific properties and methods from each class are out of scope. The depicted class schema excludes the *CIM_* prefix from the current CIM elements and the *SDN_* prefix from the extended elements. For example, *CIM_System* appears as *System* and *SDN_AgentService* as *AgentService*. To provide a better visualization, such schema displays gray background for the extended classes, white background for the CIM classes, bold characters for the extended associations, and thin characters for the CIM associations. Also, for the sake of simplicity, it omits inheritance associations between the extended classes and the CIM classes. Unless otherwise stated, general inheritance associations satisfy the following: *(i)* the extended classes with suffix *Capabilities* derive (*i.e.*, are subclasses) from the *EnabledLogicalElementCapabilities* CIM class, *(ii)* with suffix *Service* from the *Service* CIM class; and *(iii)* with suffix *Settings* from the *SettingData* CIM class. In addition, the outlined class schema skips the *BindsTo* CIM associations for the CIM classes *ServiceAccessPoint* and *ProtocolEndpoint*. The *BindsTo* association connects the class itself to define a protocol layering dependency between an upper and a lower protocol. For example, the OpenFlow protocol binds the TCP protocol to set the port and address enabled for OpenFlow communication.

3.2.1 Class schema for the Management Plane

Figure 3.2 illustrates the class schema for the proposed Management Plane. It extends five new classes to characterize the novel components defined in this approach: in the Managing Tier, the *ManagementService*, the *ManagementServiceCapabilities*, and the *AdapterService*; and in the Managed Tier, the *AgentService* and the *Notification*.

The *ManagementService* class represents Management Services that allow to carry out different SDN management functions. Through the *ElementCapabilities* association, the *ManagementServiceCapabilities* class describes both supported and excluded abilities for Management Services. The *ManagementServiceCapabilities* relies on the Functional Model to classify SDN Management Services as Fault, Configuration, Accounting, Performance, Security, or Programming services (see Subsection 3.1.4). For example, a Management Service that modifies the SBI communication of NetDevs de-

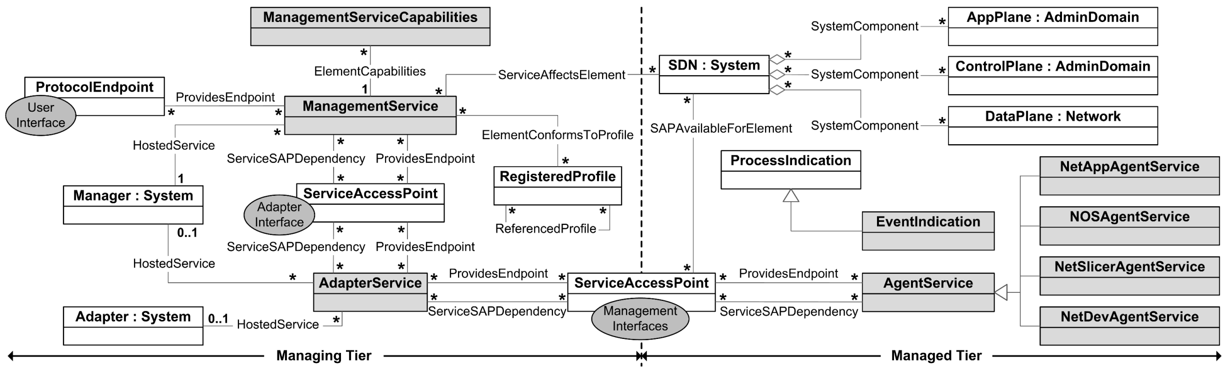


Figure 3.2. Class schema to model the SDN Management Plane

clares capabilities of Configuration Services.

The *RegisteredProfile* class models a CIM profile specification defined by any standard organization for managing SDNs. Each profile specification includes a small subset of the proposed class schema and delineates corresponding behavior as management requirements. The *ReferencedProfile* association indicates that a profile specification may reference others. In addition, the *ElementConformsToProfile* association describes which CIM profile specifications a Management Service apply. For example, a Configuration Service fulfills with a profile specification of DMTF that standardizes how to achieve seamless migration in NetDevs.

The *Manager* represents the system hosting the SDN Management Services. The *HostedService* association realizes this relationship between the *ManagementService* and the *Manager*. This model presents the *Manager* as an instance of the abstract *System* class, thus the *Manager* implements an instance of a subclass deriving from *System*, such as *ComputerSystem*, *J2eeServer*, or a new extended class. For example, a Configuration Service may be carried out as a Web application running on either an Apache Tomcat Server or a GlassFish Server.

The *ProtocolEndpoint* class tagged as *User Interface* models the communication point that enables access of Network Administrators. The corresponding *ProvidesEndpoint* association implies that the *ManagementService* supplies such user *ProtocolEndpoint*. For example, a Configuration Service provides an HTTP interface to allow Network Administrators to set SBI parameters of NetDevs through a Web browser.

The *ServiceAccessPoint* class tagged as *Adapter Interface* represents the communication point between the *ManagementService* and the *AdapterService*. The *ProvidesEndpoint* associations connected to the adapter *ServiceAccessPoint* indicate that both the *ManagementService* and the *AdapterService* establish their own communication points to allow access from the other. The *ServiceSAPDependency* associations imply that both the *ManagementService* and the *AdapterService* utilize the adapter *Service-AccessPoint* to access the other. The *ManagementService* and the *AdapterService* support properties and methods for sending and receiving requests, responses, and notifications through the adapter *ServiceAccessPoint*. For example, a Configuration Service and a NetDev Adapter establish a mutual communication using JSON over HTTP. Using this channel, the NetDev Adapter forwards to the Configuration Service a notification from a NetDev Agent that reports about misconfiguration failures. Similarly, the Configuration Service uses the same channel to fix this failure by sending a configuration request to the NetDev Adapter. The NetDev Adapter forwards this request to the corresponding NetDev Agent.

The *AdapterService* class models an Adapter in charge of parsing and forwarding requests, responses, and notifications between the *ManagementService* and the *AgentService*. The *AdapterService* is a superclass that holds properties and methods for handling the communication through the adapter and management interfaces. Four subclasses inherit from the *AdapterService*: the *NetDevAdapterService*, the *NetSlicerAdapterService*, the *NOSAdapterService*, and the *NetAppAdapterService*. For the sake of brevity and because the behavior of these subclasses is very similar, the depicted schema excludes them in Figure 3.2. Each subclass from the *AdapterService* adds properties and methods to support functionality provided by the subclass from the *AgentService* that uses the same managed resource name (*e.g.*, the *NetDevAdapterService* matches the *NetDevAgentService*). In addition, regarding this correlation, every subclass deriving from the *AdapterService* instruments specific aspects from the proposed class schema. For example, a NetDev Adapter, which matches a NetDev Agent, only concerns about functionality for managing NetDevs (see Figure 3.5).

The *AdapterService* may be hosted by either the *Manager* or the *Adapter*. Both *HostedService* associations linked to the *AdapterService* indicate this relationship. As well as the *Manager*, the *Adapter* is an instance of the abstract *System* class. For example, a NetDev Adapter may be executed on either the same server running Management Services or a different server.

The *ServiceAccessPoint* class tagged as *Management Interfaces* represents the communication point between the *AdapterService* and the *AgentService*. The *Pro-*

videsEndpoint and the *ServiceSAPDependency* associations related to the management *ServiceAccessPoint* indicate that both the *AdapterService* and the *AgentService* provide and utilize management interfaces to perform their functionality. Subclasses from the *AdapterService* and from the *AgentService* inherits these associations. Each instance of the subclasses from the *AdapterService* handles the protocol used by the corresponding instance of the subclasses from the *AgentService*, affording a protocol-agnostic communication for the *ManagementService*. For example, a NetDev Adapter uses the OF-Config protocol to access a NetDev Agent for OpenFlow switches. A second NetDev Adapter utilizes the SNMP protocol to contact a second NetDev Agent for ForCES FEs. A Configuration Service communicates with both NetDev Adapters using a standardized data format and protocol (*e.g.*, JSON over HTTP). The NetDev Adapters forward to the NetDev Agents the management requests received from the Configuration Service. Similarly, the NetDev Adapters forward to the Configuration Service responses and notifications received from the NetDev Agents. Thus, the Configuration Service carry out a protocol-agnostic management on different NetDev technology instances.

The *AgentService* class represents an Agent running on SDN managed resources, such as NetDev, NetSlicer, NOS, and NetApp. This is a superclass that defines properties and methods for supporting the management *ServiceAccessPoint* and for handling the *Notification*. The *Notification* is a subclass from the *ProcessIndication* class. The *Notification* maps an unsolicited message sent by the *AgentService* towards the *ManagementService* to inform about state changes and alerts of SDN managed resources. For example, an SNMP Agent dispatches a trap message that notifies a detected misconfiguration of its hosting NetDev. The corresponding NetDev Adapter receives and parses this unsolicited notification and forwards it to a Configuration Service.

Four subclasses derive from the *AgentService*: the *NetDevAgentService*, the *NetSlicerAgentService*, the *NOSAgentService*, and the *NetAppAgentService*. Each subclass supports methods to carry out management tasks in its hosting SDN managed resource, such as retrieving statistical information, modifying configuration parameters, discovering capabilities, and changing communication attributes.

The proposed schema uses the *System* class to model the *SDN* as an entity composed by the *DataPlane*, the *ControlPlane*, and the *AppPlane*. The *Network* class represents the *DataPlane* as a logical, virtual, or physical network that groups interconnected NetDevs capable of exchanging information. The *AdminDomain* class indicates

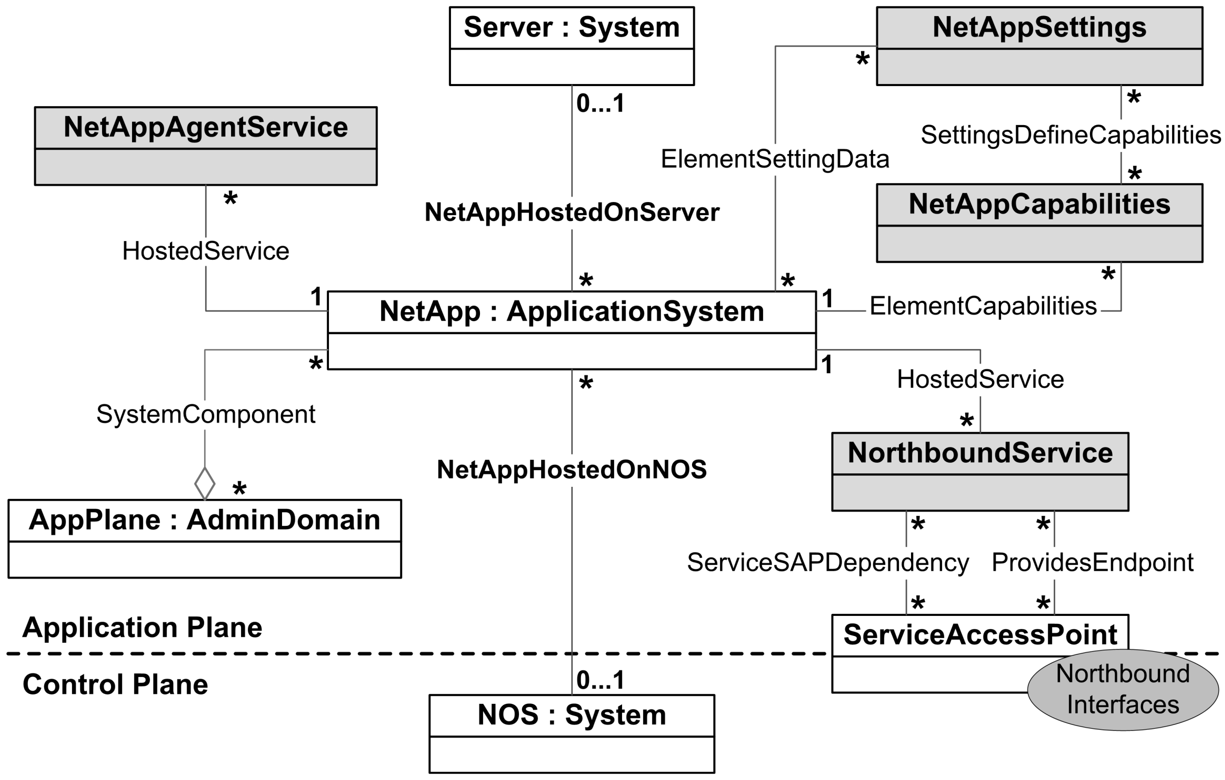


Figure 3.3. Class schema to model the SDN Application Plane

that the *ControlPlane* and the *AppPlane* gather similarly managed components, such as NetSlicers and NOSs for the Control Plane, and NetApps for the Application Plane.

The *ServiceAffectsElement* association between the *SDN* and the *Management-Service* reflects that Management Services have an effect in the entire SDN, such as changing resource behavior, monitoring failures, and analyzing performance. Besides, the *SAPAvailableForElement* association between the *SDN* and the management *ServiceAccessPoint* implies that management interfaces provide managing access for the whole SDN.

3.2.2 Class schema for the Application Plane

Figure 3.3 shows the class schema for the Application Plane. It extends three new classes and two novel associations to describe specific management features of NetApps. The extended classes are the *NetAppCapabilities*, the *NetAppSettings*, and the *NorthboundService*. The extended associations are the *NetAppHostedOnNOS* and the *NetAppHostedOnServer*.

The *AppPlane*, modeled with the *AdminDomain* class, uses the *SystemComponent* association to aggregate instances of the *NetApp*. Leveraging the *ApplicationSystem* class, the *NetApp* represents NetApps holding business logic on top of the SDN architecture. For example, NetApps that carries out load-balancing and access-control tasks.

The outlined schema uses the *HostedService* association to indicate that the *NetApp* hosts the *NetAppAgentService* and the *NorthboundService*. The *NorthboundService* class models modules that communicate with services exposed by NOSs. The *ProvidesEndpoint* and the *ServiceSAPDependency* associations reflect that the *NorthboundService* uses and provides functions through the northbound *ServiceAccessPoint*. For example, NetApps performing load-balancing and access-control functionality retrieve and supply data from and to tracking and firewall services deployed by an NOS.

The *ServiceAccessPoint* tagged as *Northbound Interfaces* models the communication between the Application Plane and the Control Plane. This northbound *ServiceAccessPoint* encompasses different NBIs, such as APIs based on protocols (*e.g.*, REST) and on programming languages (*e.g.*, Pyretic and Procera).

The *NetAppHostedOnNOS* association between the *NetApp* and the *NOS* represents local NetApps running on NOSs. Usually, these local NetApps utilize NBIs based on programming languages to access and supply functionality from and to NOS services. The *NetAppHostedOnServer* between the *NetApp* and the *Server* system models NetApps running on remote servers. These remote NetApps prefer NBIs based on protocols for communicating with the Control Plane.

Using the *ElementCapabilities* association, the *NetAppCapabilities* class describes the supported and excluded abilities of NetApps. The *NetAppSettings* class establishes configuration parameters for the *NetApp*. This relationship is depicted through the *ElementSettingData* association between the *NetAppSettings* and the *NetApp*. The *SettingsDefineCapabilities* association between the *NetAppSettings* and the *NetAppCapabilities* reflects that setting certain configuration data may affect some NetApps capabilities. For example, configuring a different load-balancing algorithm modifies the behavior of the corresponding NetApp.

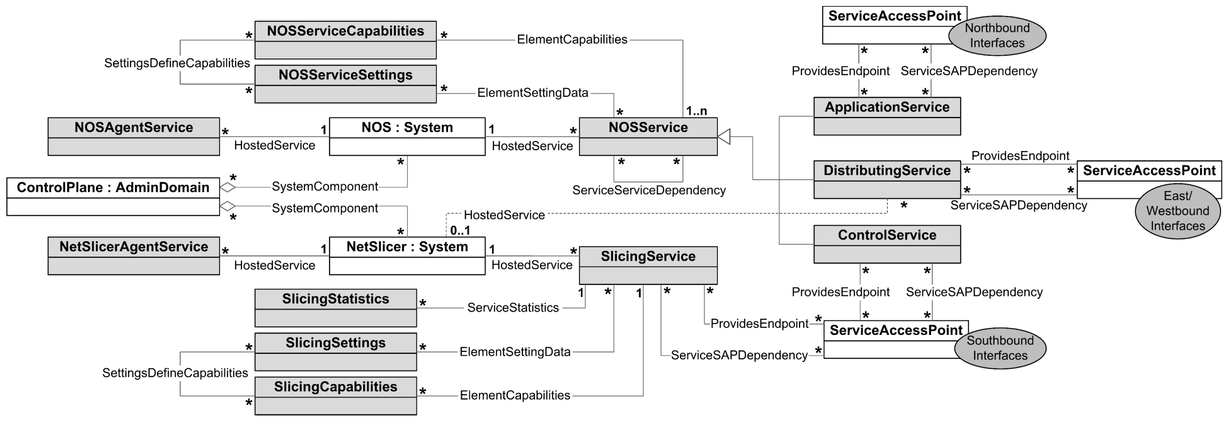


Figure 3.4. Class schema to model the SDN Control Plane

3.2.3 Class schema for the Control Plane

Figure 3.4 describes the class schema for the Control Plane. Considering that CIM lacks elements that characterize the management information of NetSlicers and NOSs, this schema extends seven new classes: the *SlicingService*, the *SlicingStatistics*, the *SlicingCapabilities*, the *SlicingSettings*, the *NOSService*, the *NOSServiceCapabilities*, and the *NOSServiceSettings*.

The *AdminDomain* class uses the *SystemComponent* association to describe the *ControlPlane* as an entity composed by NOSs and NetSlicers. The *NOS* models an NOS, such as OpenFlow controllers and ForCES CEs. The *NetSlicer* represents a NetSlicer system, such as FlowVisor for OpenFlow-based networks. The *NOS* is the hosting system for the *NOSAgentService*; the *NetSlicer*, for the *NetSlicerAgentService*.

The *NOSService* is a superclass that models network services hosted in NOSs. The *HostedService* association between the *NOSService* and the *NOS* indicates this relationship. Subclasses must inherit from the *NOSService* in order to define specific NOS services, such as tracking, route calculation, and firewall. The depicted schema presents three subclasses for NOS services: the *ApplicationService*, the *DistribuitingService*, and the *ControlService*. The *ApplicationService* depicts services that expose functionality to the Application Plane through the northbound *ServiceAccessPoint*. The *DistribuitingService* defines services that enable to deploy distributed Control Plane through the east/westbound *ServiceAccessPoint*. The *ControlService* describes services that handle the communication with NetDevs and NetSlicers through the southbound *ServiceAccessPoint*.

The *ServiceServiceDependency* association indicates that NOS services collaborate with or are necessary for other NOS services to perform their operation. For example, a topology service requires a tracking service to recognize hosts connected to specific switches.

The proposed schema uses the *ProvidesEndpoint* and the *ServiceSAPDependency* associations to correlate the *ApplicationService* with the northbound *ServiceAccessPoint*, the *DistribuitingService* with the east/westbound *ServiceAccessPoint*, and the *ControlService* with the southbound *ServiceAccessPoint*.

The *ServiceAccessPoint* tagged as *East/Westbound Interfaces* represents the communication point between distinct Control Plane domains. For example, the SDNi protocol from IETF and the CE-CE interface from ForCES. Although the exchange of information is usually carried out by NOSs, NetSlicers may also need to deploy a distributed architecture. This is reflected by the dashed *HostedService* association between the *DistribuitingService* and the *NetSlicer*.

The *ServiceAccessPoint* tagged as *Southbound Interfaces* models the communication point between the Control Plane and the Data Plane. The *ServiceSAPDependency* and the *ProvidesEndpoint* associations reflect that the *ControlService* uses and supplies the southbound *ServiceAccessPoint* to send and receive messages to and from the Data Plane. This southbound *ServiceAccessPoint* encompasses different SBI protocols, such as OpenFlow, ForCES, and POF.

The *NOSServiceCapabilities* class declares the supported abilities of NOSs services. The *ElementCapabilities* association between the *NOSServiceCapabilities* and the *NOSService* reflects this relationship. The *NOSServiceSettings* class defines the configuration parameters for NOSs services. The *ElementSettingData* association between the *NOSServiceCapabilities* and the *NOSService* indicates this relationship. The *SettingsDefineCapabilities* association between the *NOSServiceSettings* and the *NOSServiceCapabilities* implies that configuring NOS services establishes some capabilities. For example, the time interval of a tracking service to send discovery messages.

The *SlicingService* class represents the functionality of a NetSlicer: dividing the Data Plane into several isolated logical network instances (*a.k.a.* slices) and assigning them to different NOSs. The *NetSlicer* hosts the *SlicingService* using the *HostedService* association. The *ProvidesEndpoint* and the *ServiceSAPDependency* associations between the *SlicingService* and the southbound *ServiceAccessPoint* indicate that NetSlicers provide and use SBIs to communicate with NetDevs and NOSs. For example, FlowVisor uses the OpenFlow protocol to communicate with both OpenFlow switches and OpenFlow controllers.

The *SlicingStatistics* class defines collections of metrics suitable to instances of the *SlicingService*. The *SlicingStatistics* is a subclass that derives from the *StatisticalData*. The *ServiceStatistics* association relates the *SlicingStatistics* with the *SlicingService*. For example, the total number of slices handled by a NetSlicer.

Through the *ElementCapabilities* association, the *SlicingCapabilities* class describes the supported and excluded capabilities of the *SlicingService*. Some of these capabilities are specified in the *SlicingSettings* class by means of the *SettingsDefineCapabilities* association. The *SlicingSettings* delineates the configuration parameters for the *SlicingService*. The *ElementSettingData* association between the *SlicingSettings* and the *SlicingService* reflects this relationship. For example, the maximum number of concurrent slices supported by a NetSlicer.

3.2.4 Class schema for the Data Plane

Figure 3.5 depicts the class schema for the Data Plane. In order to describe specific management features of NetApps, this schema extends five new classes: the *NetDevCapabilities*, the *NetDevResource*, the *NetDevResourceSettings*, the *NetDevService*, and the *NetDevServiceSettings*.

As aforementioned, the *Network* class indicates that the *DataPlane* models a network composed by interconnected NetDevs. The *NetDev* represents a NetDev system within a network, such as OpenFlow switches and custom forwarding hardware (*e.g.*, OpenWrt and NetFPGA). The *DataPlane* aggregates the *NetDev* using the *SystemComponent* association. The *NetDev* hosts the *NetDevAgentService*.

The supported and excluded abilities of a NetDev are described by instances of the *NetDevCapabilities*. The *ElementCapabilities* association between the *NetDev* and the *NetDevCapabilities* indicates this relationship. For example, an OpenFlow switch declares simple capabilities such as forwarding functions based on match/action flow tables; a NetFPGA programmable hardware exposes more complex capabilities such as plugging modules to enable customizable functions. Both kinds of NetDevs also reveal network capacity enabled by its components, such as speed of ports and size of queues.

The *NetDevResource* class inherits from the *EnabledLogicalElement* to model network elements composing a NetDev, such as ports, queues, and tables. The *Sys-*

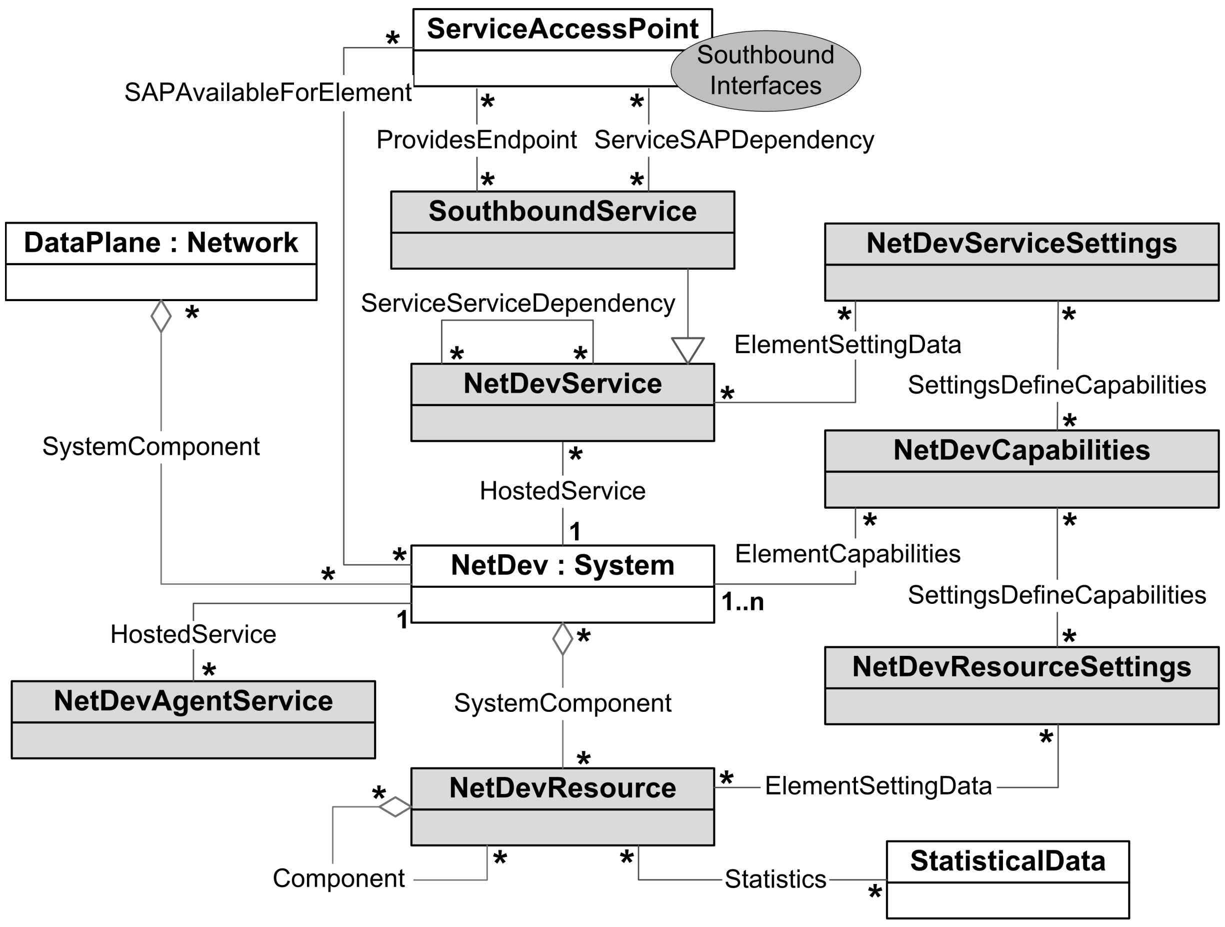


Figure 3.5. Class schema to model the SDN Data Plane

temComponent association between the *NetDev* and the *NetDevResource* implies this aggregation. The *NetDevResource* is a superclass from which individual subclasses derive to represent NetDev components. For example, a subclass called *FlowTable* to characterize flow tables that compose OpenFlow switches. In addition, the *Component* association connected to the *NetDevResource* models a network element composed by others, such as ports including various queues.

The *StatisticalData* is a superclass to define arbitrary collections of statistical information applicable to instances of the *NetDevResource*. The *Statistics* association attaches the *StatisticalData* with the *NetDevResource*. For example, ports in switches delineate transmission metrics, such as received and transmitted bytes, packets, and errors.

The *NetDevResourceSettings* class describes the configuration of network elements that compose NetDevs. The *ElementSettingData* association between the *NetDevRe-*

sourceSettings and the *NetDevResource* reflects this relationship. For example, the speed operation of ports and the buffer size of queues. The *SettingsDefineCapabilities* association between the *NetDevResourceSettings* and the *NetDevCapabilities* indicates that the configuration parameters of NetDev components specify some capabilities of NetDevs.

The *NetDevService* is a superclass that represents network services hosted in NetDevs. This hosting relationship is depicted with the *HostedService* association between the *NetDevService* and the *NetDev*. Subclasses must derive from the *NetDevService* in order to model particular NetDev services, such as forwarding, route calculation, and firewall. This is the case of the *SouthboundService*, which defines services that query, receive, and execute instructions to and from the Control Plane. The *SouthboundService* inherits from the *NetDevService* and includes properties and methods to handle the communication through the SBIs. For example, the Secure Channel component in OpenFlow switches manipulates the OpenFlow protocol to communicate with external controllers and update internal flow tables.

The *ServiceServiceDependency* association indicates that NetDev services cooperate with or are required for other NetDev services to perform their functions. For example, an inspection service requires a forwarding service to redirect malicious packets to a specific destination.

The *ServiceSAPDependency* and the *ProvidesEndpoint* associations reflect that the *SouthboundService* uses and supplies the southbound *ServiceAccessPoint* to send and receive messages to and from the Control Plane. The *SAPAvailableForElement* association between this *ServiceAccessPoint* and the *NetDev* implies that the SBIs allow access from the Control Plane for managing NetDevs components and hosted services. For example, the OpenFlow protocol enables to manipulate flow tables within OpenFlow switches and the ForCES protocol facilitates to configure logical functions residing in ForCES FEs.

Through the *ElementSettingData* association, the *NetDevServiceSettings* class describes the configuration parameters of services hosted in NetDevs. For example, the routing algorithm of a route calculation service and the list of OpenFlow controllers of a southbound service. The *SettingsDefineCapabilities* association between the *NetDevServiceSettings* and the *NetDevCapabilities* implies that the configuration of NetDev services characterizes some capabilities of NetDevs.

3.3 Manager based on Web 2.0/3.0 technologies

As stated in Section 3.1, the Manager component of the Management Plane coordinates and deploys the Management Services to allow Network Administrators to carry out management functions on the SDN architecture. This thesis proposes an approach based on technologies from Web 2.0 and Web 3.0 for realizing such Manager in order to facilitate Network Administrators to perform, in a high-level abstraction, management tasks in the virtual, dynamic, and heterogeneous environment of SDN. The following subsections detail the fundamental concepts and the structure of the approach.

3.3.1 SDN Mashup

SDN Mashups are high-level composite Web applications aimed to manage resources deployed by an SDN-based network (*i.e.*, NetDev, NetSlicer, NOS, and NetApp). In this approach, Network Administrators are able to create SDN Mashups by using end-user oriented techniques, such as wiring and drag-and-drop mechanisms. Thus, Network Administrators do not require deep knowledge about the APIs and data formats used to communicate with the SDN managed resources. It is important to highlight that an end-user programming approach, as defined by SDN Mashups, provides flexibility for Network Administrators to build customized SDN management tools by themselves and promotes the innovation in SDN management solutions.

The SDN Mashup concept poses some features that existing mashups do not support. First, it combines information, on the fly, from multiple managed resources, such as NetDev, NetSlicer, NOS, and NetApp. Second, it hides the heterogeneity and complexity of these SDN resources in order to facilitate the carrying out of management tasks. Third, it blends local and remote visualization APIs to generate integrated Graphical User Interfaces (GUI). 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, this approach leads the SDN management towards an end-user centric environment, where several Network Administrators are able to participate and collaborate in order to cope their own needs, and even obtain profits.

In a general way, an SDN Mashup is formed by combining Management Services, Operation Services, and Visualization Services. The Management Services perform

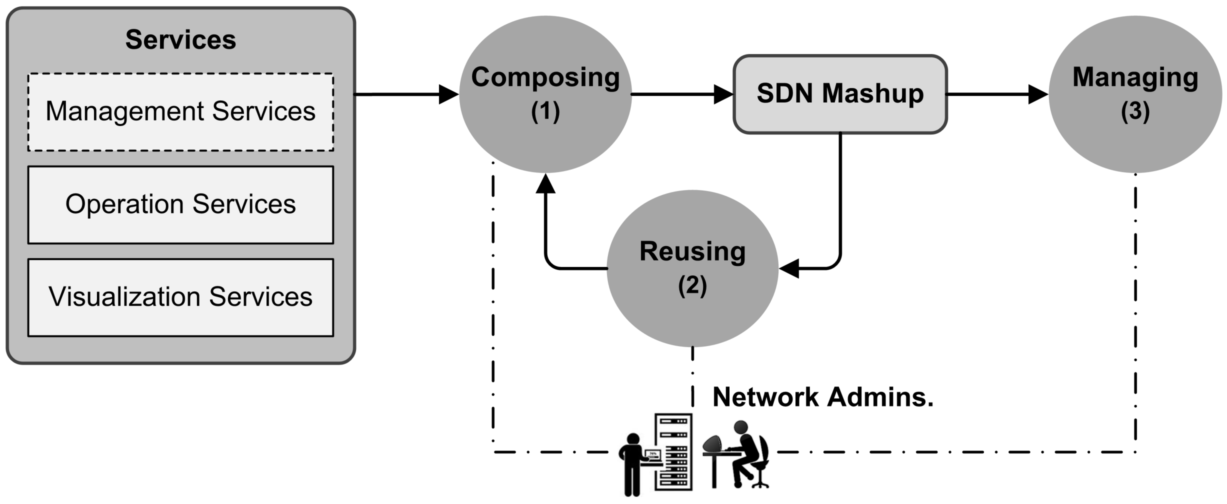


Figure 3.6. Global vision of SDN Mashups

different management functions on SDN resources. Specifically, this approach defined the following Management Services (see details in Subsection 3.1.4): Fault, Configuration, Accounting, Performance, Security, and Programming (FCAPS+P). The Operation Services provide control patterns and structures of mashup composition, such as sequential, conditional, and merge. An Operation Service can be used, for instance, to sort, filter, and aggregate the information retrieved by a Performance Service from one or more NOSs. The Visualization Services represent graphics and presentation libraries that allow to generate the integrated GUIs of SDN Mashups.

Figure 3.6 presents the global vision of SDN Mashups focusing on how Networks Administrators create SDN management solutions by conducting three processes: Composing, Reusing, and Managing. The Composing process defines that Network Administrators select, combine, and configure the available Services (*i.e.*, Management, Operation, and Visualization) for building the target SDN Mashup. The Reusing process represents that any built SDN Mashup serves to create another one. Different Network Administrators may use the same or similar candidate Services to compose a new SDN Mashup. The Managing Process indicates that Network Administrators execute SDN Mashups in order to manage one or several resources deployed by an SDN-based network. An SDN Mashup carries out the management functions through an underlying process that is always hidden for Network Administrators.

3.3.2 Mashup-based and Event-driven Framework

The Mashup-based and Event-driven (MaE) Framework leverages the main foundations of the mashup technology, CEP mechanisms, rule-based systems, and the JSON format to simplify the management tasks that Network Administrators carry out by interacting with the Manager component of the Management Plane. In particular, the M&E framework offers: *(i)* the static composition and execution of SDN Mashups, *(ii)* the automatic recognition of events occurring in the SDN environment; and *(iii)* the dynamic composition and execution of SDN Mashups by using static compositions that point to tackle specific events.

Figure 3.7 depicts the structure defined by the M&E Framework to deploy the Manager component of the Management Plane. Four modules constitute this framework: the Service Container, the M&E Builder, the Mashup Executor, and the Publisher. In addition, the M&E Framework considers three users: the Network Administrator, the M&E Developer, and the Service Developer. These users interact with the proposed framework by using a Web Client, a Mobile Client, or an Integrated Development Environment (IDE).

The Service Container stores the Services that form SDN Mashups, providing them as mashable entities to the M&E Builder and the Mashup Executor. This approach defines three types of Services: the Management Services, the Operation Services, and the Visualization Services. The Management Services carry out managing functions on resources deployed by the SDN environment (*i.e.*, NetDev, NetSlicer, NOS, and NetApp). Depending upon the functionality performed, these Management Services divide into the FCAPS+P services. It is important to remind that the Management Services rely on the components and interfaces defined by the Management Plane (see Section 3.1) to achieve a two-way, technology-neutral communication with their corresponding SDN managed resources.

The Operation Services provide elements that allow combining mashable entities (*i.e.*, Services) and, so, building up and generating SDN Mashups. There are three types of Operation Services: *(i)* the control patterns that allow defining the workflow of SDN Mashups (*e.g.*, sequential, parallel, and conditional), *(ii)* the structures for configuring and invoking the Services that form SDN Mahups (*e.g.*, functionalities to set security credentials); and *(iii)* the structures for receiving, sorting, and filtering the retrieved information from any type of Service (*e.g.*, functionalities to perform information

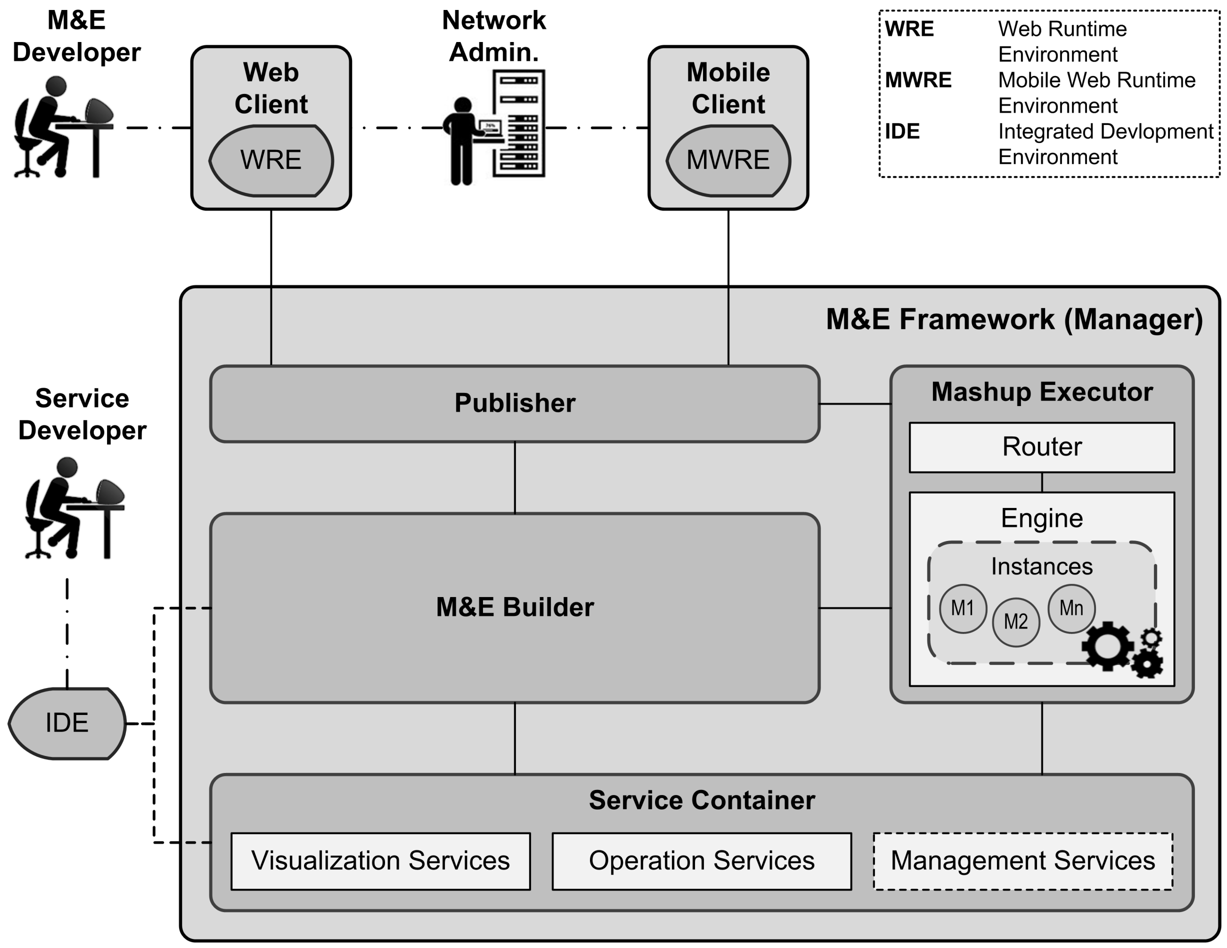


Figure 3.7. M&E Framework

selection on text-plain data).

The Visualization Services supply graphics and presentation APIs from local and remote locations that enable constructing integrated GUIs for SDN Mashups. Examples of Visualization Services are the Multi Router Traffic Grapher (MRTG) [74] for generating Web pages with images presenting the traffic of an SNMP-enabled NetDev, the Round Robin Database tool (RRDtool) [75] for displaying over time the performance data of NetDevs, the Yahoo Maps API [76] for showing the geographic location of several NetDevs, and the Google Visualization API [77] for supporting a flexible rendering of the information retrieved from any SDN managed resource.

Figure 3.8 depicts how internally the Service Container use the REpresentational State Transfer (REST) [78] architectural model to deploy the Services (*i.e.*, Management, Operation, and Visualization). Since every Service is based on REST, the Service Container establishes a Uniform Resource Identifier (URI) for pointing to each

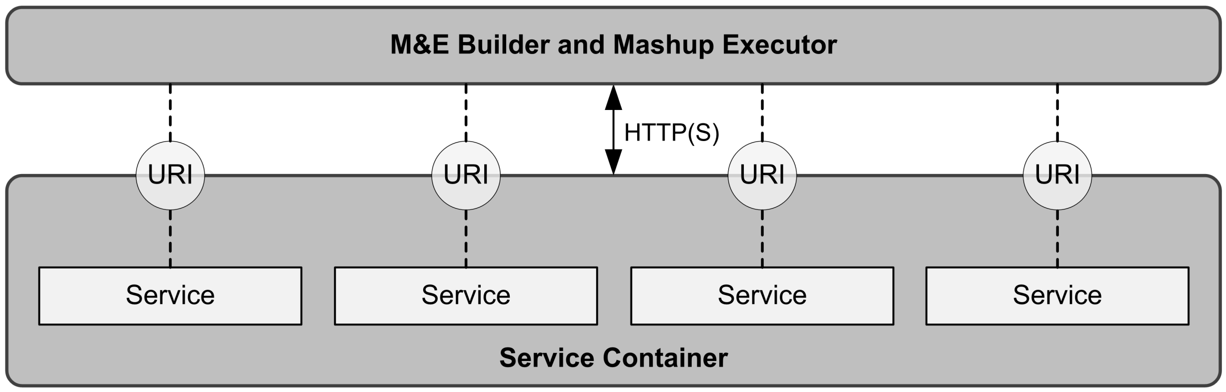


Figure 3.8. REST-based Service Container

Service. These Service URIs are invoked through HTTP(S) requests, such as GET and POST. The replies of Services are HTTP(S) responses, such as 200 OK and 404 Not Found. Thus, the Service Container provides a mediator channel in which the communication is conducted by following the request-response model of HTTP(S). This channel enables the interaction of the Service Container with both the M&E Builder and the Mashup Executor.

The Service Container responds to HTTP(S) requests as follows. First, the Service Container targets each request to the Service URI that points to the corresponding Service. Second, each Service carries out the requested functionality by performing its logic. Third, every Service encode results on JSON data and put such data on HTTP(S) responses. Fourth, the Service Container sends these HTTP(S) responses to the requesting modules (*i.e.*, the M&E Builder and the Mashup Executor).

Regarding the user interaction with the Service Container, first, the Network Administrator and the M&E Developer never access this module directly. Both users interact with the Service Container through the Presentation, the M&E Builder, and the Mashup Executor modules. Second, the Service Developer is responsible for implementing and updating Services by using a suitable IDE in order to extend and improve the Service Container. Details about IDEs for the Service Developer are out of scope of the M&E framework, and hence of this thesis.

In general terms, the M&E Builder *(i)* allows the Network Administrator and the M&E Developer to create and execute SDN Mashups (*i.e.*, static SDN Mashups) and *(ii)* automatically generates SDN Mashups according to recognized events (*i.e.*, dynamic SDN Mashups). The M&E Builder provides flexibility to the Manager component through a high-level abstraction of Management Services, Operation Services, and Visualiza-

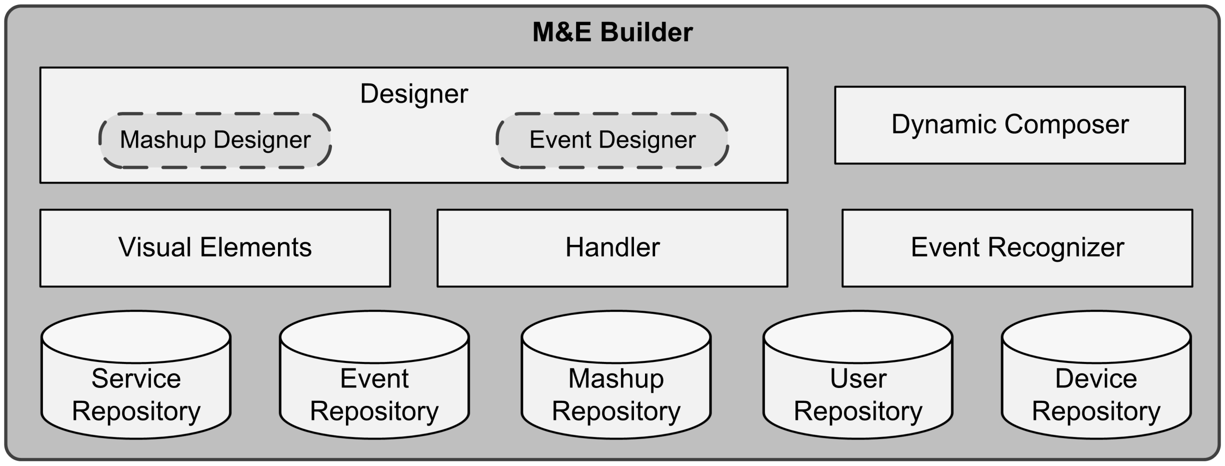


Figure 3.9. M&E Builder submodules

tion Services used to compose SDN Mashups. In this sense, the M&E Builder copes with the need of the Network Administrator and the M&E Developer to work with data mapping during the creation of SDN Mashups. It is to point out that the data mapping is one of the least intuitive tasks in current mashup composition systems because non-programmers—as the Network Administrator—are usually not able to specify it correctly. The M&E Builder (see Figure 3.9) defines the following submodules: the Service Repository, the Event Repository, the Mashup Repository, the User Repository, the Device Repository, the Visual Elements, the Handler, the Designer, the Event Recognizer, and the Dynamic Composer.

The Service Repository stores metadata that describes and points out the mashable entities offered by the Service Container (*i.e.*, the Management Services, the Operation Services, and the Visualization Services). Similarly to the Service Container, the Service Developer is also in charge of extending and keeping updated the Service Repository. The metadata of Services represented in the JSON format is `{"uri":` *URI_SERVICE*, `"service":` *NAME_SERVICE*, `"operations": [`*OPERATION*, `...,` *OPERATION*`]}`, where *URI_SERVICE* and *NAME_SERVICE* are the unique identifier and the name of the Service, respectively. The *OPERATION* array defines the collection of operations offered by the Service. Each *OPERATION* is `{"operation":` *OPERATION_NAME*, `"produce":` *OPERATION_RETURN*, `"parameters": [`*PARAMETER*, `...,` *PARAMETER*`]}`. Here *OPERATION_NAME* establishes the name of the operation, *OPERATION_RETURN* determines the data type that the operation produces, and the *PARAMETER* array specifies the name of the parameters required to invoke the corresponding operation.

The following example of metadada describes two operations provided by a particular Performance Service (*i.e.*, Management Service). The first operation retrieves the list of OpenFlow switches (*i.e.*, NetDevs) handled by a concrete OpenFlow Controller (*i.e.*, NOS). The second operation obtains the port statistics of a specific OpenFlow switch.

```
{
  "uri"        : "/pathToService",
  "service"    : "ofPerformanceService",
  "operations": [
    {
      "operation" : "getSwitchList",
      "produce"   : "json",
      "parameters": ["controllerIp", "controllerPort"]
    },
    {
      "operation" : "getSwicthPortStats",
      "produce"   : "json",
      "parameters": ["controllerIp", "controllerPort", "switchId"]
    }
  ]
}
```

The Event Repository houses patterns of the events designed by the M&E Developer. This framework relies on the model of network management situations proposed in [79] to characterize such event patterns as a combination of entities, attributes, and constraints. The metadata of events encoded in the JSON format is `{"complexEvent":` *COMPLEX_EVENT_NAME*, `"events": [`*EVENT*`, ..., `*EVENT*`]}` in which *COMPLEX_EVENT_NAME* represents the global name of a complex event formed by multiple simple events (*i.e.*, *EVENT* array). Each *EVENT* is `{"event":` *EVENT_NAME*, `"eac": [`*EAC*`, ..., `*EAC*`]}`. Here, *EVENT_NAME* defines the specific name of a simple event and the *EAC* array determines the collection of entities, attributes, and constraints involved in such simple event. Each *EAC* is `{"entity":` *ENTITY*, `"properties": [`*AC*`, ..., `*AC*`]}`, where each *AC* is `{"attribute":` *ATTRIBUTE*, `"constraint":` *CONSTRAINT*`}`. These last metadata present that each entity has a collection of attributes and their corresponding constraints. Note that such constraints establish conditions to detect when a particular event happens.

For example, the following metadata depicts two simple events composing a complex event. The first simple event represents an increase of transmitted dropped packages by any port of a NetDev (*e.g.*, an OpenFlow Switch). The second event describes a significant decrease of the traffic sent through any port of a NetDev. Separately, these simple events do not represent a critical situation in the managed SDN environment since *(i)* the drop of transmitted packages may occur because certain link has reached its bandwidth limit and *(ii)* the reduction of output throughput may be caused by the low traffic requirements of the network. Nevertheless, when both events (*i.e.*, complex event) happen on the same port of a NetDev, it represents a fail on the corresponding link because there is a drop of transmitted packages without using the maximum output throughput.

```
{
  "complexEvent": "failOnLink",
  "events"      : [
    {
      "event": "increaseTransmittedDroppedPackages",
      "eac"  : [
        {
          "entity"    : "netdev",
          "properties": [
            {
              "attribute" : "port",
              "constraint": "all"
            }, {
              "attribute" : "transmitedDroppedPackages",
              "constraint": "> 10%"}
          ]
        }
      ]
    }, {
      "event": "decreaseOutputThroughput",
      "eac"  : [
        {
          "entity"    : "netdev",
          "properties": [
            {
              "attribute" : "port",
              "constraint": "all"
            }, {
              "attribute" : "outputThroughput",
              "constraint": "< 80%"
            }
          ]
        }
      ]
    }
  ]
}
```

The Mashup Repository contains metadata of all SDN Mashups *(i)* statically designed by the Network Administrator and the M&E Developer and *(ii)* dynamically generated by the M&E System. Similarly to the event patterns, this framework employs the model of rich dynamic mashups for situation management outlined in [79] to define the metadata of SDN Mashups in JSON format as `{"id":` *ID_MASHUP*`, "name":` *NAME_MASHUP*`, "workflow":` *WORKFLOW*`, "events": [`*EVENT_REF*`, ..., `*EVENT_REF*`]}`. Here, *ID_MASHUP* represents the unique identifier of the SDN Mashup, *NAME_MASHUP* establishes the friendly name of the SDN Mashup, *WORKFLOW* describes the execution flow of the SDN Mashup, and each *EVENT_REF* references either the complex or simple events addressed by the SDN Mashup. Specifically, *WORKFLOW* is `{"elements": [`*ELEMENT*`, ..., `*ELEMENT*`], "links": [`*LINK*`, ..., `*LINK*`]}` in which the *ELEMENT* array determines the mashable elements that compose the SDN Mashup and the *LINK* array indicates the set of logical connections among these elements. Each *ELEMENT* is `{"id":` *ELEMENT_ID*`, "element":` *ELEMENT_NAME*`, "service":` *SERVICE_REF*`, "config":` *CONFIG*`}`, where *ELEMENT_ID* defines the unique identifier of the mashable element in the *WORKFLOW*, *ELEMENT_NAME* describes the global name of the element, *SERVICE_REF* targets the Service performed by the element, and *CONFIG* specifies *CONFIG_NAME*`:` *CONFIG_VALUE* pairs to provide the names and the values of the configuration parameters of the element. Each *LINK* is `{"source":` *ELEMENT_REF* `, "destination":` *ELEMENT_REF*`}`, where *ELEMENT_REF* establishes the identifiers of the source and destination elements that form the logical connection. Note that *WORKFLOW* define how SDN Mashups handle events that happen in the SDN environment.

The User Repository stores the profiles of users performing as the Network Administrator and the M&E Developer. The Publisher employs such user profiles for controlling and customizing the access to the M&E Builder. The Device Repository hosts information related to device capabilities. This information serves to identify the type of Client device (*i.e.*, Web Client and Mobile Client) that is able to run the M&E Builder and SDN Mashups.

The Visual Elements provide graphical mashable representations of the elements stored in the Service Repository and the Mashup Repository. Therefore, the Visual Elements defines two types: Visual Services and Visual Mashups. The Visual Services represent the Management Services, the Operation Services, and the Visualization Services offered by the Service Container. Alike the Service Container and the Service Repository, the Service Developer uses an IDE for programming and deploying such Visual Services. The Visual Mashups display graphic elements automatically generated by the Handler for each SDN Mashup saved in the Mashup Repository. The Visual Mashups facilitate reusing SDN Mashups to compose complex ones.

The Handler parses and manages (*i.e.*, search, create, update, and delete) *(i)* the metadata of events between the Event Repository and the Event Designer and *(ii)* the metadata of SDN Mashups between the Mashup Repository and the Mashup Designer. In addition, the Handler employs the metadata from the Mashup Repository to automatically generate Visual Mashups that allow to build more complex SDN Mashups by composing others.

The Designer is an end-user oriented interface made up of the Mashup Designer and the Event Designer. The Mashup Designer allows the Network Administrator and the M&E Developer to build, handle, and execute, in an easy way, SDN Mashups. To achieve this goal, the Mashup Designer supplies *(i)* the drag-and-drop mechanism to select Visual Elements (*i.e.*, *SERVICE_REF*), *(ii)* the wiring mechanism to combine Visual Elements (*i.e.*, *LINK*), *(iii)* the configuring mechanism to set the parameters of Visual Elements (*i.e.*, *PARAMETER* in *OPERATION*) and to determine the tackled events by SDN Mashups (*i.e.*, *COMPLEX_EVENT_REF*), *(iv)* the handling mechanism to call the Mashup Repository operations provided by the Handler (*e.g.*, search, load, save, and delete); and *(v)* the launching mechanism to invoke the Mashup Executor. In turn, the Event Designer enables the M&E Developer to characterize events that may occur in the managed SDN environment. In this sense, the Event Designer *(i)* offers an event pattern-based (*i.e.*, *EAC*) interface that facilitates defining such events and *(ii)* interacts with the Handler for managing the event patterns as metadata stored in the Event Repository.

Both the Event Repository and the Mashup Repository support the automatic and dynamic SDN Mashup generation process. This process depicts two phases based on concepts from rule-based systems: *(1)* the left hand side (*i.e.*, *when*) for the automatic recognition of events by using matching pattern algorithms stored in the Event Repository and *(2)* the right hand side (*i.e.*, *then*) for the dynamic composition of SDN Mashups by using the composition metadata stored in the Mashup Repository.

(1) *when* < *event* > — phase that automatically recognizes events.

(2) *then* < *sdnMashup*> — phase that dynamically composes SDN Mashups.

The Event Recognizer conducts the automatic recognition of events. Figure 3.10 presents the Event Recognizer as a two-part structure: the Sensor and the Pattern Matcher. The

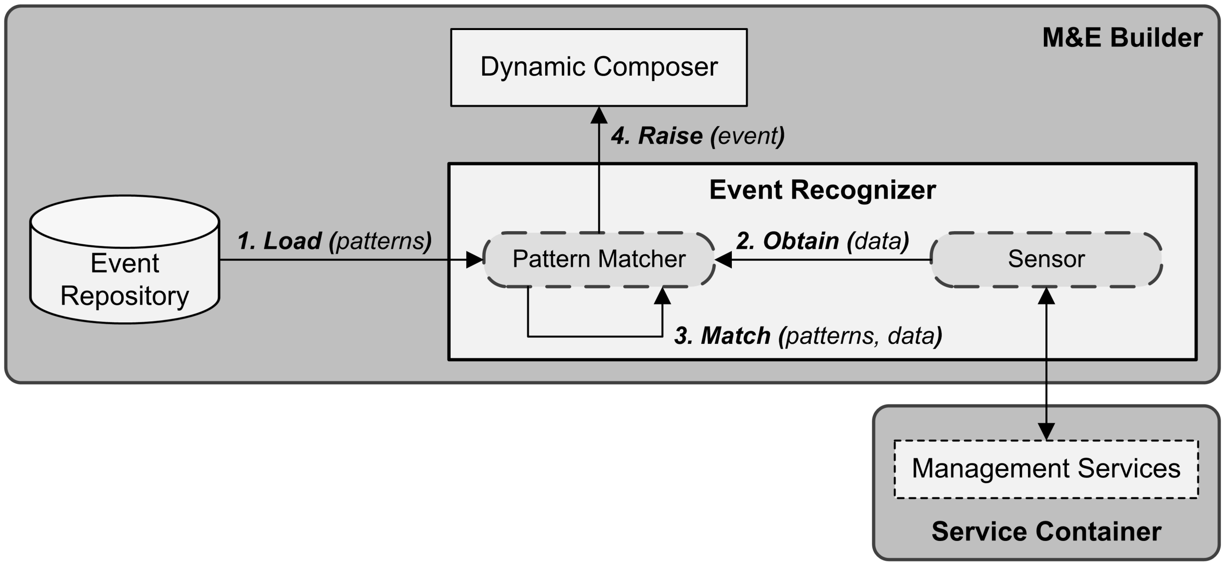


Figure 3.10. Automatic recognition of events

Sensor retrieves SDN management data by interacting with the Management Services and delivers such data as streaming to the Pattern Matcher. The retrieving of information depends on the communication model (*e.g.*, pull or push) provided by each Service.

The Pattern Matcher recognizes events as follows. First, it reads and loads event patterns from the Event Repository. Second, it obtains SDN management data from the Sensor. Third, it conducts matching operations—comparison of samples against patterns—between the SDN management data and the loaded event patterns. RETE [80] and PHREAK [81] are algorithms for carrying out such matching operations. Fourth, every time the Pattern Matcher detects a correlation, it raises the corresponding event to the Dynamic Composer. The metadata of this raised event contains specific information about the involved SDN managed resources (*i.e.*, *EAC*).

The Dynamic Composer performs the dynamic composition of SDN Mashups by automating the tasks that the Network Administrator and the M&E Developer carry out on the Mashup Designer. Specifically, the Dynamic Composer automates the mechanisms for selecting, combining, and configuring the Visual Elements that form the SDN Mashup. Figure 3.11 depicts the components of the Dynamic Composer: the Selector and the Generator. The Selector, first, receives the event from the Event Recognizer. Second, it selects from the Mashup Repository the more suitable SDN Mashup for addressing the received event. Since several SDN Mashups might point to tackle anal-

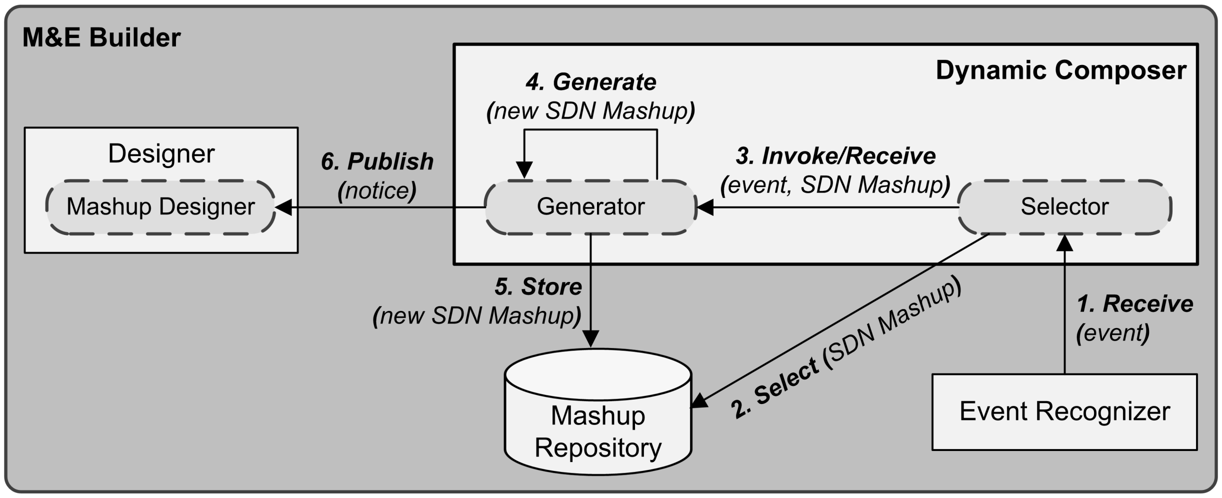


Figure 3.11. Dynamic composition of SDN Mashups

ogous events, the Selector employs the linguistic similarity algorithm [82] to calculate the highest similarity value between the received event and the set of events addressed by SDN Mashups. The linguistic similarity algorithm is based on ElementMatch [83], CheckSynonym [84], and NGram [85]. Third, the Selector invokes the Generator with the received event and the selected SDN Mashup as parameters.

The Generator operates as follows. First, it receives the event and the SDN Mashup from the Selector. Second, it generates a new instance of SDN Mashup by using the metadata of the event and the SDN Mashup received. In particular, the Generator reuses *WORKFLOW* and reconfigures it with the specific information provided by *EAC*. Third, the Generator stores in the Mashup Repository the generated SDN Mashup by writing the corresponding metadata, allowing the Network Administrator and the M&E Developer to interact (*e.g.*, reuse, improve, and launch) with such new SDN Mashup by means of the Mashup Designer. Fourth, the Generator publishes in the Mashup Designer a notice about the generation of a new SDN Mashup for addressing a recognized event.

It is important to highlight that the M&E Framework deploys a Event-Driven Architecture (EDA) [86] as following: *(i)* the Management Services act as the Event Generators, *(ii)* the interface provided by the Service Container to access the deployed Management Services represents the Event Channel, *(iii)* both the Event Recognizer and the Dynamic Composer perform the Event Processing operations; and *(iv)* the Mashup Designer carry out functionalities as the Downstream Event-Driven Activity.

The Mashup Executor executes SDN Mashups. The Mashup Executor comprises the Router and the Engine. The Router coordinates the execution flow of SDN Mashups (*i.e.*, *WORKFLOW*). On runtime, the Router *(i)* receives invocations from the Engine for selecting SDN Mashups that serve initial requests, *(ii)* selects and combines multiple Services and SDN Mashups to attend invocations, by reading metadata from the Service Repository and the Mashup Repository; and *(iii)* calls the Engine to request the instantiation of SDN Mashups and their constitutive elements.

The Engine is a lifecycle manager responsible for creating, deleting, and caching instances of SDN Mashups. For each initial request for executing an SDN Mashup from either the Web Client or the Mobile Client, the Engine invokes the Router. Subsequently, the Engine listens for indications from the Router to manage the instances of SDN Mashups and their constituent elements.

The Publisher adapts the GUI of each SDN Mashup to different Client devices (*i.e.*, the Web Client and the Mobile Client). Moreover, this module retrieves information from the User Repository to control and customize the access of end-users (*i.e.*, the Network Administrator and the M&E Developer) to functions and elements provided by the M&E Framework.

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 (WRE) and the latter a Mobile Web Runtime Environment (MWRE) to execute client-side mashup functionalities. Both types of Clients can execute SDN Mashups. Therefore, browsers running on personal computers, notebooks, and smartphones, act as front-ends of SDN management tools based on mashups. In return, the M&E Builder 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 M&E Builder deploy Web 2.0 user interfaces, Client devices must support JavaScript, Asynchronous Javascript And XML (AJAX), Cascading Style Sheets (CSS), HyperText Markup Language (HTML) version 5, among other technologies for supporting Web 2.0-based solutions.

Regarding the users, the M&E Framework considers three actors: the Service Developer, the M&E Developer, and the Network Administrator. The Service Developer represents an information technology programmer responsible for *(i)* developing and deploying Services and Visual Elements and, with this, *(ii)* extending and improving the Service Container, the Service Repository, and the M&E Builder. The Service Developer performs these tasks by using a suitable IDE. The M&E Developer is in charge of *(i)* composing, customizing, and extending SDN Mashups that carry out management tasks in the SDN environment, *(ii)* defining event patterns that allow the M&E Framework to automatically recognize events occurring in SDN managed resources, and *(iii)* configuring relationships between events and SDN Mashups that enable the M&E Framework to dynamically generate SDN Mashups that address specific events. The M&E Developer interacts with the M&E Framework via the Designer (*i.e.*, the Mashup Designer and the Event Designer) running on a Web Client. The Network Administrator is responsible for *(i)* performing management functions—including coping with recognized events—on the SDN environment by launching and using statically and dynamically generated SDN Mashups and *(ii)* composing, customizing, and extending such SDN Mashups. The Network Administrator interacts with the M&E Framework by means of SDN Mashups executed on either Clients (*i.e.*, the Web Client and/or the Mobile Client) and the Mashup Designer running on the Web Client.

3.4 Final remarks

This chapter presented a groundbreaking Management Plane approach that leveraged CIM and Web (2.0/3.0) technologies for carrying out an effective management of the virtual, dynamic, and heterogeneous environment of SDN. The contributions achieved in this thesis divides into two categories: conceptual and specific.

- Conceptual contributions.
  - The Management Plane that referenced the OSI network management submodels (*i.e.*, Information, Organizational, Communication, and Functional) to depict an integrated management architecture for the virtual, dynamic, and heterogeneous SDN environment.
  - The SDN Mashup concept that defined the use of the mashup technology for building and customizing, in a high-level abstraction, management tools in the SDN environment.
- Specific contributions.
  - The CIM-based Information Model that represented the entire SDN management environment as a technology-independent and consistent conceptual framework (*i.e.*, CIM Schemas) across distinct vendors and SDN instances.

- The M&E Framework that proposed a structure to deploy in the Manager component of the Management Plane the creation and execution of static and dynamic SDN Mashups.

Chapter 4

Evaluation

This chapter provides an extensive evaluation about the feasibility of employing the Management Plane based on a common Information Model and on Web technologies (2.0/3.0) for effectively carrying out management tasks in the virtual, dynamic, and heterogeneous environment of SDN. Specifically, this chapter presents two case studies for evaluating the main components of such Management Plane approach: the CIM-based Information Model and the M&E Framework.

4.1 Case study for the CIM-based Information Model

The case study for evaluating the proposed CIM-based Information Model, first, establishes a network management scenario that deploys different SDN management technologies. Second, it implements a system prototype that relies on the introduced Management Plane reference architecure. Third, it builds up a test environment based on the described scenario. Fourth, it conducts a late evaluation of the architecture components deployed for supporting the CIM-based Information Model. A late evaluation of a software architecture takes place when its implementation is complete [87]. The late evaluation is performance-based [88] and intented to determine the feasibility of using the introduced approach in terms of time-consuming, response time, and network traffic. The time-consuming is related to the process required to conduct configuring operations on heterogenous SDN resources. The response time and the network traffic correspond to the behavior on runtime of solutions used to configure the heterogeneous SDN environment.

4.1.1 Scenario

In this scenario, a Cloud Service Provider enables access to its cloud resources by deploying a basic SDN data center network: three tiers (*i.e.*, core, aggregation, and edge) of NetDevs handled by a *Current NOS* and arranged in a simple tree topology (*i.e.*, each NetDev has a single parent). Usually, the Network Administrator of such Cloud Service Provider purchases SDN resources from *Vendor A*. However, at some point of time, the Network Administrator decided to buy NetDevs from *Vendor B* because of economic profits. Now, the Network Administrator requires to configure each NetDev for being controlled by a *New NOS* that offers better performance, reliability, and security features.

Considering that *Vendor A* provides a different NetDev management interface than *Vendor B*, the Network Administrator typically would use an *Isolated Solution* (*i.e.*, *Vendor A* Tool and *Vendor B* Tool) to execute specific configuration commands on NetDevs from distinct vendors. This solution hinders and retards managing tasks of the Network Administrator. Instead, the Management Plane approach hides network heterogeneity by establishing a common NetDev configuration model and by adapting to each vendor management interface. Thus, the Management Plane affords an *Integrated Solution* that allows the Network Administrator to seamlessly configure every NetDev to be controlled by the *New NOS*, mitigating the complexity and time-consumption of managing hetereogeneous SDN resources.

Figure 4.1 illustrates the above described scenario. NetDevs provided by *Vendor A* are OpenFlow switches that enable the OVSDB management interface to accept configuration requests (*i.e.*, OVSDB switches). NetDevs from *Vendor B* are OpenFlow switches that run the OF-Config server to support configuration through the OF-Config protocol (*i.e.*, OF-Config switches). The *Current NOS* that initially handles the OpenFlow switches is a Floodlight controller. The *New NOS* that has to be set in the OpenFlow switches for controlling them is an Opendaylight controller. This scenario defines the *SetController* operation as the process of configuring a number of OVSDB switches and OF-Config switches for being controlled by the Opendaylight controller.

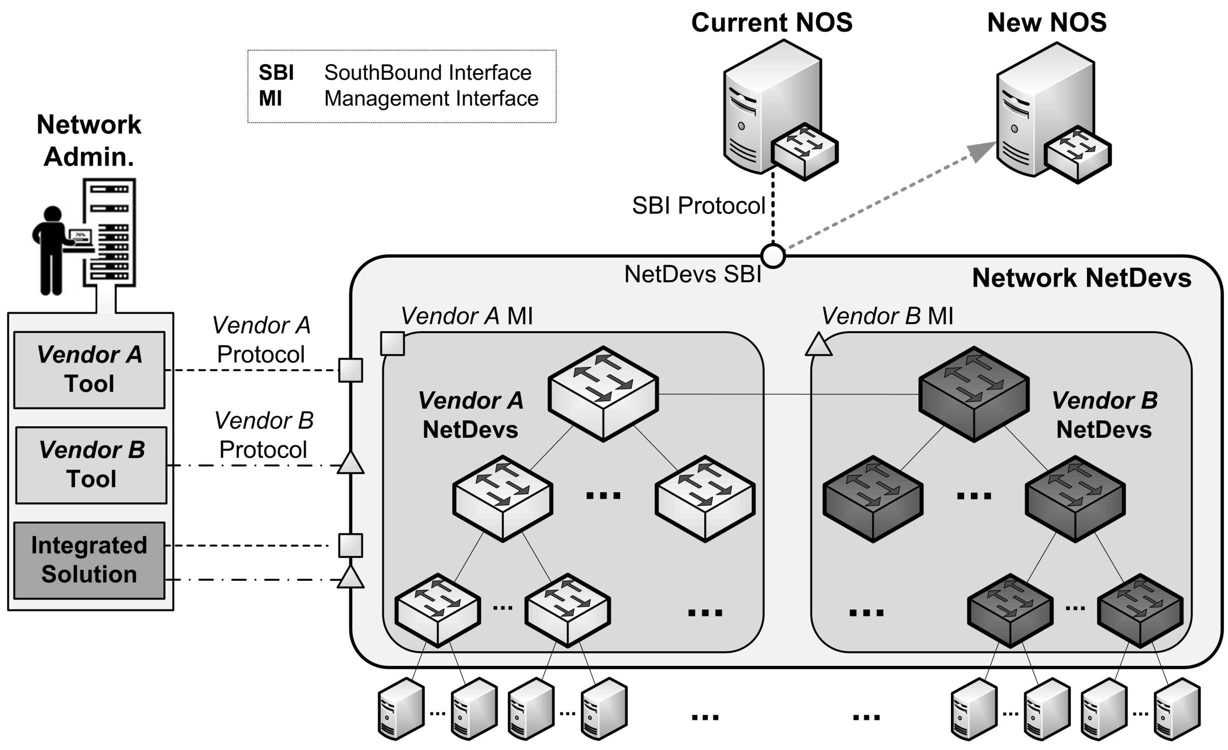


Figure 4.1. Scenario: configuring heterogeneous SDN resources

4.1.2 Reference implementation

To carry out the present case study, two prototypes were developed to conduct the *SetController* operation: the *Integrated Solution* and the *Isolated Solution*.

Integrated Solution

The *Integrated Solution* prototype relies on the reference architecture of the proposed Management Plane to perform *SetController* regardless the different configuration interfaces. Figure 4.2 depicts the implemented *Integrated Solution*. The Data Repository is a CIM Object Manager (CIMOM) that provides RRM as CIM schemas and stores instance data as CIM instances. CIMOM is the main component of a Web-Based Enterprise Management (WBEM) framework [89]. CIM schemas characterize the SDN managed environment, while CIM instances represent the SDN managed resources. To build RRM, this prototype compiled both the CIM Schema 2.18.1 and the SDN Extension Schema. The SDN Extension Schema implements the introduced CIM-based Information Model.

Since this prototype focuses on the *SetController* operation, the compiled SDN Ex-

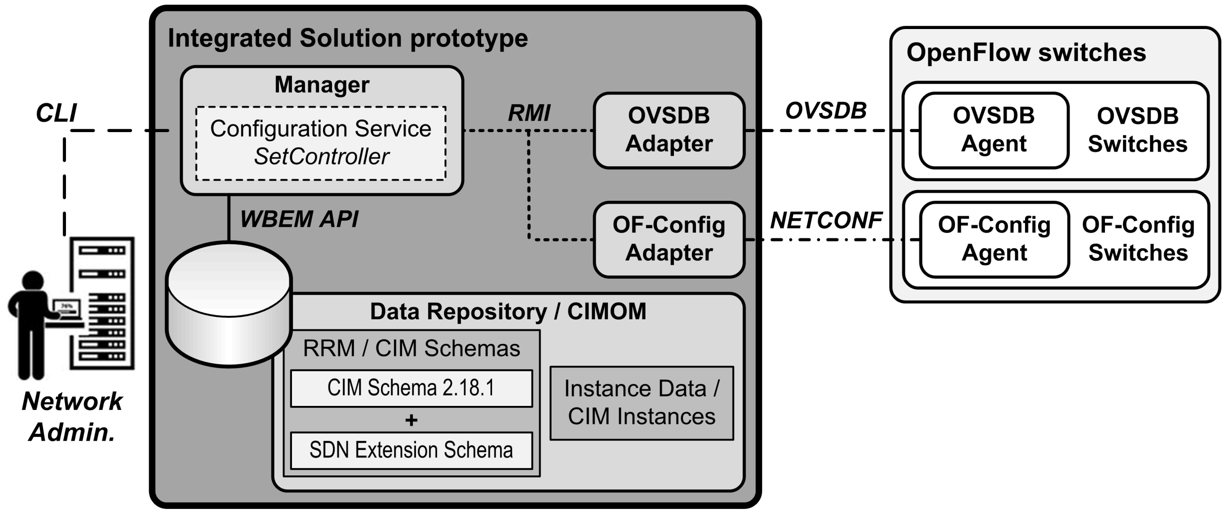


Figure 4.2. Prototype based on the Management Plane reference architecture

tension Schema is limited to the following extended classes from the CIM-based Information Model: *AgentService*, *NetDevAgentService*, *AdapterService*, *NetDevAdapterService*, *NetDevService*, and *SouthboundService*. In addition, CIMOM stores CIM instances of the above extended clases and of those included in the CIM Schema 2.18.1: *ComputerSystem*, *HostedService*, *RemotePort*, *ServiceSAPDependency*, *TCPProtocolEndpoint*, *ProvidesEndpoint*, *IPProtocolEndpoint*, and *BindsTo*. In addition, the *Integrated Solution* uses the Managed Object Format (MOF) to formally express the SDN Extension Schema and the CIM instances.

The Manager is carried out in a Java application. This Manager deploys the *SetController* operation as a Configuration Service. The User Interface of the Manager is a simple Command Line Interface (CLI) that allows executing the configuration commands of *SetController*. The Repository Interface uses the WBEM API to read and write instances stored in CIMOM. The Adapter Interface relies on the Remote Method Invocation (RMI) to communicate the Manager and the Adapters.

The Java application also deploys the NetDev Adapters: the OVSDB Adapter and the OF-Config Adapter. The OVSDB Adapter employs the Opendaylight OVSDB API to communicate with the OVSDB Agent that maintains the configuration database of OVSDB switches. The OF-Config Adapter relies on the NETCONF4J API to connect with the OF-Config Agent deployed by the OF-Config switches for accepting configuration requests. OF-Config utilizes NETCONF as the associated protocol.

Based on the information retrieved from CIMOM, the Manager invokes the appropiate Adapter. Once achieved the configuration in each requested OpenFlow switch, the Manager updates the instance data stored in CIMOM.

Isolated Solution

This prototype describes a classic solution of using a configuration tool for each management technology (*i.e.*, OVSDB and OF-Config) to perform operations as *Set-Controller*. Both the OVSDB Tool and the OF-Config Tool are bash scripts that automatize the usage of their underlying software. The OVSDB Tool uses the *ovs-vsctl* program to configure OVSDB switches. The OF-Config Tool employs the NETCONF client *netopeer-cli* to communicate with OF-Config switches. Both tools provide a simple CLI to specify the configuration parameters.

4.1.3 Test Environment

This case study was conducted in a test environment that allowed to deploy the above-mentioned scenario. Figure 4.3 depicts this environment formed by two OpenFlow networks, two OpenFlow controllers, the Manager Client, and CIMOM. Each OpenFlow network ran on an Ubuntu Server 14.04 Virtual Machine (VM) with 1 virtual processor and $1.5GB$ RAM assigned, both hosted by an Ubuntu 14.04 machine with $2.53GHz$ Intel Core i5 processor and $4GB$ RAM. Each VM executed Mininet 2.2.1, a software for emulating OpenFlow-based networks [90], to deploy a simple tree topology with 111 Open vSwitches 2.3.1. A tunnel over an IP network interconnected the root switches from each tree topology. The Open vSwitches used the OpenFlow protocol over a second IP network to communicate with an specific OpenFlow controller: Floodlight v1.0.1 or Opendaylight Helium. Later, the evaluation for each metric defines the exact quantity of switches per controller. The OpenFlow controllers operated on an Ubuntu 14.04 machine with $2.4GHz$ Intel Core 2 duo processor and $2GB$ RAM.

4.1.4 Time-consuming: results and analysis

The time-consuming (*i.e.*, t_C) is the time that the Network Administrator spends for carrying out management tasks on the SDN environment. Specifically for this case, t_C represents the time in seconds (s) that the Network Administrator takes to conduct the *SetController* operation by means of the CLIs provided by the *Integrated Solution*

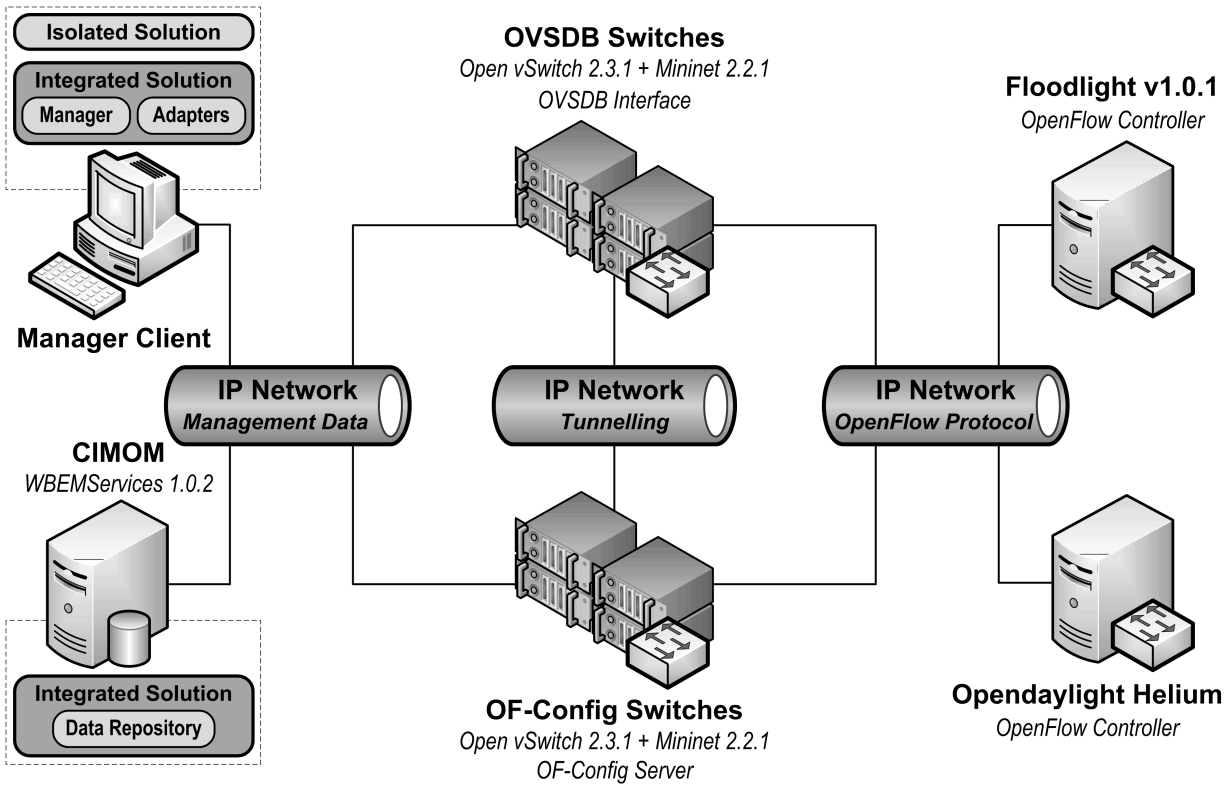


Figure 4.3. Test environment for the heterogeneous SDN configuration scenario

and the *Isolated Solution*. Certainly, the *Isolated Solution* aggregates an overhead (*i.e.*, $t_{C:oh:isolated}$) compared with the time consuming of the *Integrated Solution* (*i.e.*, $t_{C:integrated}$). This is because the *Isolated Solution* forces the Network Administrator to decide which tool must use to configure each OpenFlow switch. Unlike this, the *Integrated Solution* abstracts the heterogeneity of the configuration technologies of the OpenFlow switches.

Computing the time-consuming was realized using the Keystroke-Level Model (KLM) [91] because it is useful to estimate the time that an expert user (*i.e.*, the Network Administrator) spends to accomplish a routine task (*i.e.*, perform *SetController* on the CLI) supported on computer keyboard and mouse. In KLM, each task is modeled as a sequence of actions. Table 4.1 presents the original KLM-actions [91] and some helpful extensions [92] found in the literature.

Compared with the *Isolated Solution*, in the best case, the Network Administrator carries out on the *Isolated Solution* the following additional actions: *(i)* change once from one tool to another by pressing ALT and TAB keys, and *(ii)* decide which tool must use for each switch. Based on these actions, $t_{C:oh:isolated}$ is proportional to the number

Table 4.1. KLM actions

Action	Description	Time-average
K	Press and release a key	$0.2s$
$NC * K$	Type a string	$NC * 0.2s$
P	Point the mouse	$1.1s$
B	Hold or release the mouse	$0.1s$
H	Move the hand from mouse to keyboard	$0.4s$
M	Mentally preparing for executing physical actions	$1.35s$
Dnd	Drag-and-drop a visual element	$1.3s$
$Wire$	Wire two visual elements	$4.1s$

of configured switches (*i.e.*, n_{Sw}): $t_{C:oh:isolated} = K + M * n_{Sw} = 0.4 + 1.35 * n_{Sw}$.

4.1.5 Response time: results and analysis

Continuing the evaluation, it was measured the response time of the *Integrated Solution* and the *Isolated Solution* when used in the test environment (see Figure 4.3) to carry out the operation *SetController*. This response time represents the time in seconds (s) measured since the Network Administrator executes the operation *SetController* on the Manager Client until receiving the reply of the last configured switch. Nevertheless, since a configuration reply is received for each switch, a per switch basis analysis is also provided. This evaluation was conducted by using different amounts of configured switches: 2, 20, 50, 100, 150, and 200. Half were OVSDB Switches, and the other half were OF-Config Switches. In this and the following evaluations involving average values for time and network traffic, 30 measurements were took with a 95% confidence level.

Figure 4.4 depicts the response time results. Considering that the response time (r in s) of Web systems can be ranked as optimal ($r \leq 0.1$), good ($0.1 < r \leq 1$), admissible ($1 < r \leq 10$), and deficient ($r > 10$) [93], these results reveal: *(i) SetController* of both the *Isolated Solution* and the *Integrated Solution* has an admissible r that grows

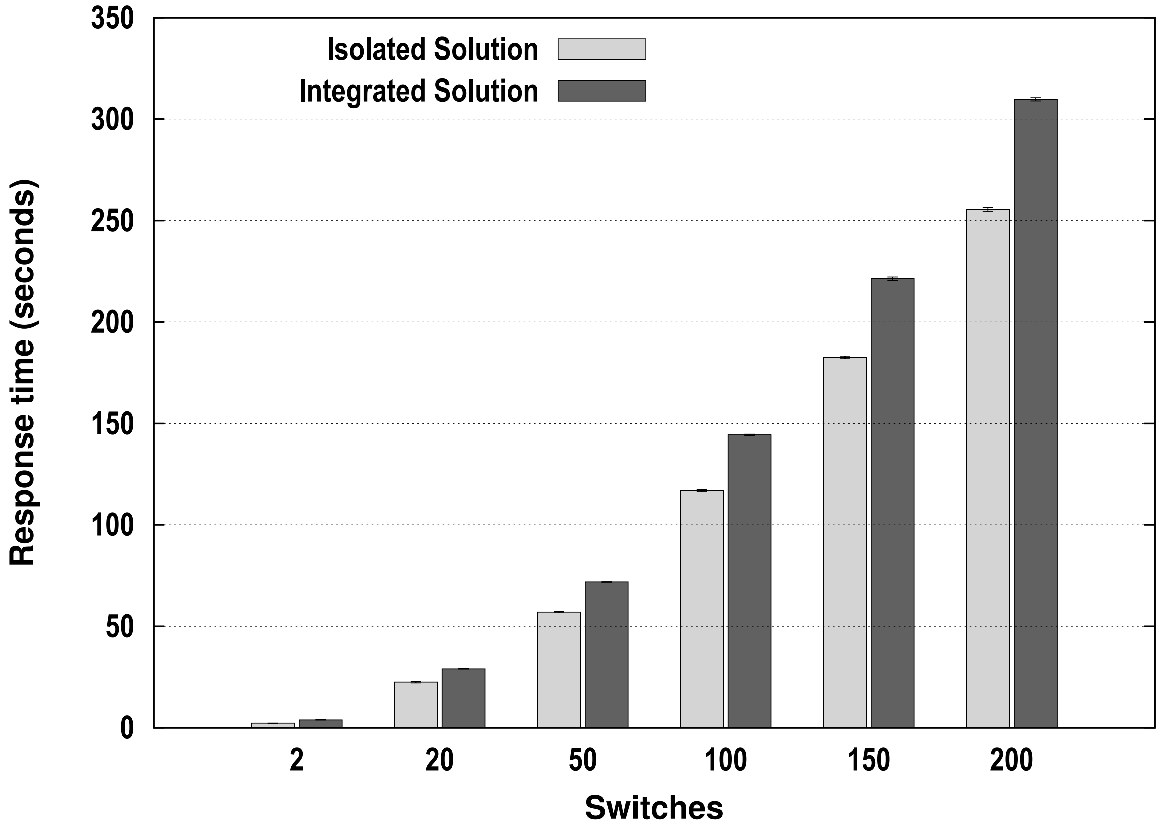


Figure 4.4. Response time on conducting the *SetController* operation

moderately (less than $1.5s$ and $1.8s\ per\ switch$, respectively) when the number of configured switches increases, *(ii)* the *Integrated Solution* takes longer than the *Isolated Solution*, as expected for using additional components (*e.g.*, CIMOM and Adapters) to cope with the heterogeneity; and *(iii)* the response time overhead per switch of the *Integrated Solution* is $0.8s$ for 2 switches and less than $0.35s$ for 20 switches or more. Based on the above results, the response time overhead of the *Integrated Solution* (*i.e.*, $t_{R:oh:integrated}$) depends on n_{Sw} (*i.e.*, the amount of configured switches) as shown in Equation 4.1. Furthermore, the results obtained for the time response and the previously estimated results for the time-consuming reveal that $t_{R:oh:integrated}$ is always less than $t_{C:oh:isolated}$.

$$t_{R:oh:integrated} \begin{cases} \leq 0.8 * n_{Sw} & , n_{Sw} < 20 \\ \leq 0.35 * n_{Sw} & , n_{Sw} \geq 20 \end{cases} \tag{4.1}$$

Summing up, although the *Integrated Solution* includes additional modules (*e.g.*, CIMOM and adapters) to cope with the complexity of managing heterogeneous SDN environments, it introduces a response time overhead shorter than the time-consuming overhead of the *Isolated Solution*. Certainly, the difference between the time overheads increases as more switches and distinct technologies incorporate the SDN managed environment. Additionally, considering the total time (t_T) as the sum of the response time and the time-consuming, the difference between the time overheads demonstrates that the *Integrated Solution* reduces the time for carrying out the operation *SetController*, as can be seen in Equation 4.2. Therefore, the response time results corroborate that, in terms of such metric, it is feasible to use the proposed approach for executing management operations like the proved *SetController*.

$$\begin{gathered}
t_{T:integrated} = t_{R:integrated} + t_{C:integrated} \\
t_{T:isolated} = t_{R:isolated} + t_{C:isolated} \\[2ex]
t_{R:integrated} = t_{R:oh:integrated} + t_{R:isolated} \\
t_{C:isolated} = t_{C:oh:isolated} + t_{C:integrated} \\[2ex]
t_{T:integrated} = t_{C:isolated} + t_{R:isolated} + t_{R:oh:integrated} - t_{C:oh:isolated} \\[2ex]
t_{T:integrated} \begin{cases} t_{T:isolated} - 0.4 - 0.55 * n_{Sw} & , n_{Sw} < 20 \\ t_{T:isolated} - 0.4 - n_{Sw} & , n_{Sw} \geq 20 \end{cases}
\end{gathered} \tag{4.2}$$

4.1.6 Network traffic: results and analysis

To continue the evaluation, it was proceeded to measure the network traffic generated by the *Integrated Solution* and the *Isolated Solution* when carrying out the *SetController* operation in the deployed test environment (see Figure 4.3). Such network traffic is the amount of data in kilobytes (kB) transmitted and received by the network interface of the Manager Client. Similarly to the response time, the number of configured switches for this evaluation varied among 2, 20, 50, 100, 150, and 200, where half were OVSDB Switches and the other half were OF-Config Switches.

Figure 4.5 presents the network traffic results. These results reveal: *(i)* the traffic generated by *SetController* on both the *Isolated Solution* and the *Integrated Solu-*

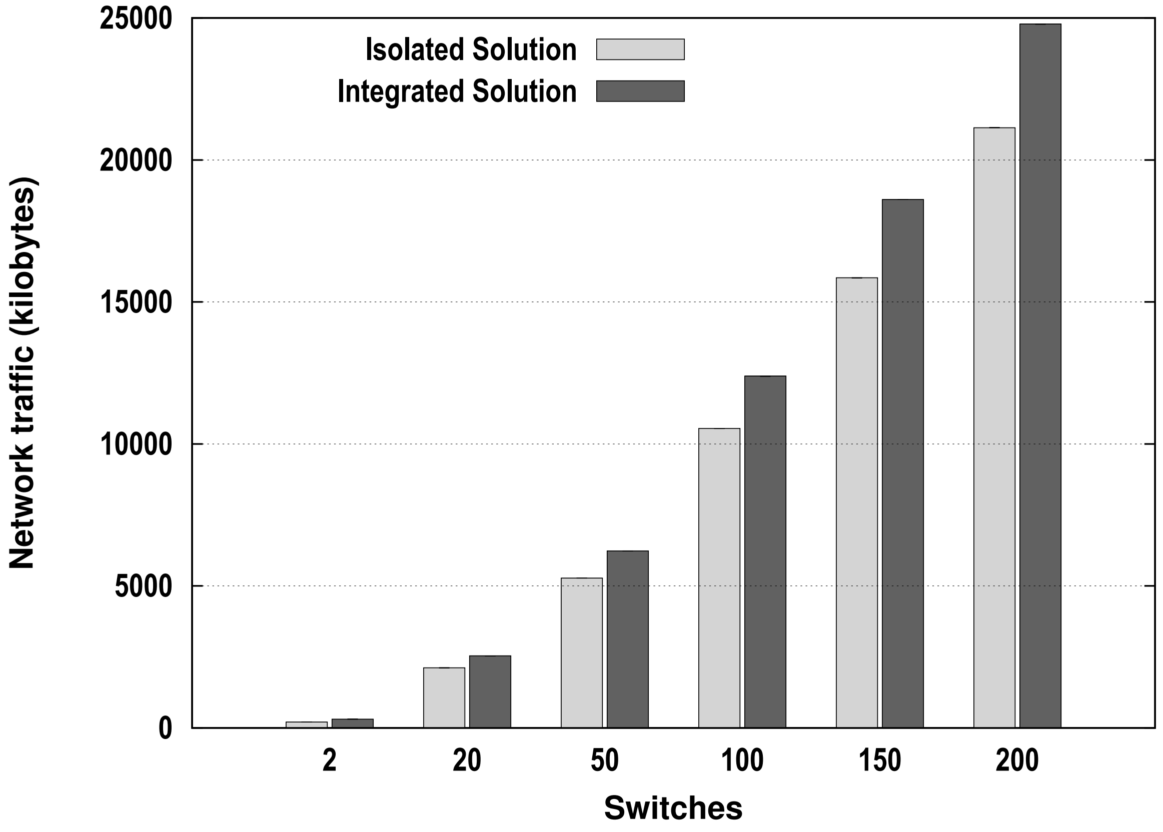


Figure 4.5. Network traffic on conducting the *SetController* operation

tion grows moderately (approximately $106kB$ and $124kB\ per\ switch$, respectively) when the number of configured switches increases, *(ii)* the *Integrated Solution* generates more traffic than the *Isolated Solution*, as expected for handling managed information of switches in CIMOM; and *(iii)* the additional traffic generated by the *Integrated Solution* is 32% for 2 switches and less than 17% for 20 switches or more. Considering that the *Integrated Solution*, unlike the *Isolated Solution*, copes with the heterogeneity of SDN environments by operating with standardized management data, the above facts corroborate that *SetController* of the *Integrated Solution* has a good behavior on network traffic.

Regarding the results obtained in the response time and network traffic evaluation of the operation *SetController*, it is important to mention: *(i)* approximately 94% of the response time of *Isolated Solution* corresponds to the OF-Config Tool, *(ii)* the OVSDB Tool generated about 87% of the network traffic of *Isolated Solution*; and *(iii)* the response time and network traffic overheads introduced by the *Integrated Solution* is smaller for many switches than for a few; this is because the connection and authentication with CIMOM was realized just once for any number of configured switches. Summarizing, the time and traffic results demonstrated that, in terms of such metrics, it is feasible to use the Management Plane along with the CIM-based Information Model in heterogeneous SDN environments to perform operations as the executed *SetController*.

4.2 Case study for the M&E Framework

To assess the M&E Framework, first, this case study outlines a network management scenario that deploys virtual SDN resources using different technologies. Second, it implements an M&E System prototype as an instance of the proposed framework. Third, it builds a test environment regarding the explained scenario. Fourth, similarly to the previous case study, it conducs a performance-based late evaluation of the M&E Framework in order to determine its feasibility in terms of time-recognition, time-composition, time-consuming, response time, and network traffic. The time-recognition and the time-composition correspond to the behavior for recognizing events and dynamically composing SDN Mashups, respectively. The time-consuming is associated with the process that the Network Administrator requires for performing and coping with monitoring requirements. The response time and the network traffic are related to the behavior on runtime of solutions used to monitor heterogeneous SDN resources.

4.2.1 Scenario

This scenario assumes that a Network Administrator manages an SDN-based Network Operator that provides virtual network infrasctructure to Small and Medium Enterprises (SME) by using SDN resources, such as OpenFlow controllers (*i.e.*, NOS) and virtual switches (*i.e.*, NetDev), supplied by several Virtual Network Providers (VNP). In this sense, the Network Administrator is responsible for monitoring the virtual SDN resources provisioned by each VNP, since, for example, an abrupt performance degradation in virtual links of one or more SMEs (*e.g.*, caused by unidentified errors in the virtual switches of VNPs) might lead the operator to break the Service Level Agreements established with such SMEs and, as a result, lose money.

Figure 4.6 shows the above described scenario. Considering that each VNP uses distinct technologies for deploying virtual SDN resources (*i.e.*, Beacon Controller, POX

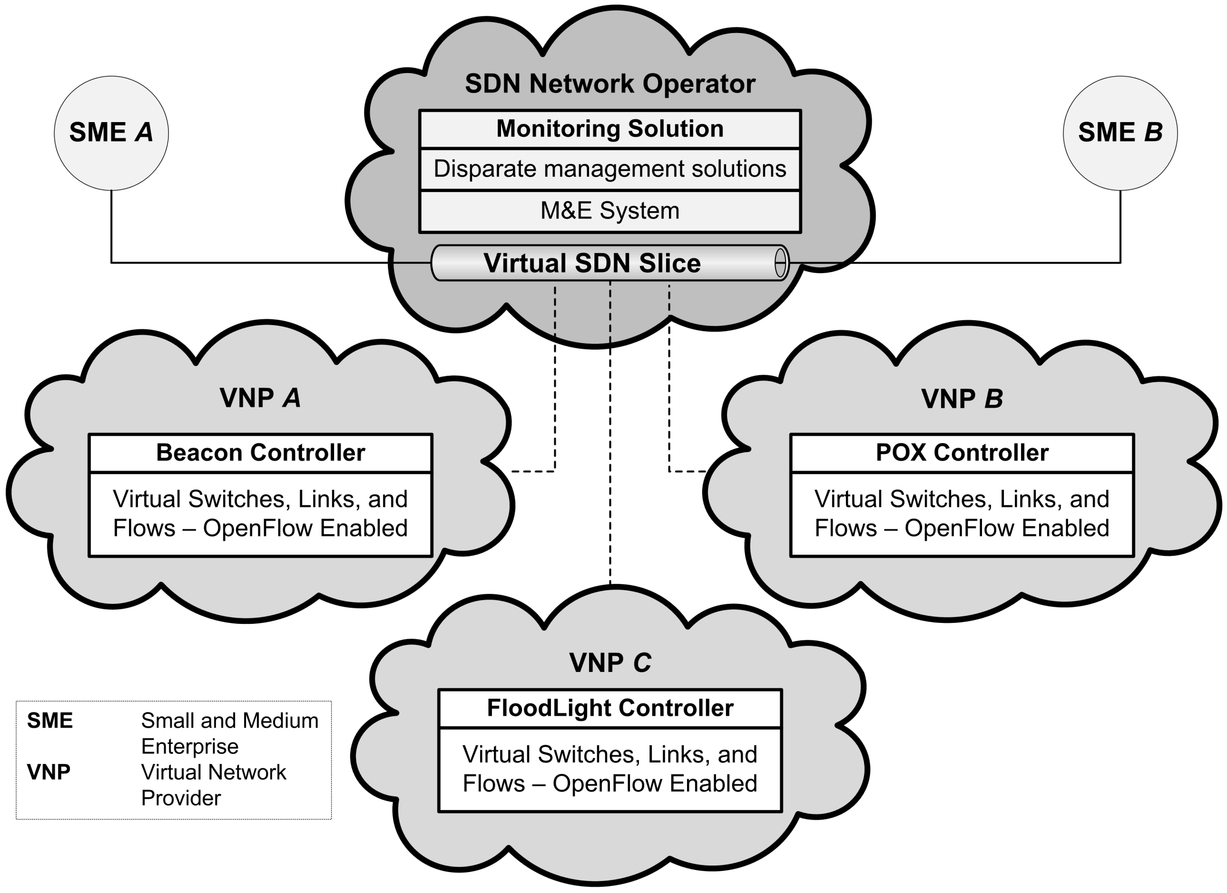

SDN Network Operator
Monitoring Solution
Disparate management solutions
M&E System
Virtual SDN Slice
SME *A*
SME *B*
VNP *A*
Beacon Controller
Virtual Switches, Links, and Flows – OpenFlow Enabled
VNP *B*
POX Controller
Virtual Switches, Links, and Flows – OpenFlow Enabled
VNP *C*
FloodLight Controller
Virtual Switches, Links, and Flows – OpenFlow Enabled
SME Small and Medium Enterprise
VNP Virtual Network Provider

Figure 4.6. Scenario: monitoring heterogeneous virtual SDN resources

Controller, and Floodlight Controller), the Network Administrator usually employs disparate management solutions (*e.g.*, proprietary CLIs and administrative Web interfaces) for monitoring the virtual network infrastructure. Instead, the M&E Framework allows the Network Administrator to build, in a high-level abstraction, own management tools (*i.e.*, SDN Mashups) that carry out monitoring tasks on such virtual SDN resources. For example, a SDN Mashup that display NOS performance data of all VNPs in an understandable, friendly, and integrated way. Therefore, the M&E Framework enables the Network Administrator to easily and rapidly conduct management tasks and cope with events on the virtual network infrastructure.

4.2.2 Reference Implementation

Figure 4.7 depicts the M&E System prototype. The prototype was built upon the M&E Framework and deployed by using a MySQL Server and an Apache Tomcat

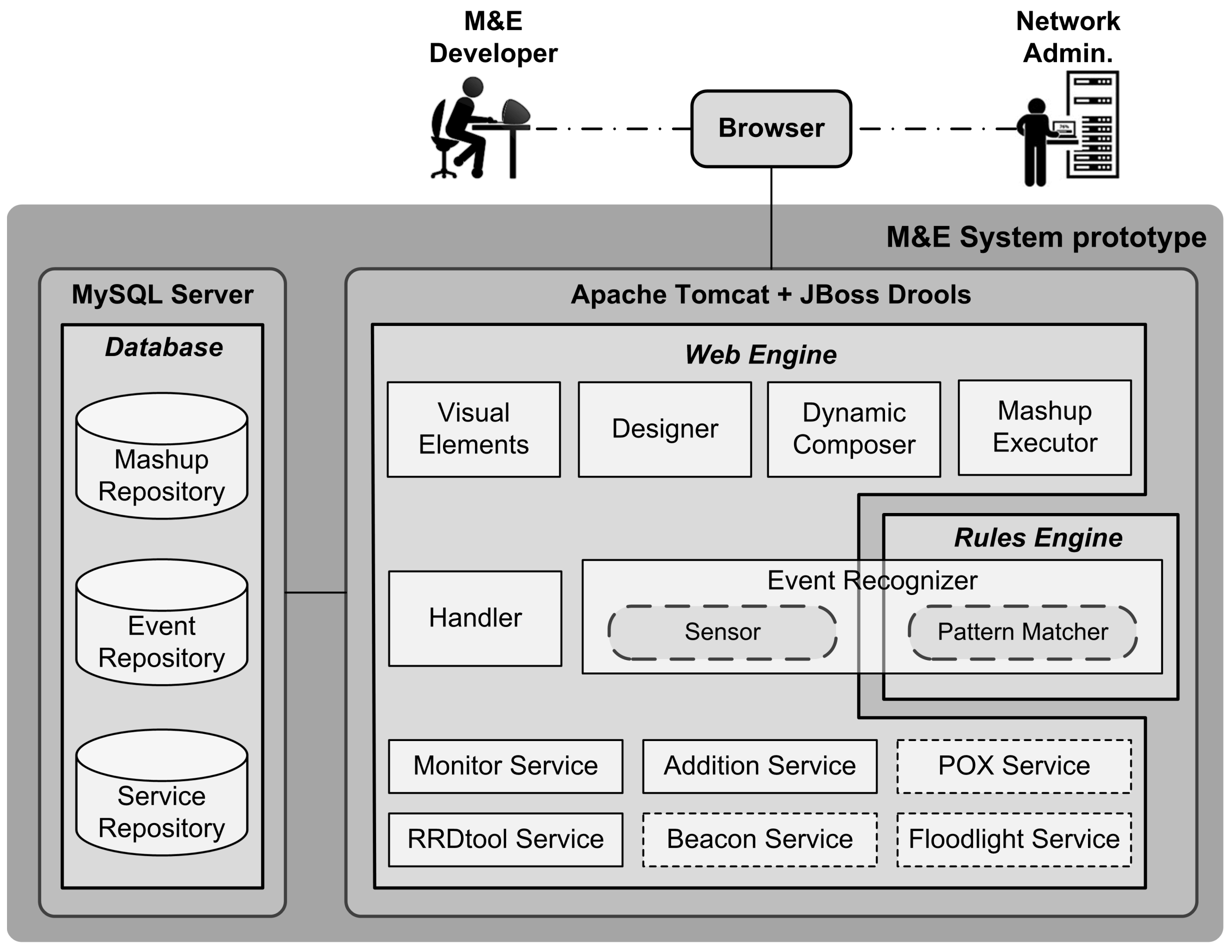


Figure 4.7. M&E System prototype

Server. The MySQL Server hosts the Database that implements the Mashup Repository, the Event Repository, and the Service Repository. The Apache Tomcat Server provides a Web Engine and uses the Spring API provided by JBoss Drools to integrate a Rules Engine.

The Web Engine displays through a browser-based interface the Designer (*i.e.*, the Mashup Designer and the Event Designer), the Visual Elements, and the results of the Mashup Executor (*i.e.*, executed SDN Mashups). This Web Engine relies on HTML5, JavaScript, Java Servlets, and AJAX for granting interactive and dynamic interaction among the Designer, the Visual Elements, the Handler, the Mashup Executor, and the Services.

The Designer incorporates the Yahoo User Interface (YUI) API and the WireIt API to assist *(i)* the M&E Developer and the Network Administrator for composing, reusing, and launching static and dynamic SDN Mashups and *(ii)* the M&E Developer for defining event patterns. The Visual Elements are JavaScript representations that allow the M&E Developer and the Network Administrator to select and combine Services and SDN Mashups on the Designer. The Handler is a Java-based application for managing the Mashup Repository, the Event Repository, and the Service Repository. The Mashup Executor is a Web application that implements the Router and the Engine. Internally, the Mashup Executor invokes by HTTP(S) the Services that form SDN Mashups. The Web Engine displays the results of these SDN Mashups on the browser-based interface by using graphics APIs.

The M&E System prototype deploys six Services: the Beacon Service, the POX Service, the Floodlight Service, the Addition Service, the Monitor Service, and the RRDtool Service. The Beacon Service, the POX Service, and the Floodlight Service are Management Services for carrying out monitoring fuctions on OpenFlow Controllers (*i.e.*, Beacon, POX, and Floodlight, respectively). For the sake of evaluating the M&E Framework, these Management Services directly communicate with the OpenFlow Controllers (*i.e.*, SDN managed resources) instead of interacting with the components and interfaces defined by the Management Plane.

The Addition Service is an Operation Service that aggregates data from two or more Management Services. This service checks that every input data holds the same format and produces an output data with such format. The Monitor Service and the RRDtool Service are instances of Visualization Services that provide graphics APIs to display management information. The Monitor Service relies on the Google Visualization API and the RRDtool Service provides a local implementation of the RRD4J API.

The Web Engine also deploys the Dynamic Composer and the Sensor component of the Event Recognizer. The Dynamic Composer is a Java-based application that customizes SDN Mashups. It is noteworthy that, first, the M&E Developer and the Network Administrator build and configure these SDN Mashups to address specific events. Second, such SDN Mashups are stored in JSON format in the Mashup Repository. Third, once an SDN Mashup has been dynamically generated, it is also stored using JSON in the Mashup Repository and, further, automatically exposed in the Designer as a Visual Element (*i.e.*, Visual Mashup).

The Sensor is a Java-based application that calls and listens to the Services for retrieving management data. This Sensor delivers such data to the Pattern Matcher deployed by the Rules Engine. The Pattern Matcher is also a Java-based application that employs the PHREAK algorithm implemented by JBoss Drools for recognizing events. Furthermore, this Pattern Matcher translates from JSON to the Drools Rule Language (DRL) the event patterns stored in the Event Repository—JBoss Drools only understands DRL.

4.2.3 Test environment

Figure 4.8 depicts the test environment deployed for conducting this case study. In such environment, the M&E System prototype ran on a Apache Tomcat 7.0.26 (*i.e.*, Web Engine) that integrated a JBoss Drools 6.1 (*i.e.*, Rules Engine) by means of the Spring API. The Database operated on a MySQL 5.5 Server. An Ubuntu 12.04 machine with $2.53GHz$ Intel Core i5 processor, $4GB$ RAM, and $250GB$ hard disk, hosted the M&E System and the Database. In turn, the applications used to perform the evaluation ran on an Ubuntu 12.04 Test Client with $2.4GHz$ Core 2 duo processor and $2GB$ RAM.

Based on the scenario, the test environment also included three OpenFlow-based networks (*i.e.*, in reference to SDN resources from VNPs) each controlled by different OpenFlow controllers: Beacon 1.0.2, POX 1.0.0, and Floodlight 0.9. These OpenFlow-based networks deployed a lot of Open vSwitches 1.4; later each evaluation case defines the exact quantity of switches per network. The OpenFlow controllers operated on an Ubuntu 12.04 machine with $2.33GHz$ Core 2 duo processor and $2GB$ RAM. In turn, the Open vSwitches were executed using Mininet 2.2.1 on an Ubuntu 12.04 server with $3.4GHz$ Core i7 processor and $8GB$ RAM.

4.2.4 Time-recognition: results and analysis

The time-recognition is the time in milliseconds (ms) that the M&E System takes for recognizing event patterns. To measure the time-recognition, this evaluation used the OpenFlow-based networks controlled by POX and Floodlight (see Figure 4.8). Each OpenFlow controller handled a datacenter network topology with 259 switches distributed in four levels of depth (*i.e.*, layers of access, aggregation, core, and edge) and 6 servers per rack. Thus, in total, the test environment for evaluating the time-recognition deployed 2 controllers, 518 switches, and 3626 ports.

Using the Event Designer, an event pattern was defined to detect when any port of any switch controlled by either POX or Floodlight had more than 5% of dropped packets. It is important to highlight that, first, the M&E System performs the translation from

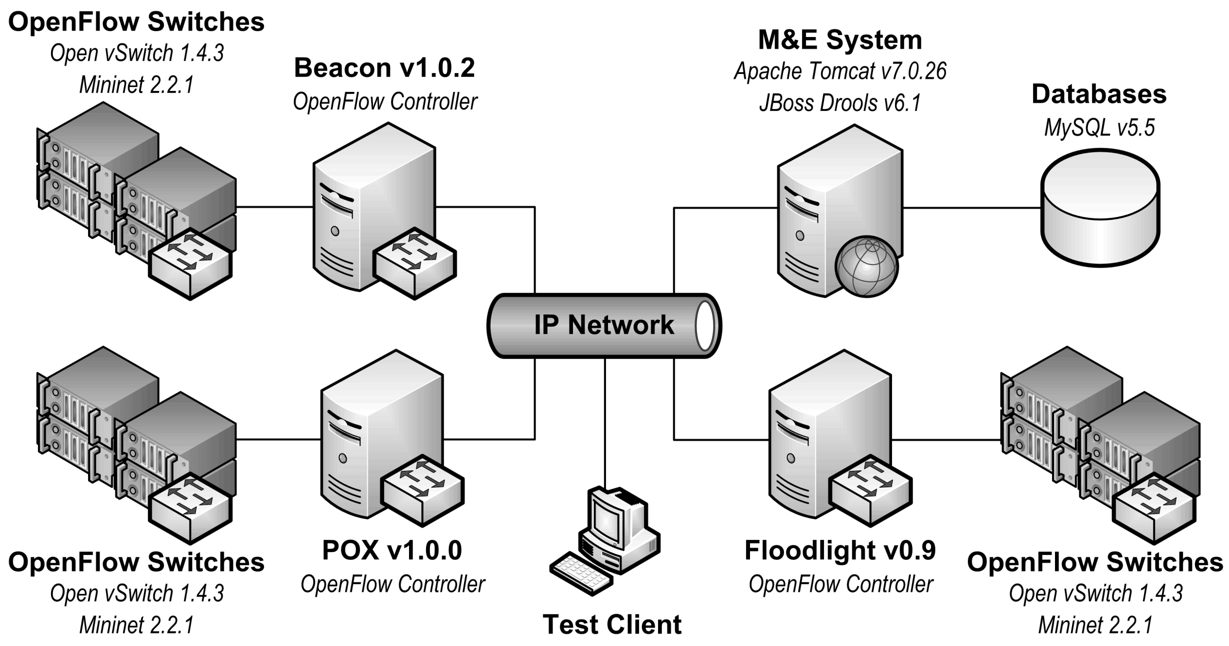


Figure 4.8. Test environment for the heterogeneous SDN monitoring scenario

JSON to DRL. Second, such translation is always hidden for the Network Administrator and the M&E Developer. Following, the event pattern encoded using the JSON and DRL formats.

JSON

```
{
  "complexEventName": "testEvent",
  "events"          : [
    {
      "eventName": "increaseTransmittedDroppedPackages",
      "eac"      : [
        {
          "entity"    : "openflowController",
          "properties": [
            {
              "attribute" : "ip",
              "constraint": "143.54.12.210 or 190.5.203.123"
            }, {
              "attribute" : "port",
              "constraint": "8080"
            }, {
              "attribute" : "type",
              "constraint": "pox or floodlight"
            }, {
              "attribute" : "openflowElement",
              "constraint": "openflowSwitch"
            }
          ]
        }, {
          "entity"    : "openflowSwitch",
          "properties": [
            {
              "attribute" : "dpid",
              "constraint": "all"
            }, {
              "attribute" : "openflowSwitchComponent",
              "constraint": "openflowPort"
          ]
        }, {
          "entity"    : "openflowPort",
          "properties": [
            {
              "attribute" : "number",
              "constraint": "all"
            }, {
              "attribute" : "percentTransmittedDropped",
              "constraint": "> 5"
          ]
        }
      ]
    }
  ]
}
```

DRL

```
package net.mashment.drools.rules
import net.mashment.drools.entities.*
import net.mashment.drools.DynamicMashmentComposer
rule "test"
  when
    $e0 : OpenflowController(
              ip == "143.54.12.210" || == "190.5.203.123",
              port == "8080", type == "pox" || == "floodlight")
    $e1 : OpenflowSwitch(openflowController == $e0)
    $e2 : OpenflowPort(openflowSwitch == $e1,
              percentTransmittedDropped > 5)
  then
    DynamicComposer($e0,$e1,$e2)
end
```

The time-recognition evaluation, first, loaded one single event pattern (*i.e.*, just one rule in the Rule Engine) and varied the quantity of generated events from 250 to 1750 in each OpenFlow-based network. Thus, the total number of generated events ranged from 500 to 3500. Second, this evaluation varied the number of loaded rules from 1

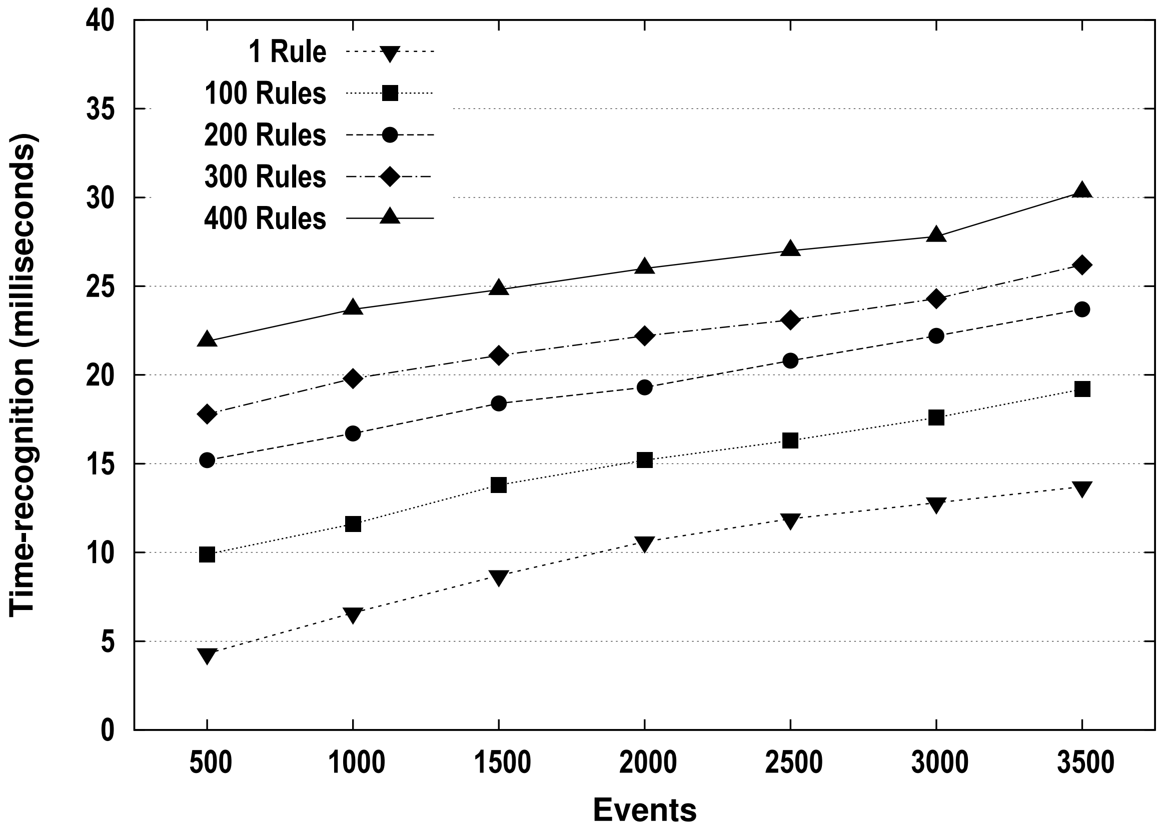


Figure 4.9. Time-recognition behavior

to 400; for each variation about the quantity of rules, the amount of generated events ranged from 500 to 3500.

Figure 4.9 presents the time-recognition results. These results reveal that the M&E System is able to recognize event patterns in a short time. The worst behavior took approximately $30.3ms$ to identify 3500 events with 400 loaded rules/patterns. Furthermore, the time-recognition increases linearly in a negligible way with the growth of events and loaded rules. Consequently, this evaluation demonstrates that, in terms of time-recognition, it is feasible to deploy the M&E Framework for effectively dealing with events that happen in the SDN environment.

4.2.5 Time-composition: results and analysis

The time-composition is the time in milliseconds (ms) that the M&E System takes for dynamically composing SDN Mashups. This evaluation measured the time-composition by using the OpenFlow-based networks handled by Beacon, POX, and Floodlight (see Figure 4.8). Each controller was in charge of handling a datacenter network topology with 259 switches distributed in four levels of depth and 6 servers per rack. Thus, in total, the test environment for evaluating the time-composition deployed 3 controllers, 777 switches, and 4662 ports.

By using the Mashup Designer, several SDN Mashups were built and configured for carrying out monitoring tasks when any port of any switch handled by either Beacon, POX, or Floodlight had more than 5% of dropped packages. Following, a snippet of the *WORKFLOW* in JSON format of a static SDN Mashup that lacks configuration parameters and the corresponding customization conducted by the M&E System to generate a dynamic SDN Mashup.

SDN Mashup statically created without configuration parameters

```
{
  "elements": [
    {
      "id": 1,
      "element": "Monitor",
      "service": "monitorService",
      "config" : {
        "position": [298,109],
        "input"   : "[wired]"
      }
    }, {
      "id": 2,
      "element": "Openflow Controller",
      "service": "",
      "config" : {
        "position"   : [35,113],
        "ip"         : "",
        "port"       : "",
        "type"       : "",
        "eventParams": ""
      }
    }
  ],
  "links": [
    {"source": 1, "destination": 2}
  ]
}
```

SDN Mashup dynamically generated

```
{
  "elements": [
    {
      "id": 1,
      "element": "Monitor",
      "service": "monitorService",
      "config" : {
        "position": [298,109],
        "input"   : "[wired]"
      }
    }, {
      "id": 2,
      "element": "Openflow Controller",
      "service": "poxService",
      "config" : {
        "position"   : [35,113],
        "ip"         : "143.54.12.210",
        "port"       : "8080",
        "type"       : "pox",
        "eventParams": {
          "openflowSwitch": {
            "dpid"        : "00:00:00:00:00:00:04:0e",
            "openflowPort": {
              "number"                   : "1",
              "percentTransmittedDropped": "5"
            }
          }
        }
      }
    }
  ],
  "links": [
    {"source": 1, "destination": 2}
  ]
}
```

The time-composition evaluation, first, varied the number of elements forming the static SDN Mashup from 2 to 8. It is noteworthy that more than 60% of mashups consist of 3 to 8 resource components (*i.e.*, elements) [94]. Second, this evaluation modified the amount of static SDN Mashups from 10 to 50; note that the number of static SDN Mashups defines the quantity of simultaneously generated dynamic SDN Mashups.

Figure 4.10 presents the time-composition results. These results reveal that the M&E System generates dynamic SDN Mashups by customizing static ones in a short time. The worst behavior took approximately $14500ms$ to dynamically compose 50 SDN Mashups formed by 8 resources. Furthermore, the time-composition increases linearly with the growth of dynamically generated SDN Mashups and of the elements that compose such SDN Mashups. Considering the above results, this evaluation corroborates that *(i)* the module for generating dynamic SDN Mashups has a similar time-composition

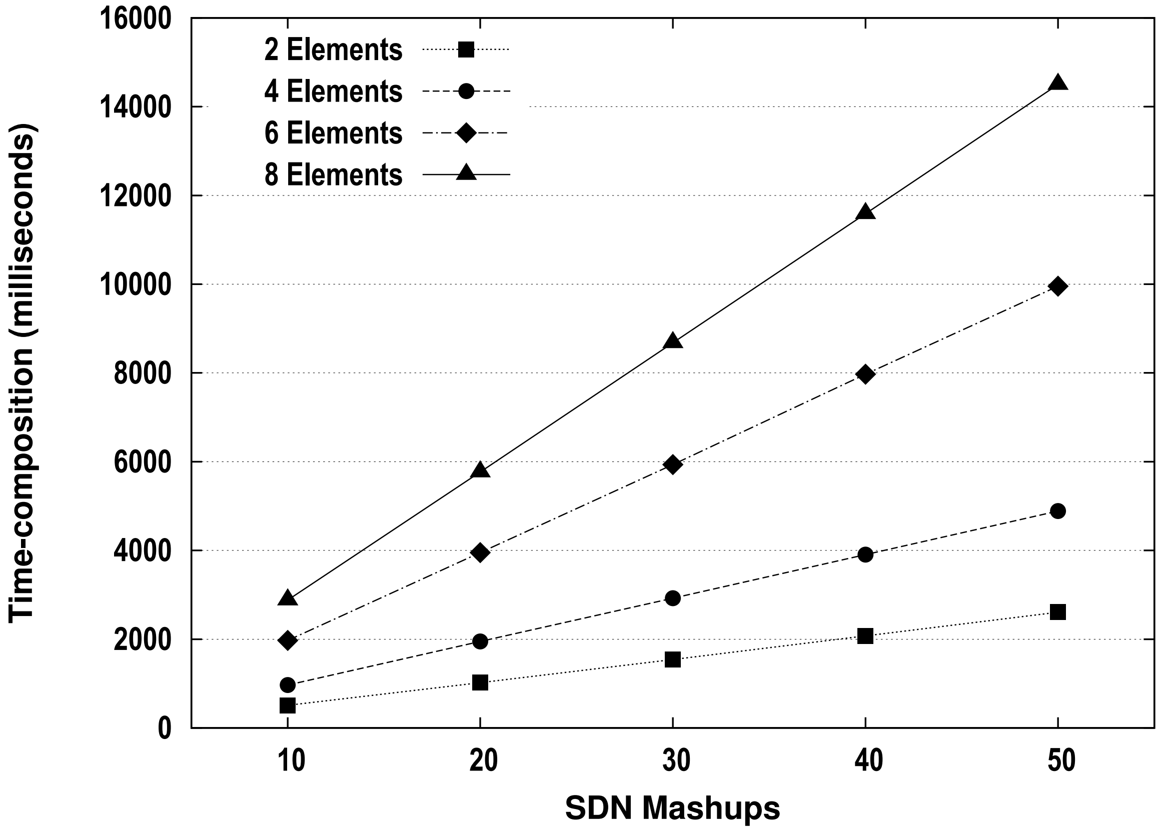


Figure 4.10. Time-composition behavior

behavior than the composition proposals introduced on other application domains [95] and *(ii)* it is feasible, in terms of time-composition, to implement the M&E Framework for coping effectively with events occurring in the SDN environment.

4.2.6 Time-consuming: results and analysis

The time-consuming (*i.e.*, t_{cons}) is the time in seconds (s) that the Network Administrator spends to carry management tasks on SDN managed resources. This evaluation raised a management operation called *MonitoringAS*. Such operation considered the OpenFlow-based networks handled by Beacon, POX, and Floodlight as three inter-connected Autonomous Systems (AS), called AS_1, AS_2, and AS_3, respectively. If an unexpected degradation takes place in the performance of the links that connect these ASs, the Network Administrator needs to identify in each AS which are the switches that are causing such problem. In this sense, the Network Administrator requires to rapidly and easily obtain a monitoring tool that presents, in an integrated, visual, and intelligible way, information about these switches and their connecting links.

To conduct the *MonitoringAS* operation, the Network Administrator tests several options: the *Monitoring Script*, the *Static Monitoring Mashup*, and the *Dynamic Monitoring Mashup*. In general terms, the *Monitoring Script* is an application programmed and executed by the Network Administrator in a low-level abstraction. The *Static Monitoring Mashup* is an SDN Mashup statically built and launched by the Network Administrator using the M&E System. The *Dynamic Monitoring Mashup* is an SDN Mashup automatically generated by the M&E System that offers the same functionalities as the *Static Monitoring Mashup*.

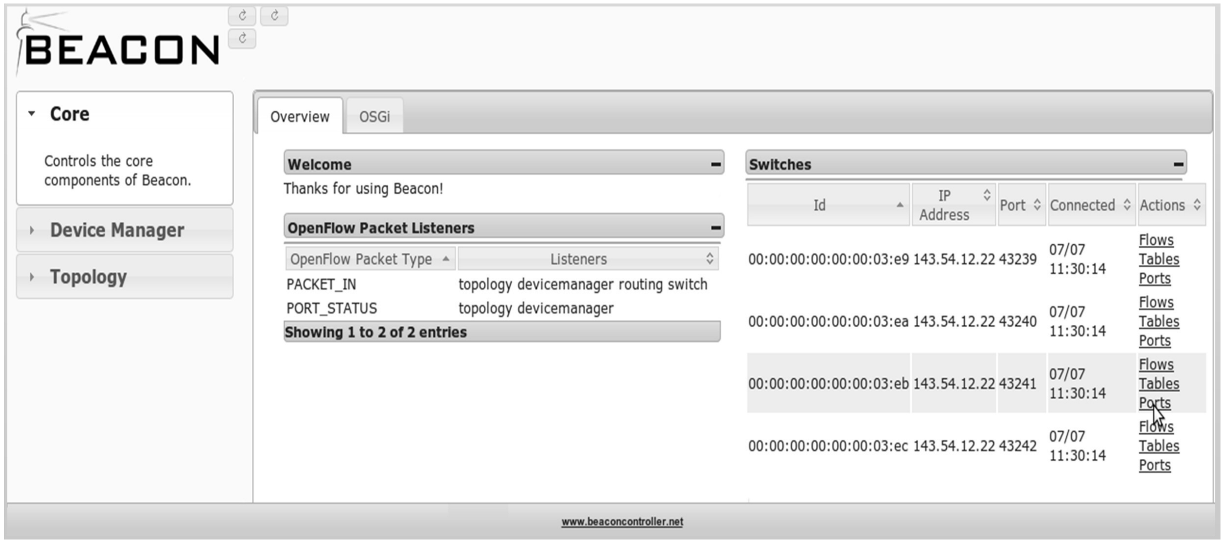
To assess t_{cons}, this evaluation uses KLM (see Table 4.1). A result publication of this thesis [96] corroborates the feasibility of using KLM for estimating time-consuming by conducting an experiment that measured the time spent by end-users for carrying out management tasks—like the aforedescribed *MonitoringAS*—on a similar *Static Monitoring Mashup* approach.

Time-consuming of *Monitoring Script*

$t_{cons:script}$ is the time spent by the Network Administrator for carrying out the *MonitoringAS* operation with the *Monitoring Script*. Without the M&E System, the Network Administrator retrieves monitoring information by using one Web-based management tool per controller, such as the *Beacon Web UI*, the *POX Web UI*, and the *Floodlight Web UI*. As these tools operate similarly, $t_{cons:beacon} = t_{cons:pox} = t_{cons:floodlight}$.

The Network Administrator retrieves information in HTML tables about the packet traffic of a switch from the *Beacon Web UI* (see Figure 4.11) by conducting the following actions: *(i)* point the mouse to Core Components tab, *(ii)* press and release the mouse to select the Core Components tab, *(iii)* point the mouse to the Overview tab that presents a list of switches, *(iv)* press and release the mouse to select the Overview tab, *(v)* point the mouse to select a switch, *(vi)* press and release the mouse to select a switch, *(vii)* point the mouse to the Ports link; and *(viii)* press and release the mouse to select Ports of switch. Considering these actions, $t_{cons:beacon} = H+(4*P)+(8*B) = 5.6s$. Therefore, the time for separately using the above-mentioned tools is $t_{cons:nonIntegrated} = 3*t_{cons:beacon} = 16.8s$.

To obtain in a unique GUI the retrieved information by the Web-based tools, the Network Administrator develops (*i.e.*, $t_{dev:script}$) and launches (*i.e.*, $t_{lau:script}$) the *Monitoring Script*. As this script generates HTML tables and RRDtool images $t_{dev:script} =$



Controls the core components of Beacon.

Thanks for using Beacon!

OpenFlow Packet Type	Listeners
PACKET_IN	topology devicemanager routing switch
PORT_STATUS	topology devicemanager

Showing 1 to 2 of 2 entries

Id	IP Address	Port	Connected	Actions
00:00:00:00:00:00:03:e9	143.54.12.22	43239	07/07 11:30:14	Flows Tables Ports
00:00:00:00:00:00:03:ea	143.54.12.22	43240	07/07 11:30:14	Flows Tables Ports
00:00:00:00:00:00:03:eb	143.54.12.22	43241	07/07 11:30:14	Flows Tables Ports
00:00:00:00:00:00:03:ec	143.54.12.22	43242	07/07 11:30:14	Flows Tables Ports

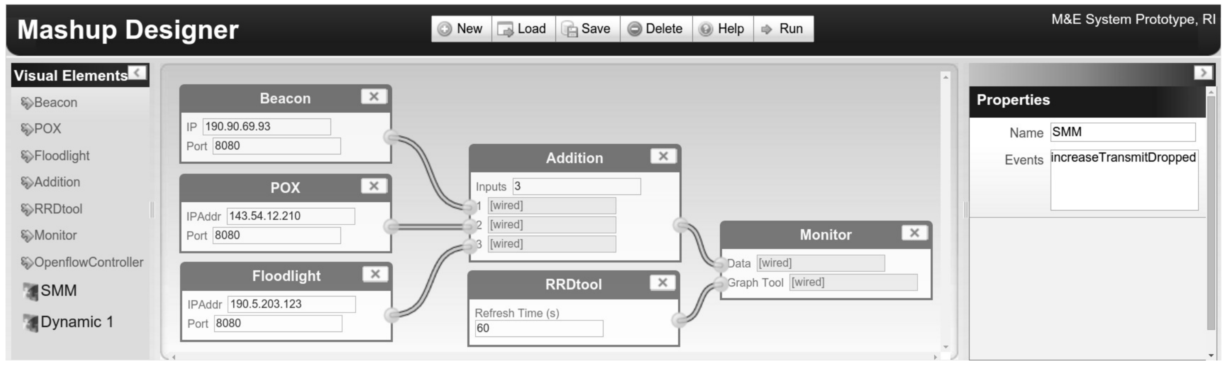
Figure 4.11. Beacon Web UI

$t_{dev:table} + t_{dev:rrdImages}$. Considering solely the time to type the code for generating the tables and images, $t_{dev:script} = [H + (290 * K)] + [H + (1200 * K)] = 298.8s$. In turn, $t_{lau:script} = H + P + (2 * B) + (11 * K) = 3.9s$. Thus, the time-consuming for conducting the *MonitoringAS* operation by using the *Monitoring Script* is $t_{cons:script} = t_{cons:nonIntegrated} + t_{dev:script} + t_{lau:script} = 319.5s$.

Time-consuming of *Static Monitoring Mashup*

$t_{cons:static}$ is the time spent by the Network Administrator for carrying out the *MonitoringAS* operation with the *Static Monitoring Mashup*. Here, $t_{cons:static} = t_{dev:static} + t_{lau:static} + t_{use:static}$. Since the Network Administrator uses the Mashup Designer to build the *Static Monitoring Mashup*, then $t_{dev:static} = t_{sel} + t_{con} + t_{com:}$ [96].

To develop the *Static Monitoring Mashup* (see Figure 4.12), the Network Administrator initially selects the mashable entities by dragging-and-dropping the corresponding Visual Elements: Beacon, POX, Floodlight, Addition, RRDtool, and Monitor. Thus, $t_{sel} = 6 * Dnd = 7.8s$. Afterwards, the Network Administrator configures the selected resources by providing the operation parameters of Beacon, POX, Floodlight, Addition, and RRDtool. As this end-user manually writes these parameters, $t_{con:beacon} = t_{con:pox} = t_{con:floodlight} = \{2 * [P + H + (2 * B)]\} + [(15 + 5) * K] = 7.4s$ and $t_{con:addition} = t_{con:rddtool} = P + H + (2 * B) + (3 * K) = 2.3s$. Therefore, $t_{con} = 26.8s$. Finally, the Network Administrator combines the selected and configured Visual Elements by wiring Beacon



Beacon: IP 190.90.69.93, Port 8080
POX: IPAddr 143.54.12.210, Port 8080
Floodlight: IPAddr 190.5.203.123, Port 8080
Addition: Inputs 3, 1 [wired], 2 [wired], 3 [wired]
RRDtool: Refresh Time (s) 60
Monitor: Data [wired], Graph Tool [wired]
Properties: Name SMM, Events increaseTransmitDropped

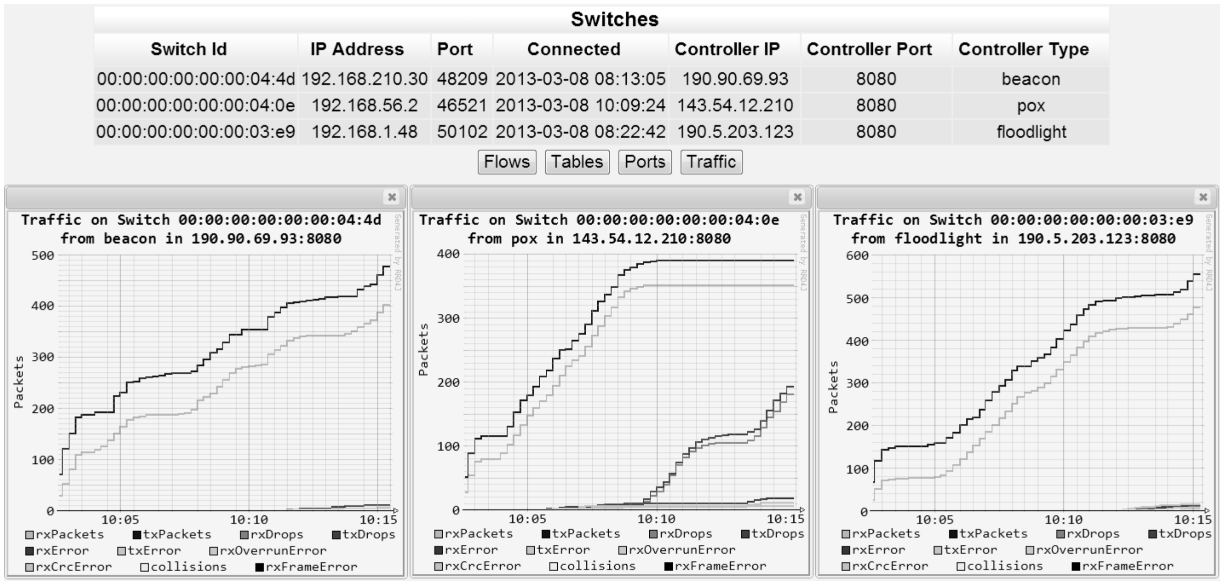
Figure 4.12. Static and dynamic monitoring SDN Mashup

with Addition, POX with Addition, Floodlight with Addition, Addition with Monitor, and RRDtool with Monitor. Thus, $t_{com} = 5 * Wire = 20.5s$.

Before requesting the execution of the *Static Monitoring Mashup*, the Network Administrator saves it: *(i)* point the mouse in the Save button and click it, *(ii)* point the mouse in the dialog that asks for the name of the SDN Mashups and click it; and *(iii)* type the string “SMM”—stands for *Static Monitoring Mashup*. Once the *Static Monitoring Mashup* has been saved, the Network Administrator launches it by clicking the button Run. Thus, $t_{lau:static} = 3 * [H + P + (2 * B)] + (3 * K) = 5.7s$.

On runtime, the *Static Monitoring Mashup* (see Figure 4.13) allows the Network Administrator to retrieve monitoring information about switches and links from three different controllers by the following actions: *(i)* points the mouse to the Controllers list, *(ii)* presses and releases the mouse to select three distinct controllers, *(iii)* points the mouse to the button Switches/Links; and *(iv)* presses and releases the mouse to click the button Switches/Links. Furthermore, to retrieve information about flows, tables, ports, or traffic of three switches: *(i)* presses and releases the mouse to select three switches in different controllers, *(ii)* points the mouse to the button Flows, Tables, Ports, or Traffic; and *(iii)* presses and releases the mouse to click the button Flows, Tables, Ports, or Traffic. Thus, $t_{use:static} = H + (3 * P) + (16 * B) = 5.3s$.

Finally, the computed values of $t_{dev:static}$, $t_{lau:static}$, and $t_{use:static}$ provide an estimated value of $t_{cons:static}$. As a result, it is expected that the Network Admistrator takes $66.1s$ to carry out *MonitoringAS* with the *Static Monitoring Mashup*.



Switches						
Switch Id	IP Address	Port	Connected	Controller IP	Controller Port	Controller Type
00:00:00:00:00:00:04:4d	192.168.210.30	48209	2013-03-08 08:13:05	190.90.69.93	8080	beacon
00:00:00:00:00:00:04:0e	192.168.56.2	46521	2013-03-08 10:09:24	143.54.12.210	8080	pox
00:00:00:00:00:00:03:e9	192.168.1.48	50102	2013-03-08 08:22:42	190.5.203.123	8080	floodlight

Flows
Tables
Ports
Traffic
Traffic on Switch 00:00:00:00:00:00:04:4d from beacon in 190.90.69.93:8080
Traffic on Switch 00:00:00:00:00:00:04:0e from pox in 143.54.12.210:8080
Traffic on Switch 00:00:00:00:00:00:03:e9 from floodlight in 190.5.203.123:8080
Packets
10:05
10:10
10:15
rxPackets
txPackets
rxDrops
txDrops
rxError
txError
rxOverrunError
rxCrcError
collisions
rxFrameError

Figure 4.13. Static and dynamic monitoring SDN Mashup on runtime

Time-consuming of *Dynamic Monitoring Mashup*

$t_{cons:dynamic}$ is the time spent by the Network Administrator for carrying out the *MonitoringAS* operation with the *Dynamic Monitoring Mashup*. As the *Dynamic Monitoring Mashup* is generated by the M&E System (see Figure 4.12), $t_{cons:dynamic} = t_{lau:dynamic} + t_{use:dynamic}$. Since the *Dynamic Monitoring Mashup* is an SDN Mashup dynamically composed, the Network Administrator launches it by clicking the button Run. Therefore, $t_{lau:dynamic} = H + P + (2 * B) = 1.7s$. Furthermore, since on runtime both the *Dynamic Monitoring Mashup* and the *Static Monitoring Mashup* provide identical functionalities and show the same GUI (see Figure 4.13), $t_{use:dynamic} = 5.3s$. Therefore, $t_{cons:dynamic} = 7s$.

Figure 4.14 depicts the time-consuming results that reveal: *(i)* the time-consuming for developing the *Static Monitoring Mashup* (*i.e.*, $t_{dev:static} = 55.1s$) is less than for the *Monitoring Script* (*i.e.*, $t_{dev:script} = 298.8s$), achieved through the mechanisms provided by the Mashup Designer, *(ii)* the time that the Network Administrator takes for developing the *Dynamic Monitoring Mashup* is zero, attained by the automatic recognition of events and the dynamic composition of SDN Mashups, *(iii)* since the Mashup Designer requires saving SDN Mashups before executing them, the time-consuming for launching the *Static Monitoring Mashup* (*i.e.*, $t_{lau:static} = 5.7s$) is greater than for the *Monitoring Script* (*i.e.*, $t_{lau:script} = 3.9s$); and *(iv)* because every dynamic SDN Mashup is ready to

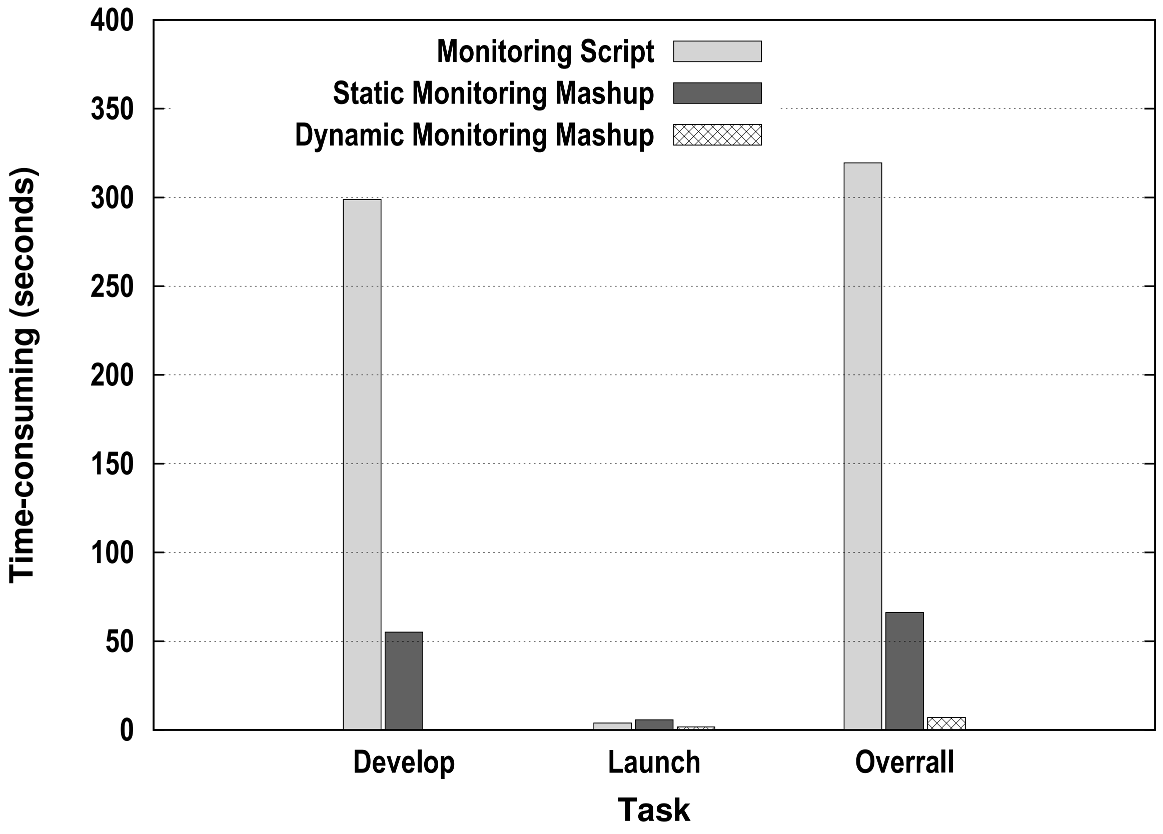


Figure 4.14. Time-consuming on conducting the *MonitoringAS* operation

be launched by the Mashup Designer, the time for launching the *Dynamic Monitoring Mashup* (*i.e.*, $t_{lau:dynamic} = 1.7s$) is less than for the *Monitoring Script* and the *Static Monitoring Mashup*.

Summing up, the time-consuming results for conducting the *MonitoringAS* operation with the *Static Monitoring Mashup* (*i.e.*, $t_{cons:static} = 66.1s$) and the *Dynamic Monitoring Mashup* (*i.e.*, $t_{cons:dynamic} = 7s$) are less—about 79.3% and 97.8%—than without these approaches (*i.e.*, $t_{cons:script} = 319.5s$). In fact, the *Dynamic Monitoring Mashup* reduces approximately 89.4% of the time-consuming estimated for the *Static Monitoring Mashup*. The overall result and the results per task demonstrated that, in terms of time-consuming, it is feasible to deploy the M&E Framework for carrying out management tasks in the SDN environment.

4.2.7 Response time: results and analysis

To continue the evaluation, it was measured the response time in milliseconds (ms) of the *Monitoring Mashup*—on runtime both the *Static Monitoring Mashup* and the *Dynamic Monitoring Mashup* work equally—and the *Beacon Web UI* when conducting the operations *MonitoringSwitches* and *MonitoringLinks* on the SDN resources deployed by the test environment (see Figure 4.8). These operations offer visual information of the switches and their connection links, which is useful to carry out the *MonitoringAS* operation.

The response time evaluation of *MonitoringSwitches* varied the number of switches from 20 to 100 per OpenFlow-based network. Thus, the total number of switches in each evaluation was 60, 120, 180, 240, and 300. Figure 4.15 presents the corresponding results. Considering the ranking of the response time (r in s) established by the performance analysis for Java Websites [93], the response time results reveal: *(i) MonitoringSwitches* of the *Monitoring Mashup* has a good r that grows negligibly (less than $1ms\ per\ switch$) when the number of switches is increased; and *(ii)* r is ranked as optimal for the *Beacon Web UI* and as good for the *Monitoring Mashup*; this result was expected because *Beacon Web UI* operates with one type of OpenFlow controller (*i.e.*, Beacon) and the *Monitoring Mashup* with three different types of OpenFlow controller (*i.e.*, Beacon, POX, and Floodlight).

The response time evaluation of *MonitoringLinks* modified the number of links from 50 to 250 per OpenFlow-based network. Therefore, the total number of links in each evaluation was 150, 300, 450, 600, and 750. Figure 4.16 depicts the corresponding results that reveal: *(i) MonitoringLinks* of the *Monitoring Mashup* has a good r that grows negligibly (less than $1ms\ per\ switch$) when the number of links is increased; and *(ii)* r is ranked as optimal for *Beacon Web UI* and as good for the *Monitoring Mashup*; again, this result was expected because *Beacon Web UI* works with one type of OpenFlow controller and the *Monitoring Mashup* with three different types.

Although, at runtime, the *Monitoring Mashup* uses several software modules (*e.g.*, the Beacon Service and the OFMonitor Service) to integrate and present monitoring information from different controllers, its behavior on response time ranks as good for the most operations, regardless of deployed technologies and the number of monitored switches and links. Such behavior is because the Adapters hides the heterogeneity of OpenFlow controllers and, in turn, their centralized nature handles the number of

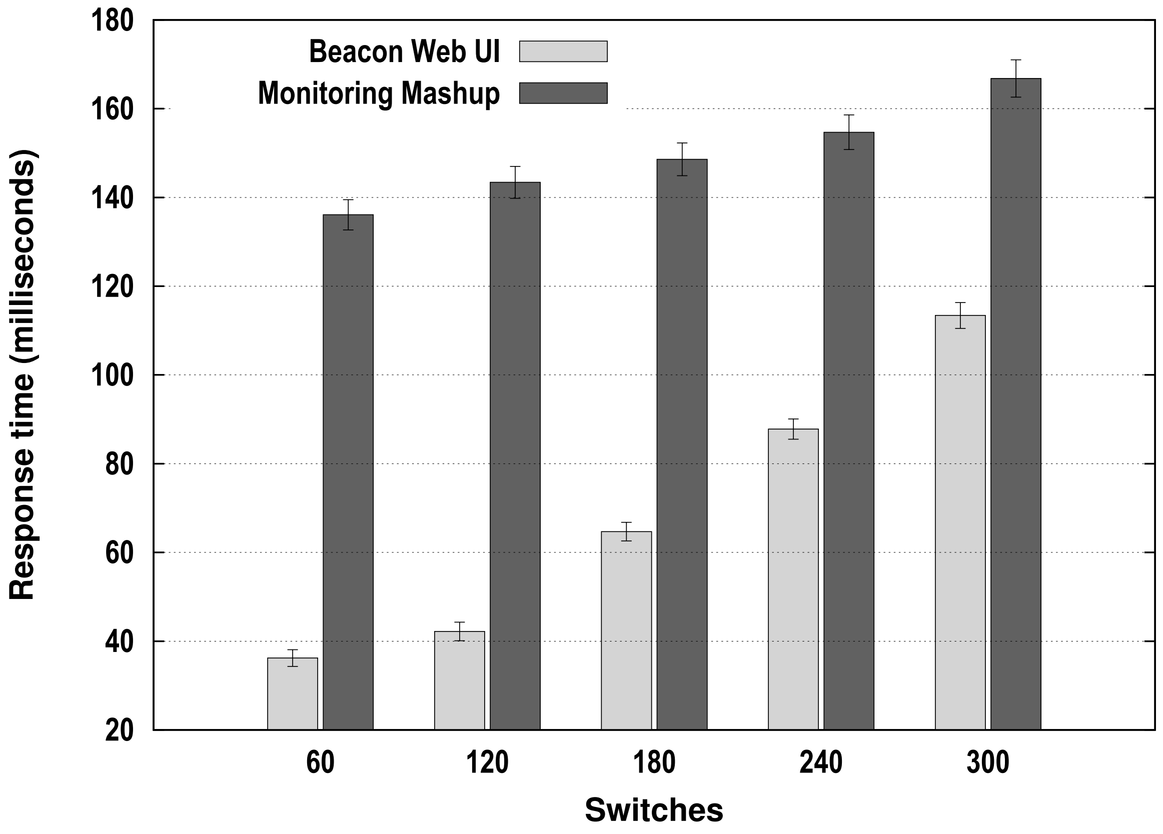


Figure 4.15. Response time on conducting the *MonitoringSwitches* operation

network elements. Summing up, the response time results corroborated that, in terms of such metric, it is feasible to deploy the M&E Framework for conducting management tasks on SDN resources.

4.2.8 Network traffic: results and analysis

Continuing the evaluation, it was proceeded to measure the network traffic in kilobytes (kB) generated by the *Monitoring Mashup* and the *Beacon Web UI* when carrying out *MonitoringSwitches* and *MonitoringLinks* on the SDN-based networks deployed by the test environment (see Figure 4.8). This evaluation uses bytes (B) and kilobytes (kB) to express the network traffic. It is to highlight that the traffic generated by the *Monitoring Mashup* and the *Beacon Web UI* for the aforementioned operations is independent of the deployed topology (e.g., linear and tree).

The network traffic evaluation of *MonitoringSwitches* varied the number of switches from 20 to 100 per OpenFlow-based network. Thus, the total number of switches in

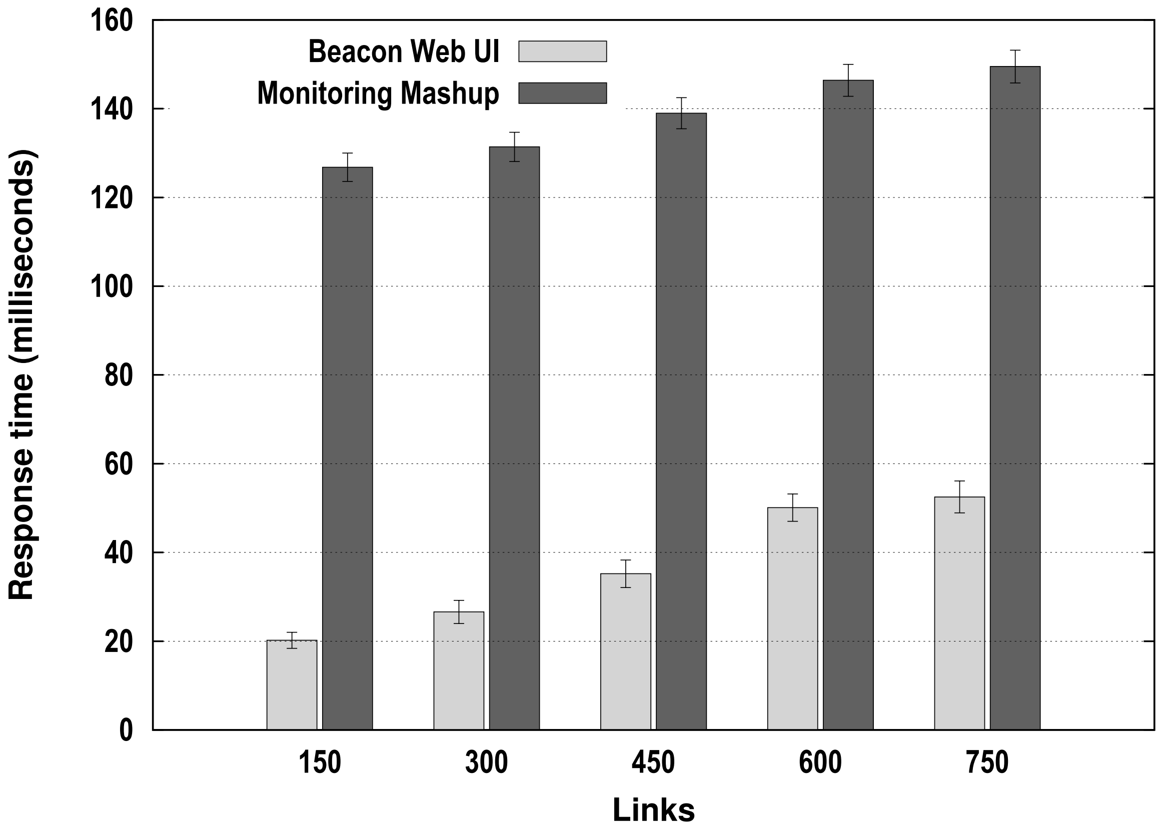


Figure 4.16. Response time on conducting the *MonitoringLinks* operation

each evaluation was 60, 120, 180, 240, and 300. Figure 4.17 presents the corresponding results, revealing: *(i)* the traffic generated by *MonitoringSwitches* of the *Monitoring Mashup* grows negligibly (approximately $112B\ per\ switch$) when the number of switches is increased, *(ii)* the *Monitoring Mashup* generates more traffic than the *Beacon Web UI*; and *(iii)* the additional traffic generated by the *Monitoring Mashup* is always less than 10%. Considering that the *Beacon Web UI* operates with just one type of controller and the *Monitoring Mashup* integrates data from three different types, the above facts corroborated that *MonitoringSwitches* of he *Monitoring Mashup* has a good behavior on network traffic.

The network traffic evaluation of *MonitoringLinks* modified the number of links from 50 to 250 per OpenFlow-based network. Therefore, the total number of links in each evaluation was 150, 300, 450, 600, and 750. Figure 4.18 depicts the corresponding results that reveal: *(i)* the traffic generated by *MonitoringLinks* of the *Monitoring Mashup* grows negligibly (about 129 Bytes per link) when the number of links is increased, *(ii)*

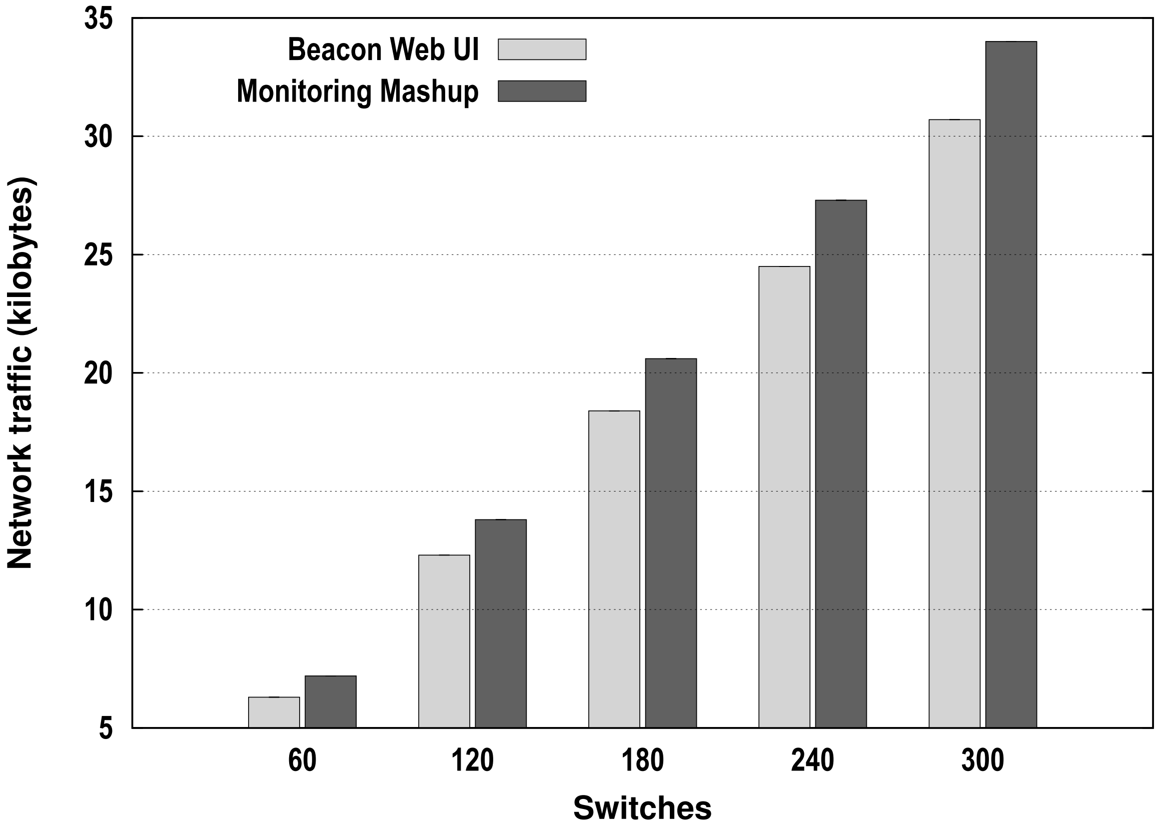


Figure 4.17. Network traffic on conducting the *MonitoringSwitches* operation

the *Monitoring Mashup* generates more traffic than the *Beacon Web UI*; and *(iii)* the additional traffic generated by the *Monitoring Mashup* is always less than 5%. Since the *Beacon Web UI* operates with just one type of controller and the *Monitoring Mashup* integrates data from three different types, the above facts corroborated that *MonitoringLinks* of the *Monitoring Mashup* has a good behavior on network traffic.

Regarding the results of the network traffic evaluation of the *Monitoring Mashup*, it is important to mention: *(i)* JSON was used to decrease the size of information exchanged between the modules of the M&E System because JSON is less verbose than XML; and *(ii)* the size of GUIs is too small to impact the quantity of traffic generated by the *Monitoring Mashup*. Furthermore, although the *Monitoring Mashup* integrates monitoring information from different controllers by using several additional software modules, its extra traffic is always less than 10% (the worst operation was *MonitoringSwitches*). Summing up, the above results demonstrated that, in terms of network traffic, it is feasible to implement the M&E Framework for performing management tasks

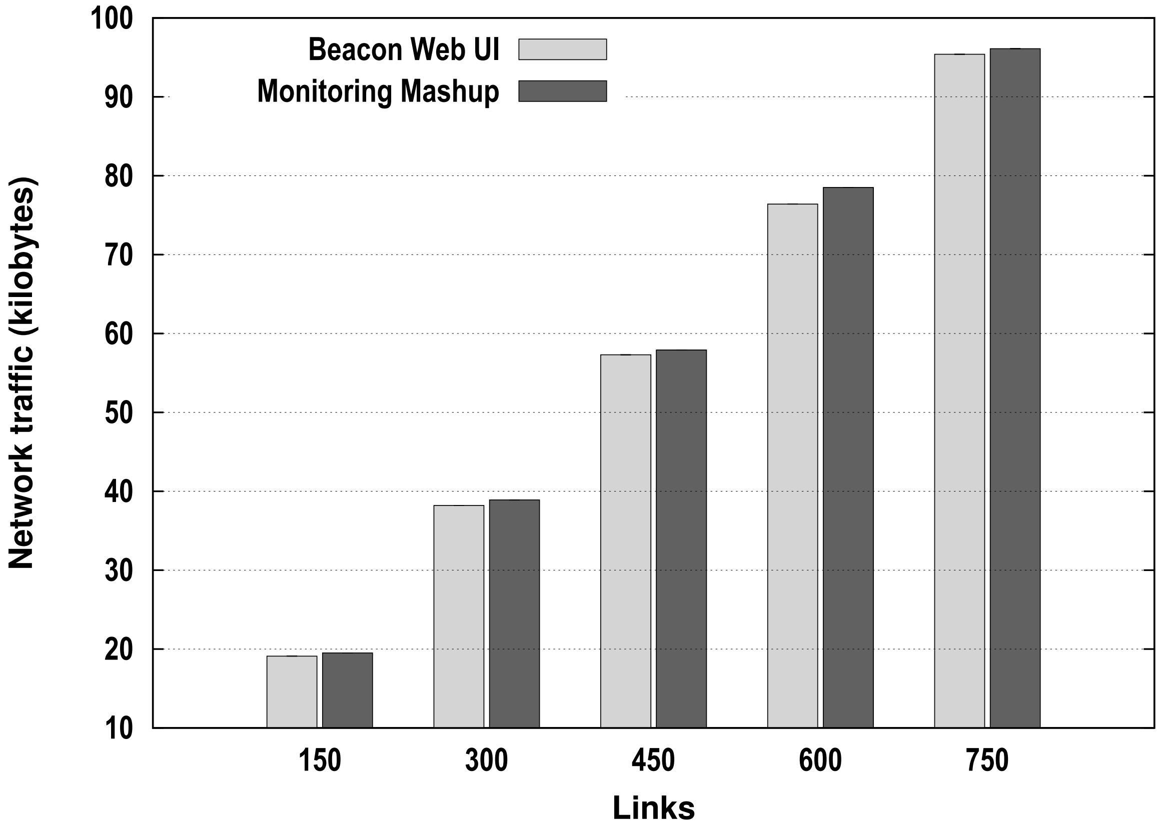


Figure 4.18. Network traffic on conducting the *MonitoringLinks* operation

in the SDN environment.

4.3 Final remarks

This chapter presented the evaluation and analysis of the CIM-based Information Model and the M&E Framework by conducting a case study for each one. Through a quantitative perspective, this evaluation and analysis was carried out in terms of the following metrics: *(i)* time-consuming, related to the process required by Network Administrators to perform managements operations in the virtual, dynamic, and heterogeneous environment of SDN, *(ii)* response time and network traffic, associated with the behavior on runtime of solutions used to manage SDN-based networks; and *(iii)* time-recognition and time-composition, related to the M&E Framework modules for automatically recognizing events and dynamically composing SDN Mashups, respectively.

The evaluation results from the case study for the CIM-based information model revealed several facts. First, regarding the performance analysis for Java Websites, the proposed approach has an admissible behavior in terms of response time, similar than using isolated tools. Second, it introduces a small response time overhead (*i.e.*, < 0.8s per switch) compared with the minimal time required by Network Administrators to handle dispersed solutions (*i.e.*, > 1.35s per switch). Third, it has a good behavior on network traffic when managing several devices (*i.e.*, < 17% for 20 switches or more) in relation to employing distinct tools.

Regarding the case study for the M&E Framework, the evaluation results demonstrated that, first, if the Network Administrator carries out management operations (*e.g.*, *MonitoringAS*) by developing and launching static SDN Mashups, the time-consuming decreases. Second, such decreasing is greater when mechanisms for automatic recognition of events and dynamic composition of SDN Mashups are used. Third, although SDN Mashups use extra software modules to cope with events, such layers generate few additional response time and network traffic in relation to Web-based network management tools (*e.g.*, Beacon Web UI) and proprietary scripts. This last fact demonstrates that the proposed approach has a good behavior in terms of response time and network traffic.

From a qualitative point of view, the introduced Management Plane reference architecture provides mainly simplicity and formalization. The simplicity refers to that this Management Plane facilitates integrating the SDN management operations of network administrators. They do not require to employ multiple solutions to completely manage SDNs deployed with various technologies because this approach addresses the management requirements of the whole SDN environment and hides the heterogeneity of the deployed resources. Regarding the formalization, the CIM-based Information Model introduced in this paper can be considered as a step forward in unifying a conceptual understanding of the virtual, dynamic, and heterogeneous environment of SDN. It is feasible because the Information Model relies on CIM to provide a technology-agnostic and consistent characterization of SDN. Moreover, future SDN proposals may extend this approach for tackling arising challenges in SDN management.

In turn, the M&E Framework provides mainly flexibility and extensibility. The flexibility refers to that the M&E Framework allows the Network Administrators and the M&E Developer to customize and improve their workspace. These end-users do not require a lot of Web programming skills to compose management tools (*e.g.*, the *Static Monitoring Mashup and* the *Dynamic Monitoring Mashup*) because the M&E Framework provides a high-level abstraction (*i.e.*, mashable entities) of network management technologies as well as of management operations.

Chapter 5

Conclusions

This chapter starts summarizing the research work carried out in this thesis. Then, it provides the answers for the fundamental questions that guided the verification of the hypothesis defended in this thesis. Afterward, the chapter overviews the main contributions achieved when conducting such verification. The last section outlines directions for future work.

This thesis presented the investigation carried out to verify the hypothesis: "**a Management Plane that establishes a common Information Model and employs Web technologies (2.0/3.0) would provide an effective approach for managing virtual, dynamic, and heterogeneous networks based on the SDN paradigm**". Based on the hypothesis, this work proposed a Management Plane aimed to facilitate the integrated management of the whole SDN architecture in virtual, dynamic, and heterogeneous environments. Such Management Plane delineated a reference architecture using the four submodels that compose the OSI network management model: Information, Organizational, Communication, and Functional. Focusing on the Information Model, this thesis relied on CIM to define a consistent characterization of the entire SDN management environment regardless of the deploying technologies. This Information Model extended the CIM Schema to accomplish a generic abstraction of the SDN managed and managing resources and their relationships. Furthermore, the Management Plane introduced a M&E Framework in the Manager component to support the high-level and dynamic creation of composite applications (*i.e.*, SDN Mashups) that carry out management tasks in the SDN environment.

This thesis also displayed the reference implementation of the proposed approach as well as an extensive evaluation and analysis about effectively carrying out management tasks with and without this proposal. In particular, different case studies raised specific management requirements in realistic scenarios based on SDN to evaluate and analyze the effectivity in terms of time and network traffic. The evaluation results demonstrated that the approach is effective for managing the virtual, dynamic, and heterogeneous environment of SDN because: *(i)* the integrated solution built upon the CIM-based Information Model reduced the time for managing an SDN deployed with virtual and heterogeneous technologies, *(ii)* composing—statically or dynamically—and launching SDN Mashups decreased the time for carrying out management tasks in a virtual, dynamic, and heterogeneous SDN environment; and *(iii)* both aforementioned solutions demonstrated good behavior on the response time as well as on the network traffic.

5.1 Answers for the fundamental questions

At first, this thesis defined two fundamental questions that guided the investigation about the feasibility of deploying a Management Plane that establishes a common Information Model and employs Web technologies (2.0/3.0) for effectively managing virtual, dynamic, and heterogeneous networks based on the SDN paradigm. This section reviews and answers such questions.

Fundamental question I: *What is the performance, in terms of time and network traffic, of solutions based on a technology-agnostic and consistent Information Model for managing virtual, dynamic, and heterogeneous networks based on the SDN paradigm?*

The lack of frameworks for integrated SDN management forces network administrators to handle separated tools (*i.e.*, isolated solution) for managing resources from distinct planes as well as various technology instances. This isolated solution remains complex and time-consuming because of the diversity of vendors and technologies for deploying SDN resources. The Management Plane along with the CIM-based Information Model enabled to cope with such time—and to some extent with the complexity—corroborating the importance of establishing a common Information Model as the key towards building an integrated management architecture for the SDN environment. From a qualitative point of view, the proposed approach reduces the complexity of management by including modules (*e.g.*, Data Repository and Adapters) that hide heterogeneity of SDN environments. From a quantitative perspective, this thesis employed KLM [91] to illustrate that the proposal consumes less time than the isolated solution for carrying out management tasks in SDNs deployed with virtual and heterogeneous technologies, such as the raised case study. Particularly, the integrated solution built upon the proposed approach reduced the consuming of time at a rate about $1.35s\ per\ switch$ in relation to the isolated solution.

In addition, considering that the proposed approach incorporated extra modules to address the heterogeneity of SDN environments, the evaluation results for response time revealed a good behavior. Specifically, according to the performance analysis for Java Websites [93], the response time results for the integrated solution ranked as admissible, similar than for the isolated solution. In fact, the integrated solution introduced a small response time overhead (less than $0.8s\ per\ switch$) compared with the minimal time required by network administrators to handle the isolated solution (more than $1.35s\ per\ switch$). Likewise, the additional components of the proposed approach generated low traffic overhead, demonstrating a moderate behavior in terms of network traffic. Explicitly, the traffic overhead for managing more than 20 network devices was always less than 17%.

Fundamental question II: *What is the performance, in terms of time and network traffic, of solutions that use Web 2.0 and Web 3.0 technologies for managing virtual, dynamic, and heterogeneous networks based on the SDN paradigm?*

For network administrators—end-users with poor programming skills—carrying out management tasks in SDN environments consumes a lot of time. The M&E framework enabled network administrators to overcome such time, confirming the effectivity of employing the technologies from Web 2.0 (*i.e.*, mashups) and Web 3.0 (*i.e.*, CEP, Rule-based Systems, and JSON) to deploy the Manager component of the introduced Management Plane. This thesis used per-task metrics [97, 91] to demonstrate that the proposed solution is less consuming of time than conventional solutions (*i.e.*, proprietary and incompatible CLIs, GUIs, and scripts) used for coping with management requirements like the raised in the case study. In particular, the static SDN Mashup solution was approximately 79% less consuming of time than the conventional solution, whereas the dynamic SDN Mashup solution reduced more than 89% of the consuming of time. The latter was achieved because the M&E framework provided mechanisms for automatically recognizing events and dynamically composing SDN Mashups.

Furthermore, although the M&E framework included additional software modules for creating and executing SDN Mashups, it evidenced a good behavior in terms of response time. Indeed, regarding the performance analysis for Java Websites [93], the response time results of the proposed approach ranked as good. Similarly, the M&E framework demonstrated a good behavior in terms of network traffic because its additional modules generated few extra traffic in relation to the conventional solutions aforementioned. Specifically, the traffic overhead was always less than 10%.

5.2 Contributions

This thesis investigated the feasibility of deploying a Management Plane that establishes a common Information Model and employs Web technologies (2.0/3.0) for effectively managing virtual, dynamic, and heterogeneous networks based on the SDN paradigm. The carrying out of such investigation led to the following major contributions.

- **The reference architecture for the Management Plane.** This reference architecture relied on the OSI network management submodels (*i.e.*, Information, Organizational, Communication, and Functional) to define a Management Plane aimed to facilitate integrated management of the virtual, dynamic and heterogeneous SDN environment.

- **The CIM-based Information Model.** This model provided a technology-agnostic and consistent characterization of the entire SDN environment from a management perspective.

- **The SDN Mashup concept.** This concept presented how to use mashups to build and customize, in a high-level abstraction, management tools in the SDN environment.

- **The M&E Framework.** This mashup-based and event-driven framework enabled the creation and execution of static and dynamic SDN Mashups in the Manager component of the proposed Management Plane.

5.3 Future work

During the carrying out of this thesis, interesting opportunities for further research were observed. These opportunities are outlined below.

- SDN management architecture. The Management Plane approach provided a reference architecture focused on the Information Model. Therefore, there is an opportunity to extend this Management Plane by defining and assessing the remaining submodels (*e.g.*, Communication and Functional) in order to afford a complete, well-defined SDN management architecture. Furthermore, since SDN mutually complements with other networking technologies, such as the Network Function Virtualization and the Cloud Networking, there is a chance to improve the proposed Management Plane for encompassing the management requirements and challenges of such technologies.

- Big Data approach. The M&E framework deployed a centralized approach not suitable to handle the huge amount of data generated by networks based on the SDN paradigm. Thus, there is an opportunity to investigate the feasibility of using Big Data technologies to deploy the Manager component in order to capture, process, and analyze enormous sets of information for obtaining significant results that allow to forecast events and improve the decision-making.

References

[1] J. Pan, S. Paul, and R. Jain, "A survey of the research on future internet architectures," *Communications Magazine, IEEE*, vol. 49, no. 7, pp. 26–36, 2011. 1

[2] N. M. M. K. Chowdhury and R. Boutaba, "Network virtualization: state of the art and research challenges," *Communications Magazine, IEEE*, vol. 47, no. 7, pp. 20–26, 2009. 1

[3] Y. Luo, C. Zhang, T. Ficarra, and E. Murray, "The design, performance evaluation and use cases of a virtualised programmable edge node for network innovations," *Int. J. Commun. Netw. Distrib. Syst.*, vol. 6, pp. 249–267, April 2011. 1

[4] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn," *Queue*, vol. 11, pp. 20:20–20:40, Dec. 2013. 1, 9, 11

[5] H. Kim and N. Feamster, "Improving network management with software defined networking," *Communications Magazine, IEEE*, vol. 51, pp. 114–119, February 2013. 1

[6] 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, 2012. 1

[7] F. Hao, T. V. Lakshman, S. Mukherjee, and H. Song, "Enhancing dynamic cloud-based services using network virtualization," *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 67–74, January 2010. 1

[8] M. B. Anwer and N. Feamster, "Building a fast, virtualized data plane with programmable hardware," *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 75–82, January 2010. 1

[9] H. Shimonishi and S. Ishii, “Virtualized network infrastructure using openflow,” in *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*, pp. 74–79, 2010. 1

[10] Opendaylight Project, “The OpenDaylight platform,” 2015. [Online]. Available: http://www.opendaylight.org/project/technical-overview. 2, 11

[11] ONF, “Software-defined networking: The new norm for networks,” white paper, Open Network Foundation, April 2012. 2, 9

[12] S. Kiran and G. Kinghorn, “Cisco open network environment: Bring the network closer to applications,” White Paper C11-728045-03, Cisco, September 2015. 2, 9

[13] B. Rijsman and A. Singla, *Day One: Understanding OpenContrail Architecture*. Juniper Networks Books, November 2013. 2, 9

[14] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015. 2, 9

[15] M. Casado, N. Foster, and A. Guha, “Abstractions for software-defined networks,” *Commun. ACM*, vol. 57, pp. 86–95, October 2014. 2, 9

[16] J. Wickboldt, W. De Jesus, P. Isolani, C. Both, J. Rochol, and L. Granville, “Software-defined networking: management requirements and challenges,” *Communications Magazine, IEEE*, vol. 53, no. 1, pp. 278–285, 2015. 2, 19, 23

[17] ONF, “SDN architecture v1.0,” Technical Reference TR-502, Open Network Foundation, June 2014. 2, 19

[18] R. Wang, D. Butnariu, and J. Rexford, “Openflow-based server load balancing gone wild,” in *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*, Hot-ICE’11, (Berkeley, CA, USA), pp. 12–12, USENIX Association, 2011. 2, 13, 14

[19] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, “Elastictree: saving energy in data center networks,” in *Proceedings of the 7th USENIX conference on Networked systems design and implemen-*

tation, NSDI'10, (Berkeley, CA, USA), pp. 17–17, USENIX Association, 2010. 2, 13

[20] A. K. 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, (New York, NY, USA), pp. 11–18, ACM, 2009. 2, 13

[21] 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*, WINTECH '09, (New York, NY, USA), pp. 59–66, ACM, 2009. 2, 13, 14

[22] ONF, "Openflow management and configuration protocol (OF-CONFIG) v1.2," Technical Specification TS-016, Open Network Foundation, 2014. 2, 13, 14, 23

[23] B. Pfaff and B. Davie, "The open vswitch database management protocol," RFC 7047, Internet Engineering Task Force, December 2013. 2, 13

[24] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, (New York, NY, USA), pp. 19–24, ACM, 2012. 2, 13

[25] 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*, INM/WREN'10, (Berkeley, CA, USA), pp. 3–3, USENIX Association, 2010. 2, 13

[26] 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*, pp. 52–56, Nov 2011. 2, 13

[27] S. Natarajan and X. Huang, "An interactive visualization framework for next generation networks," in *Proceedings of the ACM CoNEXT Student Workshop*, CoNEXT '10 Student Workshop, (New York, NY, USA), pp. 14:1–14:2, ACM, 2010. 2, 13

[28] R. Corin, M. Gerola, R. Riggio, F. De Pellegrini, and E. Salvadori, "Vertigo: Network virtualization and beyond," in *Software Defined Networking (EWSDN), 2012 European Workshop on*, pp. 24–29, 2012. 2, 13

[29] E. Salvadori, R. Doriguzzi Corin, M. Gerola, A. Broglio, and F. De Pellegrini, "Demonstrating generalized virtual topologies in an openflow network," *SIGCOMM Comput. Commun. Rev.*, vol. 41, pp. 458–459, August 2011. 2, 13

[30] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, "Modular SDN programming with pyretic," *USENIX*, vol. 38, pp. 40–47, October 2013. 2, 11, 13

[31] A. Voellmy, H. Kim, and N. Feamster, "Procera: A language for high-level reactive network control," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, (New York, NY, USA), pp. 43–48, ACM, 2012. 2, 11, 13, 14

[32] E. Haleplidis, J. Hadi Salim, S. Denazis, and O. Koufopavlou, "Towards a network abstraction model for sdn," *Journal of Network and Systems Management*, vol. 23, no. 2, pp. 309–327, 2015. 2, 14, 23

[33] B. Pinheiro, R. Chaves, E. Cerqueira, and A. Abelem, "Cim-sdn: A common information model extension for software-defined networking," in *Globecom Workshops (GC Wkshps), 2013 IEEE*, pp. 836–841, Dec 2013. 2, 14, 23

[34] ISO, "Information technology – Open Systems Interconnection – Systems management overview," ISO/IEC 10040:1998, International Organization for Standardization, Geneva, Switzerland, November 1998. 3, 20

[35] L. Efremova and D. Andrushko, "What's in opendaylight?," April 2015. [Online]. Available: https://www.mirantis.com/blog/whats-opendaylight/. 9

[36] ONF, "Openflow switch specification v1.5.0," Technical Specification TS-020, Open Network Foundation, December 2014. 10, 11

[37] L. Yang, R. Dantu, T. Anderson, and R. Gopal, "Forwarding and control element separation (ForCES) framework." RFC 3746, April 2004. 10, 11

[38] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: Enabling innovation in campus networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, April 2008. 11

[39] H. Song, “Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane,” in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN ’13, (New York, NY, USA), pp. 127–132, ACM, 2013. 11

[40] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Paraulkar, “Flowvisor: A network virtualization layer,” Tech. Rep. OpenFlow-tr-2009-1, OpenFlow Team, Standford University, October 2009. 11

[41] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, “Nox: Towards an operating system for networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 105–110, July 2008. 11

[42] NOXRepo.org, “The POX controller,” October 2013. [Online]. Available: https://github.com/noxrepo/pox. 11

[43] Project Floodlight, “The Floodlight controller,” 2015. [Online]. Available: http://www.projectfloodlight.org/floodlight. 11

[44] Trema, “Trema: Full-stack openflow framework in ruby,” 2015. [Online]. Available: https://github.com/trema/trema. 11

[45] H. Yin, H. Xie, T. Tsou, D. Lopez, P. Aranda, and R. Sidi, “SDNi: A message exchange protocol for software defined networks (SDNs) across multiple domains.” Internet-Draft, December 2012. 11

[46] Project Floodlight, “Floodlight REST API,” April 2015. [Online]. Available: https://floodlight.atlassian.net/wiki/display/floodlightcontroller/Floodlight+REST+API. 11

[47] M. Monaco, O. Michel, and E. Keller, “Applying operating system principles to sdn controller design,” in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, HotNets-XII, (New York, NY, USA), pp. 2:1–2:7, ACM, 2013. 11

[48] S. Murugesan, “Understanding web 2.0,” *IT Professional*, vol. 9, no. 4, pp. 34–41, 2007. 12

[49] T. O’Reilly, “What is web 2.0: Design patterns and business models for the next generation of software,” September 2005. [Online]. Available: http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html. 12

[50] C. Cappiello, F. Daniel, M. Matera, and C. Pautasso, “Information quality in mashups,” *Internet Computing, IEEE*, vol. 14, no. 4, pp. 14–22, 2010. 12

[51] N. Laga, E. Bertin, R. Glitho, and N. Crespi, “Widgets and composition mechanism for service creation by ordinary users,” *Communications Magazine, IEEE*, vol. 50, no. 3, pp. 52–60, 2012. 12

[52] J. Yu, B. Benatallah, F. Casati, and F. Daniel, “Understanding mashup development,” *Internet Computing, IEEE*, vol. 12, no. 5, pp. 44–52, 2008. 12

[53] X. Wang, Y. Chen, and J. Sha, “The development model of web applications based on mashups,” in *Service Operations and Logistics, and Informatics, 2008. IEEE/-SOLI 2008. IEEE International Conference on*, vol. 1, pp. 1059–1062, 2008. 12

[54] D. Nations, “What is web 3.0?,” 2013. [Online]. Available: http://webtrends.about.com/od/web20/a/what-is-web-30.htm. 12

[55] S. Aghaei, M. A. Nematbakhsh, and H. K. Farsani, “Evolution of the world wide web: From web 1.0 to web 4.0,” *International Journal of Web & Semantic Technology (IJWesT)*, vol. 3, pp. 1–10, January 2012. 12

[56] T. Heath and C. Bizer, *Linked Data: Evolving the Web into a Global Data Space*, ch. The Web of Data. Morgan & Claypool, 1st ed., February 2011. [Online]. Available: http://linkeddatabook.com/editions/1.0/#htoc23. 12

[57] T. Berners-Lee, J. Hendler, and O. Lassila, “The semantic web,” *Scientific American*, vol. 284, pp. 34–43, May 2001. 12

[58] B. Balis, B. Kowalewski, and M. Bubak, “Leveraging complex event processing for grid monitoring,” in *Parallel Processing and Applied Mathematics* (R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Wasniewski, eds.), vol. 6068 of *Lecture Notes in Computer Science*, pp. 224–233, Springer Berlin Heidelberg, 2010. 12

[59] M. Potocnik and M. B. Juric, "Towards complex event aware services as part of soa," *IEEE Transactions on Services Computing*, vol. 7, pp. 486–500, July–September 2014. 12

[60] I. S. A. Pereira, P. D. Costa, and J. P. A. Almeida, "A rule-based platform for situation management," in *2013 IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA)*, pp. 83–90, February 2013. 13

[61] M. Jang and J.-C. Sohn, "Bossam: An extended rule engine for owl," in *Third International Workshop on Rules and Rule Markup Languages for the Semantic Web (RuleML)*, pp. 128–138, Springer Berlin Heidelberg, November 2004. 13

[62] JSON, "Introducing JSON." [Online]. Available: http://www.json.org/. 13

[63] M. Lanthaler and C. Gütl, "On using json-ld to create evolvable restful services," in *Proceedings of the Third International Workshop on RESTful Design*, WS-REST '12, (New York, NY, USA), pp. 25–32, ACM, 2012. 13

[64] C. R. P. dos Santos, R. S. Bezerra, J. M. Ceron, L. Z. Granville, and L. M. R. Tarouco, "Botnet master detection using a mashup-based approach," in *Network and Service Management (CNSM), 2010 International Conference on*, pp. 390–393, 2010. 15, 17

[65] R. S. Bezerra, C. R. P. dos Santos, L. M. Bertholdo, L. Z. Granville, and L. R. Tarouco, "On the feasibility of web 2.0 technologies for network management: A mashup-based approach," in *Network Operations and Management Symposium (NOMS), 2010 IEEE*, pp. 487–494, 2010. 15, 17

[66] C. R. P. dos Santos, R. S. Bezerra, J. M. Ceron, L. Z. Granville, and L. M. Rockenbach Tarouco, "On using mashups for composing network management applications," vol. 48, no. 12, pp. 112–122, 2010. 15, 17

[67] C. R. P. dos Santos, R. S. Bezerra, L. Z. Granville, L. M. Bertholdo, W. Cheng, and N. Anerousis, "A data confidentiality architecture for developing management mashups," in *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pp. 49–56, 2011. 15, 17

[68] J. Lopez de Vergara, J. Aracil, J. Martinez, A. Salvador, and J. Hernandez, "Application of ontologies for the integration of network monitoring platforms," in *Proceedings of 1st European Workshop on Mechanisms for Mastering Future Internet*, (Salzburg, Austria), July 2008. 16, 17

[69] J. E. Lopez de Vergara, A. Guerrero, V. A. Villagra, and J. Berrocal, "Ontology-based network management: Study cases and lessons learned," *J. Netw. Syst. Manage.*, vol. 17, pp. 234–254, September 2009. 16, 17

[70] B. Balis, B. Kowalewski, and M. Bubak, "Real-time grid monitoring based on complex event processing," *Future Gener. Comput. Syst.*, vol. 27, pp. 1103–1112, October 2011. 16, 17

[71] DMTF, "Common Information Model (CIM) infrastructure v2.7.0," Specification DSP0004, Distributed Management Task Force, April 2012. 20, 23

[72] J. de Vergara, V. Villagra, J. Asensio, and J. Berrocal, "Ontologies: giving semantics to network management models," *Network, IEEE*, vol. 17, no. 3, pp. 15–21, 2003. 23

[73] DMTF, "Network management profile v1.0.0a," Specification DSP1046, Distributed Management Task Force, April 2014. 23

[74] T. Oetiker, "Mrtg: The multi router traffic grapher," in *Proceedings of the Twelfth Systems Administration Conference*, LISA âĂŹ98, pp. 141–âĂŞ148, USENIX Association, December 1998. 38

[75] T. Oetiker, "Monitoring your it gear: the mrtg story," *IT Professional*, vol. 3, no. 6, pp. 44–48, 2001. 38

[76] S. Lin, Z. Gao, and K. Xu, "Web 2.0 traffic measurement: Analysis on online map applications," in *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '09, (New York, NY, USA), pp. 7–12, ACM, June 2009. 38

[77] T. L. Ruthkoski, *Google Visualization API Essentials*. Community experience distilled, Birmingham, UK: Packt Publishing, Limited, April 2013. 38

[78] R. T. Fielding and R. N. Taylor, “Principled design of the modernweb architecture,” *ACM Transactions on Internet Technology*, vol. 2, pp. 115–150, May 2002. 38

[79] O. M. C. Rendon, F. Estrada-Solano, V. GuimarÃčes, L. M. R. Tarouco, and L. Z. Granville, “Rich dynamic mashments: An approach for network management based on mashups and situation management,” *Computer Networks*, pp. –, 2015. 41, 43

[80] E. Friedman-Hill, *Jess in Action: Rule-Based Systems in Java*. Greenwich, USA: Manning Publications, July 2003. 45

[81] P. Browne, *JBoss Drools Business Rules*. Birmingham, UK: Packt Publishing, April 2009. 45

[82] D. Grigori, J. C. Corrales, M. Bouzeghoub, and A. Gater, “Ranking BPEL processes for service discovery,” vol. 3, no. 3, pp. 178–192, 2010. 46

[83] A. A. Patil, S. A. Oundhakar, A. P. Sheth, and K. Verma, “Meteor-s web service annotation framework,” in *Proceedings of the 13th International Conference on World Wide Web*, WWW ’04, (New York, NY, USA), pp. 553–562, ACM, May 2004. 46

[84] G. A. Miller, “Wordnet: A lexical database for english,” *Commun. ACM*, vol. 38, pp. 39–41, November 1995. 46

[85] R. C. Angell, G. E. Freund, and P. Willett, “Automatic spelling correction using a trigram similarity measure,” *Information Processing & Management*, vol. 19, no. 4, pp. 255 – 261, 1983. 46

[86] B. M. Michelson, “Event-driven architecture overview,” tech. rep., Patricia Seybold Group, 210 Commercial Street, Boston, MA 02109, February 2006. 46

[87] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*. SEI series in software engineering, Addison-Wesley, 2002. 51

[88] P. Shanmugapriya and R. M. Suresh, “Article: Software architecture evaluation methods ï£¡ a survey,” *International Journal of Computer Applications*, vol. 49, pp. 19–26, July 2012. Full text available. 51

[89] J. P. Thompson, “Web-based enterprise management architecture,” vol. 36, no. 3, pp. 80–86, 1998. 53

[90] 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*, Hotnets-IX, (Monterey, California), pp. 19:1–19:6, 2010. 55

[91] D. Kieras, “Using the keystroke-level model to estimate execution times,” unpublished report, University of Michigan, 2001. [Online]. Available: http://www.ict.griffith.edu.au/marilyn/uidweek10/klm.pdf. 56, 87

[92] S. Tian, G. Weber, and C. Lutteroth, “A tuplespace event model for mashups,” in *Proceedings of the 23rd Australian Computer-Human Interaction Conference*, OzCHI ’11, (Canberra, Australia), pp. 281–290, 2011. 56

[93] S. Joines, R. Willenborg, and K. Hygh, *Performance Analysis for Java Websites*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. 57, 77, 87, 88

[94] G. Huang, Y. Ma, X. Liu, Y. Luo, X. Lu, and M. B. Blake, “Model-based automated navigation and composition of complex service mashups,” *IEEE Trans. Serv. Comput.*, vol. 8, p. 494âĂŞ506, May 2015. 70

[95] A. Ordonez, V. AlcÃązar, J. C. Corrales, and P. Falcarin, “Automated context aware composition of advanced telecom services for environmental early warnings,” *Expert Systems with Applications*, vol. 41, no. 13, pp. 5907 – 5916, 2014. 71

[96] O. M. Caicedo Rendon, F. Estrada-Solano, and L. Zambenedetti Granville, “A mashup ecosystem for network management situations,” in *Global Communications Conference (GLOBECOM), 2013 IEEE*, pp. 2249–2255, 2013. 72, 73

[97] Y. Diao and A. Keller, “Quantifying the complexity of it service management processes,” in *Large Scale Management of Distributed Systems* (R. State, S. van der Meer, D. OâĂŹSullivan, and T. Pfeifer, eds.), vol. 4269 of *Lecture Notes in Computer Science*, pp. 61–73, Springer Berlin Heidelberg, 2006. 87

Appendix A

Scientific Production

The research work presented in this thesis was reported to the scientific community through paper submissions to renowned conferences and journals. The process of doing research, submitting paper, gathering feedback, and improving the work helped to achieve the maturity hereby presented.

A.1 Papers: accepted and on reviewing

The list of accepted papers to date is as follows.

1. Oscar Mauricio Caicedo Rendon, **FELIPE ESTRADA-SOLANO**, Vinicius Guimarães, Liane Margarida Rockenbach Tarouco, and Lisandro Zambenedetti Granville. **Rich dynamic mashments: An approach for network management based on mashups and situation management**. Computer Networks, volume 94, pages 285-306, January 15, 2016, ISSN 1389-1286.

    - Status: Published.
    - Colciencias index: A1.
    - Qualis CAPES index: A1.
    - Contribution: M&E Framework.
    - Abstract: In network management, significant research efforts have been carried out to automate and facilitate the tasks conducted by network administrators. However, so far, none of these efforts has exploited the opportunities of jointly using the Situation Management discipline and the mashup

technology for network management. This paper introduces an approach, called Rich Dynamic Mashments, to facilitate the daily work of network administrators when dealing with unexpected, dynamic, and heterogeneous situations. We have referred to as nmsits to such type of network management situations (e.g., a sudden packet loss in a core router of a network backbone and an unforeseen slowness in data transmission over a link between virtual routers) and mashments to tunable mashups that use Situation Management for conducting network management tasks. The proposed approach is made up by the models of nmsits and mashments, the mechanisms to automatically recognize nmsits and dynamically compose mashments, and the architecture supporting these models and mechanisms. We further implement a prototype of our approach and conduct an extensive analysis on networks based on the Software-Defined Networking paradigm. The analysis results have provided directions and evidence that corroborate the feasibility of using Rich Dynamic Mashments as an effective approach for network management in terms of time-recognition, time-composition, time-consuming, time-response, and network traffic.

2. Oscar Mauricio Caicedo Rendon, **FELIPE ESTRADA-SOLANO**, and Lisandro Zambenedetti Granville. **An Approach to Overcome the Complexity of Network Management Situations by Mashments**. The 28th IEEE International Conference on Advanced Information Networking and Applications (AINA 2014), May 13-16, 2014, Victoria, Canada.

   - Status: Published.
   - Qualis CAPES index: A2.
   - Contribution: Case study for SDN Mashup.
   - Abstract: The work performed by network administrators to address sudden, dynamic, heterogeneous, and time specific situations that happen in the network management domain is complex. In this paper, we introduce an approach that allows network administrators to overcome the complexity of handling these network management situations. The approach is made up of Mashments that are special mashups used to cope with nmsits, the process to develop and execute Mashments, and the Mashment Maker that supports

such model and process. We use IT Service Management metrics to evaluate our approach, measuring the complexity of facing, with and without the Maker, a specific nmsit that occurs in several networks based on the Software Defined Networking paradigm. The evaluation results demonstrate that the complexity decreases when network administrators use our approach to handle nmsits.

3. Oscar Mauricio Caicedo Rendon, **FELIPE ESTRADA-SOLANO**, and Lisandro Zambenedetti Granville. **A Mashup Ecosystem for Network Management Situations**. The IEEE Global Communications Conference (GLOBECOM 2013), December 9-13, 2013, Atlanta, United States.

    - Status: Published.
    - Qualis CAPES index: A1.
    - Contribution: Case study for SDN Mashup.
    - Abstract: Current network management approaches and their implementations are not intended to address dynamic situations that need rapid delivery of good-enough and comprehensive solutions. In this paper, we introduce a novel mashup ecosys- tem, called Mashment Ecosystem, that allows Network Administrators to conduct on a Mashment Maker the activities and interactions necessary to provide Mashments. Mashments are mashups aimed to tackle network management situations. We evaluate the Mashment Ecoystem by estimating with the Keystroke-Level Model and measuring in a test scenario the time that Network Administrators take to perform the activities of creating, launching, and publishing Mashments. Similarly, we evaluate the time for retrieving information about a network management situation by using or not Mashments. The evaluation results corroborated that Network Administrators, in our ecosystem, need short-time to deal with network management situations.

4. Oscar Mauricio Caicedo Rendon, **FELIPE ESTRADA-SOLANO**, and Lisandro Zambenedetti Granville. **A Mashup-based Approach for Virtual SDN Management**. The 37th IEEE Annual International Computer Software & Applications Conference (COMPSAC 2013), July 22-26, 2013, Kyoto, Japan.

- Status: Published.
- Qualis CAPES index: A2.
- Contribution: SDN Mashup concept.
- 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 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.

There is other paper that is still under reviewing.

1. **FELIPE ESTRADA-SOLANO**, Armando Ordonez, Lisandro Zambenedetti Granville, and Oscar Mauricio Caicedo Rendon. **A CIM-based Information Model for Heterogeneous SDN Management**. Computer Communications, ISSN 0733-8716.

   - Status: Submitted.
   - Colciencias index: A1.
   - Contribution: Management Plane reference architecture and CIM-based Information Model.
   - Abstract: The Software-Defined Networking paradigm establishes a three-plane architecture that facilitates the deployment of network functions and simplifies traditional network management tasks. However, this architecture lacks an integrated or standardized framework for managing the SDN itself. Some investigations have addressed such shortage by proposing different solutions that tackle specific management requirements for particular SDN

technology instances. This isolated approach forces network administrators to use multiple frameworks to achieve a complete SDN management that is complex and time-consuming in heterogeneous environments. In this paper, we introduce an Information Model based on the Common Information Model that establishes a technology-agnostic and consistent characterization of the whole SDN architecture. Such Information Model represents the core towards building a Management Plane aimed to facilitate the integrated SDN management in heterogeneous environments. To test our Information Model, we developed a prototype and conducted a performance evaluation in an SDN configuration scenario that deploys different managing technologies. The obtained results provide directions that corroborate the feasibility of our approach (in terms of time-response and network traffic) for configuring heterogeneous SDNs.

Computer Networks 94 (2016) 285–306

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Rich dynamic mashments: An approach for network management based on mashups and situation management

Oscar Mauricio Caicedo Rendon[a,*], Felipe Estrada-Solano[a], Vinicius Guimarães[b], Liane Margarida Rockenbach Tarouco[b], Lisandro Zambenedetti Granville[b]

[a] *Telematics Department - University of Cauca, Street 5 # 4–70 - Popayán, CA - Colombia*
[b] *Institute of Informatics - Federal University of Rio Grande do Sul, Av. Bento Gonçalves, 9500 - Porto Alegre, RS - Brazil*

A R T I C L E I N F O

Article history:
Received 30 January 2015
Revised 14 August 2015
Accepted 5 October 2015
Available online 10 November 2015

Keywords:
Mashup
Network management
Situation management
Web-based network management

A B S T R A C T

In network management, significant research efforts have been carried out to automate and facilitate the tasks conducted by network administrators. However, so far, none of these efforts has exploited the opportunities of jointly using the Situation Management discipline and the mashup technology for network management. This paper introduces an approach, called Rich Dynamic Mashments, to facilitate the daily work of network administrators when dealing with unexpected, dynamic, and heterogeneous situations. We have referred to as *nmsits* to such type of network management situations (*e.g.*, a sudden packet loss in a core router of a network backbone and an unforeseen slowness in data transmission over a link between virtual routers) and *mashments* to tunable mashups that use Situation Management for conducting network management tasks. The proposed approach is made up by the models of *nmsits* and *mashments*, the mechanisms to automatically recognize *nmsits* and dynamically compose *mashments*, and the architecture supporting these models and mechanisms. We further implement a prototype of our approach and conduct an extensive analysis on networks based on the Software-Defined Networking paradigm. The analysis results have provided directions and evidence that corroborate the feasibility of using Rich Dynamic Mashments as an effective approach for network management in terms of time-recognition, time-composition, time-consuming, time-response, and network traffic.

1. Introduction

The Situation Management (SM) discipline is intended to address situations happening or that might happen in dynamic systems [1]. SM aims to provide solutions that enable analyzing, correlating, and coordinating interactions among people, information, technologies, and actions targeted to overcome situations [2]. The basis of SM are [3]: (*i*) a situation modeled as a collection of entities in a domain, their attributes, and relationships in a time interval, (*ii*) the investigative aspect related to retrospective cause analysis of situations, (*iii*) the control aspect devised to change or preserve situations; and (*iv*) the predictive aspect aimed to foresee situations.

SM has been employed in diverse domains, such as disaster response [4], smart power grids [5], civil crisis [6], aviation [7], public health [8], electric power systems [9], and medical emergencies [10]. However, up to now, SM has not been used to deal with unexpected, dynamic, and heterogeneous situations that network administrators face in their daily work. Hereinafter, such type of situations are referred

* Corresponding author. Tel.: +57 3104918132.
E-mail addresses: omcaicedo@unicauca.edu.co, omcaicedo@gmail.com, omcrendon@inf.ufrgs.br (O.M. Caicedo Rendon), festradasolano@unicauca.edu.co (F. Estrada-Solano), vtguimaraes@inf.ufrgs.br (V. Guimarães), liane@penta.ufrgs.br (L.M. Rockenbach Tarouco), granville@inf.ufrgs.br (L.Z. Granville).

http://dx.doi.org/10.1016/j.comnet.2015.10.024
1389-1286/© 2015 Elsevier B.V. All rights reserved.

to as *nmsits* [11]. Some examples of *nmsit* are: (*i*) a sudden packet loss in a core router of a network backbone, (*ii*) an unforeseen slowness in data transmission over a link between two virtual routers; and (*iii*) an unexpected increases in the number of corrupted packages transmitted by switches handled by an OpenFlow Controller.

Mashups are composite Web applications centered on end-users and created by combining resources (*e.g.*, data, application logic, and user interfaces) available along the Web [12]. In the previous definition, end-user centric means that mashups may be developed by users without advanced programming skills [13]. Mashups have been used in several domains, such as fire emergencies [14], telco services [15], and immersive mirror worlds [16]. Also, we have analyzed the mashup technology as a mechanism to compose network management applications [17] and accomplish specific tasks like virtual nodes monitoring [18].

Although a large number of research efforts [19–25] has been carried out to support management tasks, to the best of our knowledge, none of such efforts has jointly used SM and mashups to automate and facilitate the work of network administrators. In our previous work [11,26], we introduced the concept of *mashments* (*i.e.*, tunable mashups that automate the investigative and control aspects of SM for carrying out network management). We observed that *mashments* are a suitable approach to facilitate the tasks of network administrators when facing *nmsits*. Nevertheless, we have identified some features that are missing in current *mashments*: (*i*) they are not able to automatically recognize *nmsits*, constraining the analysis and resolution of such situations; and (*ii*) they are not dynamically composed, limiting the overcoming of recognized *nmsits*.

In this paper, we take a step further and introduce Rich Dynamic Mashments (RDM) to facilitate the work of network administrators when facing *nmsits*. We argue that the use of mechanisms to automatically recognize *nmsits* and dynamically compose *mashments* allows to make timely decisions in a less complex way. Our major contributions are

- Mechanisms to automatically recognize *nmsits* and dynamically compose *mashments*.
- An architecture that supports the above mentioned mechanisms and enables building up RDMs.
- A prototype that implements the proposed architecture.
- The demonstration, in a network that follows the Software-Defined Networking (SDN) paradigm, of the feasibility of using RDM to deal effectively with *nmsits* in terms of time-recognition, time-composition, time-consuming, time-response, and network traffic.

The remainder of this paper is organized as follows. In Section 2, we present scenarios and challenges of *nmsits*. In Section 3, we present related work. In Section 4, we introduce the RDM Architecture. In Section 5, we describe and discuss the proof-of-concept used to evaluate RDM. Finally, in Section 6, we provide conclusions and implications for future work.

2. Scenarios and challenges

In this section, we introduce motivating scenarios and their challenges. In the first scenario, let's consider network administrators manage a large company. In this company, the communication between the Pin Pads shops and the Enterprise Resource Planning system is provided by an outsourced Internet Service Provider (ISP). If a sudden failure in packet transmission in the ISP takes place or if an internal connection error in the border router of one or more shops occurs, the company might lose a significant amount of revenues because the payment by cards becomes inoperative.

In the second scenario, let's assume network administrators manage an SDN-based Network Operator. This operator provides network infrastructure to Small and Medium Enterprises (SMEs) using resources, such as OpenFlow controllers and virtual switches, supplied by several Virtual Network Providers. If an abrupt performance degradation in virtual links of one or more SMEs (*e.g.*, generated by unidentified errors in the virtual switches of providers) occurs, the operator might break the Service Level Agreements established with SMEs and, as a result, lose money.

In these both scenarios, network administrators need to easily and rapidly overcome *nmsits*. In particular, they face the following challenges: (*i*) conduct situational management operations (*e.g.*, collect, split, filter, add, and merge data) on multiple and heterogeneous devices/networks involved in *nmsits*, (*ii*) compose and tune solutions for *nmsits* in a less complex and time consuming way; and (*iii*) visualize information of *nmsits* in an understandable and friendly way.

The challenges of *nmsits* may be addressed, among other, with the following options. The first one is to use several mismatched network management solutions from the industry [27,28] and academy [24] [25], but it hinders and overloads the tasks of network administrators. Thus, this option is complex and time consuming.

A second option to cope with *nmsits*, it is to improve existing network management solutions, such as Nagios [29], ZenOSS [27], and OpenNMS [28], by deploying on them plugins that support SM. To the best of our knowledge, up to now, there is not a research that develops and adds plugins/packages based on the SM discipline to extend and enhance the above-referred solutions.

A third option to deal with *nmsits*, it is to use homebrewed scripts that integrate two or more network management solutions. The weaknesses of this option are, first, the skill required to write and run scripts (the development of scripts is a daunting and complex task for network administrators who usually do not have advanced programming knowledge). Second, the loss of focus and time of network administrators; instead of management the network itself, they are forced to acquire knowledge that is not necessarily relevant to their daily tasks.

A fourth option to address *nmsits*, it is to use our *mashments* [11,26]. In a broad sense, a mashment is a solution based on the SM discipline and the mashup technology for carrying out network management in an effective way. In particular, a mashment is defined as a tunable mashup that combines diverse types of resources from multiple providers and automates the investigative and control aspects of SM, aiming to facilitate the work of network administrators.

We argue that *mashments* are closer and more appropriate to facilitate the daily needs of network administrators. Notwithstanding, as the network administrator is even

Table 1
Proposals on SM.

Ref	Description	Domain	Aspect		
			Investigative	Control	Predictive
[4]	An architecture based on Wireless Sensor Networks and Delay-Tolerant Networking to aid disaster responders in making decisions	Disaster response	✓	✓	✓
[5]	An architecture based on SOA that uses linked open data to meet the continuous changes in the electricity usage patterns of customers	Smart power grids	✓	✓	
[6]	A mobile agent platform to provide timely and protected access to situational information for on-site personnel and tactical center	Civil crisis	✓	✓	
[7]	An approach for deploying distributed systems that aim to assist a timely and correct making-decision during air security incidents	Aviation	✓	✓	
[8]	A rule-based platform supporting the development of situation-aware applications for monitoring suspicious cases of tuberculosis	Public health	✓	✓	✓
[9]	A situation awareness application aimed to face major disturbances (*e.g.*, snow storms) suffered by distribution system operators in Finland	Electric power systems	✓	✓	
[10]	A system based on finite-state machines and complex event processing to enhance information availability in emergency medical services	Medical emergencies	✓	✓	

responsible for identifying *nmsits* and creating the workflow of current *mashments*, the *mashment*-based approach still keeps some complexity and consumption of time. In this paper, the RDM-based approach is introduced to overcome the shortcomings and weaknesses of options previously described.

3. State-of-the-art

This section reviews the three top topics related to the RDM-based approach: situation management, mashup technology, and network management. To visit the related work on the SM literature, we focus on the SM fundamental aspects [3,30]: investigative, control, and predictive. The investigative aspect related to carrying out actions intended to comprehend situations (*i.e.*, what is happening?). The control aspect associated with creating plans for overcoming situations (*i.e.*, how to solve a situation?). The predictive aspect related to conducting actions for foreseeing future situations by taking into account past situations (*i.e.*, what is going to happen?).

Table 1 presents relevant proposals in SM by considering the application domain and the SM aspects. This table reveals that the most of SM-based proposals focuses on the investigative and control aspects. That is so because such aspects are the basis to carry out the predictive one. It is noteworthy that although SM has been used in different domains, ranging from disaster response to public health, none of SM-based proposals has been targeted to automate or facilitate tasks in network management.

Regarding mashups, it is important to mention that their use has been disseminated over the last decade thank mainly to two facts [31]: (*i*) the number of online services, widgets, and APIs rose significantly; and (*ii*) new usability-oriented technologies (*e.g.*, frameworks JavaScript, HTML5, and Web Sockets) arose to allow the creation of more dynamic applications and advanced Graphical User Interfaces (GUIs) by end-users.

Thanks the dissemination of the mashup technology, mashups mainly involving mapping services have been used in diverse domains, ranging from fire emergencies to network management. Table 2 presents relevant proposals in the mashup technology by considering its application domain and evaluated characteristics. This table reveals two facts: (*i*) the most of mashup-based proposals neither evaluates the complexity nor the time involved in the tasks supported by mashups; and (*ii*) none of proposals that uses mashups for network management employs the SM aspects.

We are pioneers in investigations that involve SM and mashups to automate and facilitate the daily tasks of network administrators. However, in the literature there are proposals for network management that uses other points-of-views. Table 3 presents relevant proposals in the network management domain by considering their main mechanism (*i.e.*, Agent, Rule, Policy, SOA, and BPM). This table reveals that in proposals such as COOLAID, NetOpen, Pyretic, and Procera, network administrators are in charge of writing/programming policies, rules, or basic services using specific languages or controllers. Thus, when using such proposals, the work of the network administrator remains complex and consumes a lot of time, hindering their use as situational solutions. In turn, OMNI and MEICAN are proposals that provide friendly GUIs to facilitate several network management tasks, but they were not conceived to be extended or improved by network administrators. Therefore, these proposals also are constrained when used as situational solutions.

Unlike the reviewed proposals, we consider concepts from SM and mashups to facilitate the handling of unexpected, dynamic, and heterogeneous situations happening in network management. It is important to highlight that in addition to the traditional metrics (*e.g.*, network traffic and time of response) evaluated in such proposals, we evaluate the

Table 2
Proposals on mashups.

Ref	Description	Evaluated characteristics				
		Domain	SM	Extensibility	Flexibility	Time of tasks
[14]	A mashup providing information of active fires, evacuation routes, and available hospitals	Fire emergencies		✓	✓	
[15]	An architecture that aims to facilitate the provisioning of telco mashups for end-users	Telco services		✓	✓	
[16]	A platform to create mashups aimed to show immersive street-level depictions of the physical world	Immersive mirror worlds		✓	✓	✓
[32]	A system that allows the creation of new musical content by developing multi-song mashups	Music		✓	✓	
[33]	A mashup-based approach to face the botnets security problem in a more flexible and extensible way	Network management		✓	✓	
[34]	A mashup system to qualitatively evaluate the feasibility of using Web 2.0 for network management	Network management		✓	✓	
[17]	A generic architecture to support the static composition of network management applications	Network management		✓	✓	

Table 3
Proposals on network management.

Ref	Description	Agent	Rule	Policy	SOA	BPM
[19]	The COnfiguring cOmpLex and dynamic networks AutomatIcally and Declarative (COOLAID) is a data-centric framework aimed to network configuration operations		✓			
[21]	The OpenFlow MaNagement Infrastructure (OMNI) proposes a multi-agent system that offers an specific API to build up applications for managing OpenFlow-based networks	✓				
[20]	NetOpen is a solution that permits to create network composite services, by combining networking primitives, aimed to monitor and configure OpenFlow-based networks				✓	
[22]	MEICAN is a solution that offers a Web-based friendly GUI to allow network administrators to participate in the provisioning process of virtual inter-domain circuits					✓
[23]	Pyretic is a domain-specific and imperative language built over Python, which supports the development of modular applications for managing OpenFlow-based networks			✓		
[24]	Procera is a control framework that provides a language and an engine for creating and executing policy-based applications intended to control and monitor SDN-based networks			✓		
[25]	A framework based on policy-controlled management patterns that offers abstractions for orchestrating OpenFlow applications by combining individual resilience services			✓		

time of recognition of *nmsits*, the time of composition of rich dynamic *mashments*, and the time spent by network administrators when facing situations.

4. Rich dynamic mashments

To detail our approach, initially, we present the model of *nmsits*. Afterwards, we introduce the model of RDMs. Finally, we present the RDM Architecture. Before introducing the models, Table 4 defines the most important abbreviations used to describe the proposed approach.

4.1. Fundamental models

We use JSON to represent the models of *nmsits* and RDMs. That is so because JSON is more lightweight than the eXtensible Markup Language (XML) [35]. In such models, capital and lowercase letters indicate names and values of JSON properties, respectively.

Model of nmsits. The *nmsits* are unexpected, dynamic, and heterogeneous situations that network administrators face in their daily work. Fig. 1(a) presents the model of *nmsits*.

The model of *nmsits* encoded in JSON is:

[{*NAMESIT*: *namesit*, *NMSIT*: [{$nmsit_1$}, {$nmsit_n$}]}]

Where, *NAMESIT* represents the global name of *NMSIT* that is formed by several $nmsit_n$. A single $nmsit_n$, in turn, is:

[{*SITUATION*: *situation*, *EAC*: [{*eac*}]}]

Where, *SITUATION* is the specific name of $nmsit_n$ and *EAC* is the collection of entities, attributes, and constraints involved in such $nmsit_n$. Each *entity* has a collection of attributes and their corresponding constraints. Note that these constraints serve to determine when $nmsit_n$ happens.

Examples of a single *nmsit* are:

$(-)$ $namesit = \{drop\ of\ received\ packages\}$,
$situation = \{sudden\ drop\ of\ received\ packages\ in\ virtual\ router\}$, and
$eac = \{ENTITY : vyattaRouter,$
$[\{ATTRIBUTE : receivedPkg, CONSTRAINT :< 95\%\}]\}$

Table 4
Abbreviations.

Abbreviation	Meaning	Explanation
EAC	Entity attribute constraint	A collection of entities, attributes, and constraints involved in a *nmsit*
NMR	Network management resource	An entity intended to conduct network management operations
NMRS	NMR as a service	A service that offers the functionalities provided by NMR
NMSIT	Network management situation	An unexpected, dynamic, and heterogeneous situation on network management
NSR	Network situational resource	An entity that provides functionalities aimed to automate SM aspects
NSRS	NSR as a service	A service that offers the functionalities provided by NSR
OpR	Operator resource	An entity suitable to combine resources and, so, to create RDMs
OpRS	OpR as a service	A service that offers the functionalities provided by OpR
RDM	Rich dynamic mashment	A solution based on SM and mashups for carrying out network management
SM	Situation management	A discipline to addresses situations in dynamic systems
SMR	Situation management resource	An entity that provides interaction to and from entities involved in *nmsits*
SMRS	SMR as a service	A service that offers the functionalities provided by SMR
WMR	Web-based network management resource	A Web entity conceived or that can be used for network management
WMRS	WMR as a service	A service that offers the functionalities provided by WMR

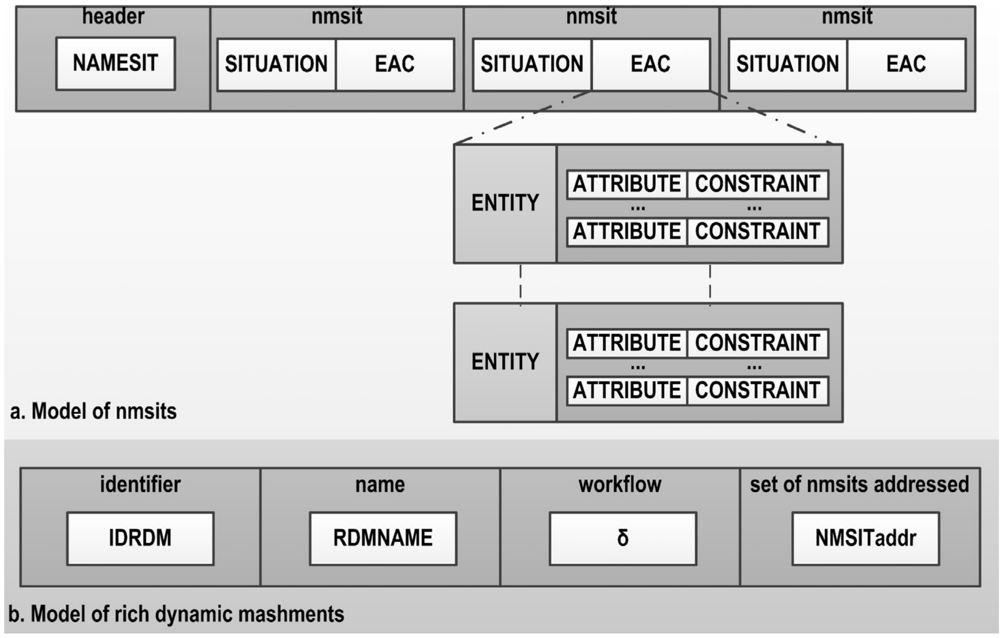


Fig. 1. Fundamental models.

$(-)$ *namesit* = {*drop of sent packages*},
situation = {*unexpected drop of sent packages in a switch*}, and
$eac = \{ENTITY : cysco2960,$
$[\{ATTRIBUTE: sentPkg, CONSTRAINT: <90\%\}]\}$

An example of a *nmsit* formed by two *nmsits* is:

(-) *namesit* = {*link has overload*},
$situation_1$ = {*switch has overload in memory and processor*},
$eac_1 = \{ENTITY : cysco1000,$
$[\{ATTRIBUTE: mem, CONSTRAINT: >95\%\},$
$\{ATTRIBUTE: processor, CONSTRAINT: >97\%\}]\},$
$situation_2$ = {*switch has overload in memory and processor*},
$eac_2 = \{ENTITY : openvSwitch,$
$[\{ATTRIBUTE: mem, CONSTRAINT: >96\%\},$
$\{ATTRIBUTE: processor, CONSTRAINT: >98\%\}]\}.$

Model of RDMs. RDMs are tunable composite solutions based on the SM discipline and the mashup technology for carrying out network management. An RDM is characterized by: (*i*) it recognizes automatically *nmsits*; and (*ii*) its workflow is generated without direct intervention of network administrators. Fig. 1 (b) presents the model of RDMs that encoded in JSON is:

$[\{IDRDM: id, RDMNAME: name, \delta: [\{delta\}], NMSIT_{addr}: [\{nmsit_{addr}\}]\}]$

Where, *IDRDM*, *RDMNAME*, $NMSIT_{addr}$, and δ are the unique identifier, the friendly name, the set of *nmsits* addressed, and the workflow of an specific RDM, respectively. In turn, δ is:

$[\{IDD: idd, RES: [\{res_1\}, \{res_n\}], CONN: [\{conn_1\}, \{conn_n\}]\}]$

Where, *IDD* is the unique identifier of δ, *RES* is the set of resources that form a particular RDM, and *CONN* is the set

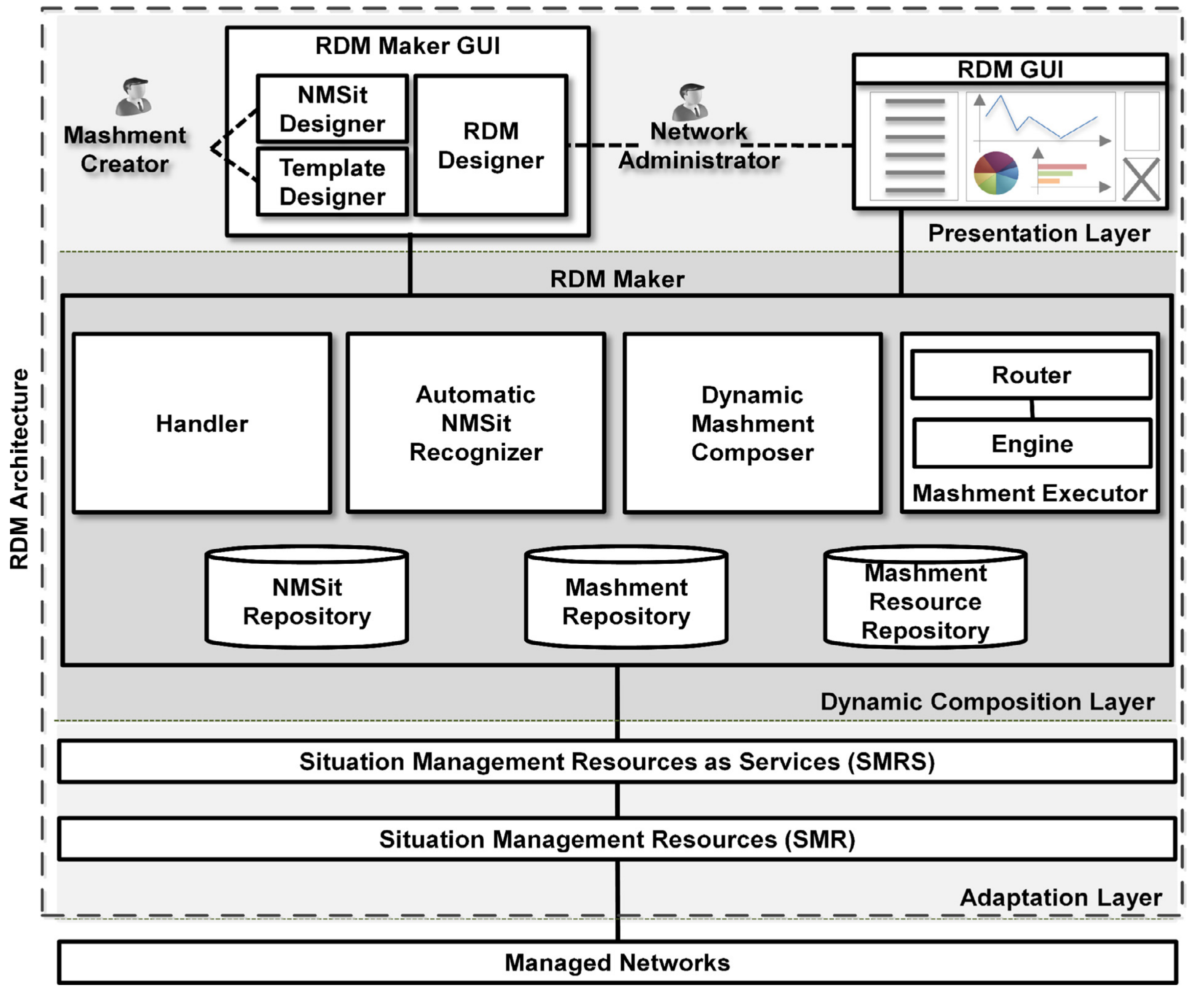


Fig. 2. Proposed architecture.

of logical connections created among these resources. Note that *RES* and *CONN* define how $NMSIT_{addr}$ are handled (*i.e.*, investigated and/or resolved).

Resources are helpful entities to interact (*e.g.*, to access, communicate, and send-retrieve information) with network elements involved in *nmsits* and for presenting, in an intelligible way, information about *nmsits*. A res_n is:

[{*IDRES*: *idres*, *RESNAME*: *resname*, *OPERATION*: [{op_1, op_n}]}]

Where, *IDRES* and *RESNAME* are the unique identifier and the name of the resource, respectively. *OPERATION* is the collection of operations offered by the resource. An op_n is:

[{*OPNAME*: *opname*, *PARAM*: [{par_1, par_n}], *PROD*: *produce*}]

Where, the name of the operation, the set of parameters need to invoke the operation, and the data type that the operation produces are, correspondingly, represented by *OPNAME*, *PARAM*, and *PRODUCE*.

Connections define the propagation of data between resources by specifying which outputs of a resource are inputs of other resources. A $conn_n$ is:

[{*IDC*: *idc*, *SRC*: [{*idresS*, *idparS*}], *DES*: [{*idresD*, *idparD*}]}]

Where, *IDC* is the unique identifier, *SRC* is the source resource, and *DES* is the destination resource of the connection, respectively. The resource (*idresS* or *idresD*) and the parameter (*idparS* or *idparD*) represents the connection among *SRC* and *DES*.

4.2. Architecture

The RDM Architecture (see Fig. 2) leverages the main foundations of the SM discipline and the mashup technology to facilitate the work of network administrators. In particular, this architecture offers: (*i*) the automatic recognition of *nmsits* by using matching pattern algorithms; and (*ii*) the dynamic creation of RDMs by using composition templates.

Two actors (*i.e.*, Mashment Creators and Network Administrators) are involved in tackling *nmsits*. The Creators are mainly responsible for: (*i*) creating models/patterns of *nmsits* that will be automatically recognized by the proposed approach, (*ii*) building up composition templates that will be dynamically customized by the RDM-based approach for addressing *nmsits* recognized; and (*iii*) customizing and extending rich dynamic *mashments*. In turn, an Administrator is fundamentally in charge of coping with *nmsits* by using RDMs as well as customizing and extending them.

Fig. 2 depicts the RDM Architecture and the networks that it can manage. Three layers made up this architecture: (*i*) the Adaptation Layer that hides the intricacy and heterogeneity of Managed Networks (*e.g.*, SDN-based, NVE, and traditional networks), (*ii*) the Dynamic Composition Layer that recognizes *nmsits* and builds RDMs to face such situations; and (*iii*) the Presentation Layer that depicts the GUIs of RDMs and the RDM Maker. The next paragraphs present in detail these architectural layers.

4.2.1. Adaptation layer

This layer provides services to the Dynamic Composition Layer and offers as a mashable resource (*i.e.*, services) any entity that is useful to deal with *nmsits*. The elements that form the Adaptation Layer are the set of Situation Management Resources (SMR) and its respective representation as a service (SMRS). An SMR is an entity that provides interaction to and from network elements or entire networks involved in *nmsits*.

There are five types of SMR: Network Management Resource (NMR), Web-based Network Management Resource (WMR), Analytics Management Resource (AMR), Network Situational Resource (NSR), and Operator Resource (OpR). An NMR is an entity intended to conduct network management operations. Examples of NMR are Ganglia [36], Nagios [29], and ZenOSS [27] to manage traditional networks, Citrix Center [37] for monitoring virtual resources, NetOpen [20] and OMNI [21] to control OpenFlow-based networks, monitoring systems based on SNMP, and all APIs that provide interaction with network elements.

A WMR is a Web entity conceived or that can be used to perform network management tasks. Examples of WMR are the Multi Router Traffic Grapher (MRTG) [38] for generating Web pages with images presenting the traffic of network links, the RRDTool [39] for displaying over time the performance data of routers, the Yahoo Maps API [40] for showing the geographic location of several network devices, and the Google Chart API [41] for depicting the memory consumption of virtual switches.

An AMR is an entity intended to analyze network management information. Examples of AMR are the Management Traffic Analyzer [42] to interpret the functioning of network devices supporting SNMP, the Junos Network Analytics Suite [43] to understand what is happening on networks using Junos devices, and the Sandvine Network Analytics [44] to get right-time information of networks regardless of underlying technologies.

An NSR is an entity that provides functionalities aimed to automate the aspects of SM. These functionalities are: (*i*) collecting to retrieve information about *nmsits*, (*ii*) fusing&correlating to merge and correlate the information retrieved by collecting, supporting the creation of investigative plans (*i.e.*, workflows useful to determine the cause of *nmsits*); and (*iii*) resolving to enable conducting network management operations aimed to change or preserve *nmsits*, assisting the creation of resolutive plans (*i.e.*, workflows helpful to solve *nmsits*).

Examples of NSR are JESS [45], JBOSS Drools [46], and Apache Camel [47]. The JESS is a general-purpose platform that permits detecting and controlling situations by rules (defined using XML or the JESS Rule Language) and Java applications. The JBOSS Drools is a solution that allows recognizing and controlling generic situations by rules (defined using Drools Rule Language - DRL) and Java applications. The Apache Camel is a platform that enables processing events (*i.e.*, generic situations) from multiple sources employing a Complex Event Processor (CEP) based on Java.

An OpR is an entity that allows combining resources and, so, building up and generating RDMs. There are three OpR types: (*i*) control patterns (*e.g.*, sequential, parallel, conditional, and templates) that allow defining the workflow of RDMs, (*ii*) structures for configuring and invoking (*e.g.*, functionalities to set security credentials of virtual routers) the resources that form RDMs; and (*iii*) structures for receiving, sorting, and filtering (*e.g.*, functionalities to perform information selection on text-plain containing data of NVEs) the retrieved information from any type of resource.

SMRS are mashable entities that offer as a service network management operations of one or more NMR, WMR, AMR, NSR, and OpR, aiming to hide the complexity of these resources. The representation of resources as services consists in defining and providing a common data format to interchange information of resources, well-known interfaces to resources, and a common protocol to communicate with such interfaces.

Types of SMRS are NMR as a Service (NMRS), WMR as a Service (WMRS), AMR as a Service (AMRS), NSR as a Service (NSRS), and OpR as a Service (OpRS). An example of NMR is the Floodlight Controller API to handle switches OpenFlow, the associated NMRS is the Floodlight Service that allows, via requests-and-responses HTTP, to monitor these switches.

Internally, a Wrapper and a UWrapper made up an SMRS. A Wrapper is a service based on the REpresentational State Transfer (REST) [48] architectural model. In turn, a UWrapper is a URI pointing to one Wrapper. Since every Wrapper is based on REST, in the RDM architecture, the set of SMRS provides a mediator bus in which the communication is conducted by following the request-response model of HTTP. This bus enables the interaction between the layers of Adaptation and Dynamic Composition.

The Adaptation Layer responds to HTTP requests from the Dynamic Composition Layer as follows. First, the requests are targeted to UWrappers. Second, Wrappers invoke one or more SMR by using protocols (*e.g.*, SNMP, SOAP, HTTP, OpenFlow, and Proprietary) provided by network vendors for managing their solutions. Third, each invoked SMR carries out the requested functionalities by performing operations in the Managed Networks. Fourth, Wrappers receive responses from the invoked SMR. Fifth, Wrappers encode SMR results on JSON data and put such data on HTTP responses. Sixth, Wrappers send their HTTP responses to the Dynamic Composition Layer.

4.2.2. Dynamic composition layer

This layer offers RDMs to the Presentation Layer. In the Dynamic Composition Layer is the RDM Maker that supports: (*i*) the definition of *nmsit* patterns, (*ii*) the recognition of *nmsit* patterns, (*iii*) the specification of composition templates;

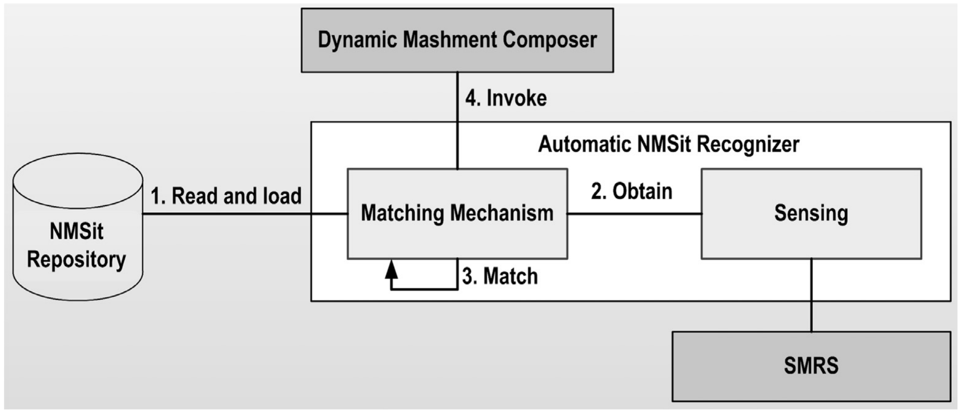


Fig. 3. Automatic recognition of nmsits.

and (*iv*) the generation from templates of RDMs that face *nmsits* recognized. The Mashment Resource Repository, NMSit Repository, Mashment Repository, Handler, Automatic NMSit Recognizer, Dynamic Mashment Composer, and Mashment Executor form the RDM Maker.

The Mashment Resource Repository stores metadata that describes and points out functionalities offered by SMRS. NMRS, AMRS, OpRS, and NSRS point out to services located on privative repositories accessible only by network administrators. In turn, WMRS by definition points out services available on the Web. A metadata of SMRS is an instance of res_n. The following example of metadada describes two operations provided by a particular NMRS. These operations are to get the list of virtual switches and get the statistics of a virtual switch by an OpenFlow Controller.

$(-)$ [{*IDRES*: */path*$_1$, *RESNAME*: *nmrs*$_1$,
OPERATION : [{*OPNAME* : *SwitchList*,
PARAM : [{*IPCTRL* : *ipctrl*, *PORT* : *port*, *USER* :
user, *PWD* : *pwd*}],
PROD: *json*},
{*OPNAME* : *SwitchStat*,
PARAM : [{*IPCTRL* : *ipctrl*, *PORT* : *port*, *USER* :
user, *PWD* : *pwd*, *IDSWITCH* : *ids*}],
PROD: *json*}]}]

The NMSit Repository stores *nmsit* patterns defined by Mashment Creators. A pattern is an instance of the *nmsits* model that represents a collection of entities, attributes, and constraints. These constraints are conditions defined in attributes of entities for detecting *nmsits*. Thus, this repository contains the patterns that allows recognizing *nmsits*.

The Mashment Repository stores metadata of composition templates and RDMs that handle the recognized *nmsits*. A metadata of RDM stored in this Repository is an instance of the RDM model (see Section 4.1). It is relevant to point out that, first, if several RDMs constitute one RDM, its metadata includes the metadata of them. This inclusion means that a δ may encompass other δs. Second, several SMRS can be used and connected to form a δ that defines how to face one or more *nmsits*. Third, as our approach inherits the end-user composition model from mashups, network administrators can also customize and enhance RDMs.

The composition templates are useful skeletons to automatically compose RDMs. The metadata of templates is:
[{*IDTEMPLATE*: *id*, *NAMESIT*: *namesit*, δ: *delta*}]
Where, *IDTEMPLATE* is the unique identifier of template and δ is a predefined workflow (*i.e.*, an investigative and/or resolutive plan) to deal with $NMSit_{addr}$ identified by *NAMESIT*.

The Handler manages (*i.e.*, search, create, update, and delete) the metadata of *nmsits* and composition templates by manipulating the repositories of NMSit and Mashment. The Mashment Creator handles in the GUI of the NMSit Designer the *nmsits* metadata. The Mashment Creator manages in the GUI of the Template Designer the metadata of composition templates. The Handler also manages the Mashment Resource Repository to support the enhancement and improvement of RDMs that can be conducted by network administrators in the GUI of the RDM Designer.

The repositories described above support the generation of RDMs. Such generation follows two phases:

(1) *when* $<$ *nmsits* $>$ — mechanism to automatically recognize *nmsits*.
(2) *then* $<$ *rdms* $>$ — mechanism to dynamically compose *mashments*.

The first phase is the automatic recognition of *nmsits* by using matching pattern algorithms. The second phase is the dynamic composition of RDMs by composition templates.

The Automatic NMSit Recognizer (see Fig. 3) conducts the automatic recognition of *nmsits*. The modules Sensing and Matching Mechanism form the Recognizer. The Sensing is in charge of retrieving network management information by SMRS and delivering such information as streaming to the Matching Mechanism. The retrieving of information depends on communication model (pull or push) provided by SMRS.

The Matching Mechanism recognizes a *nmsit* as follows. First, it reads and loads *nmsit* patterns from the NMSit Repository. Second, it obtains information from Managed Networks and their devices by Sensing. Third, it conducts matching operations (*i.e.*, comparison of samples vs patterns) among the network information and the loaded *nmsit* patterns. RETE [45] and PHREAK [46] are algorithms that can be used to carry out this matching. Fourth, every time a *nmsit* is

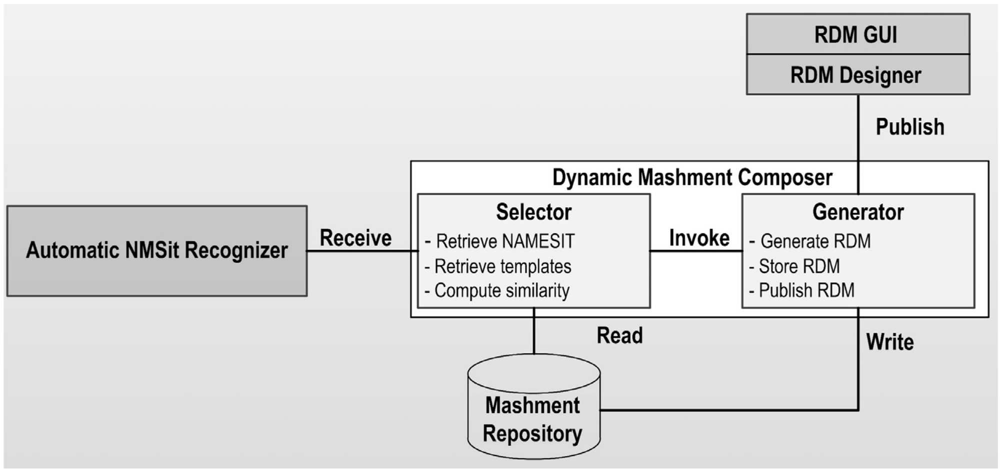


Fig. 4. Dynamic composition of mashments.

detected (*i.e.*, there is match), this mechanism defines $NMSIT_{addr}$ and invokes the Dynamic Mashment Composer.

The Dynamic Mashment Composer (see Fig. 4) performs the dynamic composition of RDMs. This Composer creates RDMs by automating our process to develop and launch *mashments* [26]. Specifically, it automates the tasks Select, Configure, and Combine of such process. Select is to define the resources (*i.e.*, *RES*) to be used to generate RDMs. Configure is to provide all functioning settings of resources selected. Combine is to define how a particular RDM will face an specific $NMSIT_{addr}$ by creating diverse connections (*i.e.*, *CONN*) among selected and configured resources.

The modules Selector and Generator form the Dynamic Mashment Composer. The Selector operates as follows. First, it receives $NMSIT_{addr}$ from the Automatic NMSit Recognizer. Second, it retrieves *NAMESIT* from $NMSIT_{addr}$. Third, it retrieves composition templates by reading the Mashment Repository. Fourth, it selects the δ (*RES* and *CONN*) for such $NMSIT_{addr}$. This selection is carried out by calculating the highest linguistic similarity among the *NAMESIT* of composition templates and the retrieved from $NMSIT_{addr}$. Such calculation is conducted by using the linguistic similarity algorithm [49] that is based on NGram [50], CheckSynonym [51], and ElementMatch [52]. Fifth, it invokes the Generator.

The Generator operates as follows: (*i*) it receives δ and $NMSIT_{addr}$ from the Selector, (*ii*) it uses such δ and $NMSIT_{addr}$ to generate an instance of the RDM model, (*iii*) it stores in the Mashment Repository the RDM generated by writing the corresponding metadata; and (*iv*) it publishes such RDM in the RDM Designer of the RDM Maker GUI. It is noteworthy that using such Designer, the network administrator can enhance, improve, and run RDMs.

The Mashment Executor executes RDMs. The Mashment Router and the Mashment Engine form the Executor. The Router is in charge of coordinating the execution of δs that are the core of RDMs. Thus, on runtime, the Router: (*i*) it receives invocations from the Engine, which means that the Router is called by the Engine to select RDMs to serve initial requests, (*ii*) it selects and links multiple resources (including one or more RDMs into another RDM) to attend invocations, by reading information from the repositories of Mashments and Resources; and (*iii*) it calls the Engine to request the instantiation of RDMs and their underlying resources.

The Mashment Engine is a lifecycle manager responsible for creating, deleting, and caching instances of RDMs. When initial requests to execute RDMs arrive from a browser, the Engine invokes the Router. Afterwards, the Engine waits indications from the Router to manage the instances of RDMs and their constitutive resources.

4.2.3. *Presentation layer*

This layer executes and presents, in the client-side, the RDM Maker GUI and the RDM GUI. The NMSit Designer, the Template Designer, and the RDM Designer form the RDM Maker GUI. All these GUIs are accessible by Web browsers.

The NMSit Designer is a Web-based friendly GUI in which Mashment Creators define *nmsit* patterns to be recognized. The Template Designer is another GUI where Mashment Creators specify composition templates used to cope with *nmsits*. The RDM Designer is the GUI in which network administrators can enhance, save, delete, and run RDMs.

The RDM GUI represents the visualizations of compositions generated by the RDM Maker. Examples of visualizations useful for network management are [53]: (*i*) a 2D scatterplot chart to present information about packet errors in a virtual router involved in a *nmsit*; and (*ii*) a link/node representation to display a network topology (*e.g.*, a glyph-based representation) and 2D charts (*e.g.*, line and bar charts) to provide additional information of corresponding nodes and links.

5. Proof-of-concept

To assess our approach, first, we implemented the RDM System prototype that is an instance of the architecture described in the previous chapter. Second, we built a test environment. Third, we conducted a late evaluation of our architecture. A late evaluation of a software architecture takes place when its implementation (*e.g.*, the RDM System prototype) is complete [54].

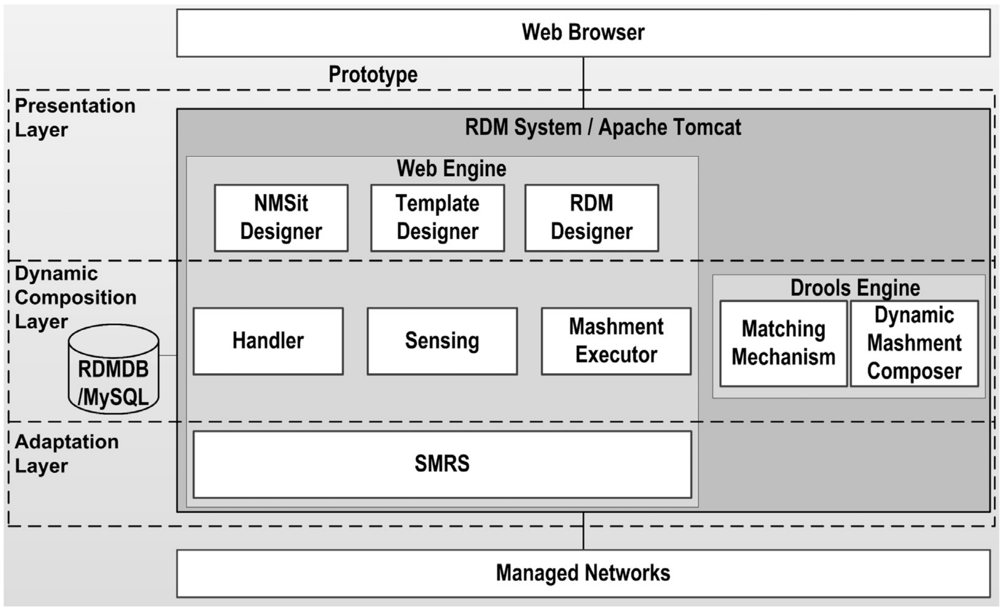


Fig. 5. RDM system prototype.

The late evaluation of the proposed approach is performance-based [55] and intended to determine its feasibility in terms of time-recognition, time-composition, time-consuming, time-response, and network traffic. The time-recognition and time-composition related to the mechanisms for automatic recognition of *nmsits* and dynamic composition of *mashments*, respectively. The time-consuming associated with the process that network administrators need to carry out for facing *nmsits*. The time-response and traffic related to the behavior during runtime of solutions used to handle *nmsits*.

5.1. Prototype

Fig. 5 depicts the RDM System prototype and the Managed Networks. The prototype was built upon the Mashment Maker [11] [26] and deployed by using a MySQL Server and an Apache-Tomcat Server (running a Web Engine and a Drools Engine). The Maker is a browser-based mashup development environment that provides functionalities for assisting Mashment Creators and network administrators in the creation, reuse, and launching of traditional *mashments*.

Presentation layer. The Web Engine deploys the NMSit Designer, the Template Designer, and the RDM Designer. These designers were built using the Yahoo User Interface (YUI) API and the Google Chart API. These APIs are based on the Asynchronous Javascript and XML (AJAX), granting dynamic and interactive interaction of all GUIs of designers with SMRS.

Dynamic composition layer. The Web Engine also deploys the modules Sensing, Handler, and Mashment Executor. These modules are services implemented by using the Java Language and following the REST architectural model [56]. REST was used because it is the de-facto model for developing mashups. Specifically, we implemented: (*i*) Sensing (*i.e.*, SensorService) for interacting with Managed Networks, (*ii*) Handler (*i.e.*, HandlerService) for managing the Mashment Repository, the NMSit Repository, and the Mashment Resource Repository; and (*iii*) Mashment Executor (*i.e.*, ExecutorService) for controlling the execution and lifecycle of RDMs.

The Drools Engine deploys the Matching Mechanism and the Dynamic Mashment Composer. The Matching Mechanism is a Java-based application that uses for recognizing *nmsits* the implementation of the PHREAK algorithm offered by Drools. Furthermore, this mechanism translates from JSON to DRL the *nmsit* patterns stored in the RDMDB. This translation is needed because Drools only understand DRL.

The Dynamic Mashment Composer is also a Java-based application that customizes composition templates. It is to note that, first, the Mashment Creator defines these templates in the Template Designer that saves them in JSON format in RDMDB. Second, once a template has been customized, it is also stored using JSON in RDMDB and, further, automatically exposed in the RDM Designer like a visual element.

RDMDB is a unique database that implements the Mashment Resource Repository, the Mashment Repository, and the NMSit Repository. A MySQL Server deploys such database.

Adaptation layer. SMRS are also services that follow the REST architectural model. In particular, REST-based services (*i.e.*, POXService, BeaconService, and FloodlightService) enables the interaction with the Managed Networks of this proof-of-concept.

5.2. Managed networks

In the proof-of-concept, we chose to manage SDN-based networks because of their commercial and investigative significance. SDN proposes an architecture for future networks

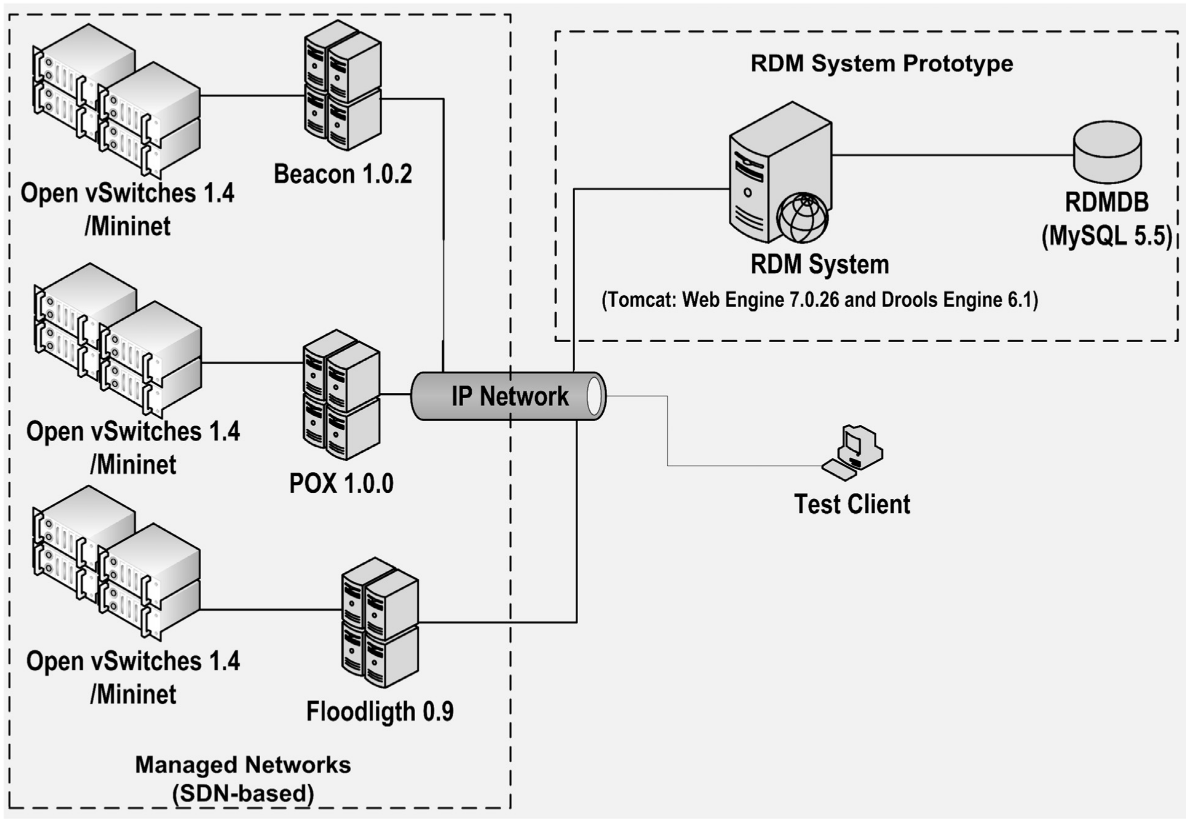

Open vSwitches 1.4 /Mininet
Beacon 1.0.2
RDM System Prototype
RDM System
(Tomcat: Web Engine 7.0.26 and Drools Engine 6.1)
RDMDB (MySQL 5.5)
Open vSwitches 1.4 /Mininet
POX 1.0.0
IP Network
Test Client
Open vSwitches 1.4 /Mininet
Floodligth 0.9
Managed Networks (SDN-based)

Fig. 6. Test environment.

[57], which separates data and decision policies to simplify the network operation. SDN-based networks follow three layers [58–60]: (*i*) the packet forwarding datapath (*e.g.*, switches and routers passing packets), (*ii*) the Network Operating System (NOS) that controls such datapath by using a vendor-independent protocol; and (*iii*) the Network Services (*e.g.*, a new routing protocol) running on the top of NOS.

There are different proposals for deploying SDN like the Forwarding and Control Element Separation (ForCES) framework [61] and OpenFlow [62]. In OpenFlow, the Controller (*i.e.*, NOS), such as POX [63], Beacon [64], and Floodlight [65], handles network devices (*i.e.*, the datapath) by the OpenFlow protocol. Furthermore, the Controller supports deploying new and centralized Network Applications (*i.e.*, Network Services), such as groundbreaking applications to path selection and novel multicasting protocols.

5.3. Test environment

To evaluate the proposed approach, we conducted a proof-of-concept in a test environment (see Fig. 6). In such environment, the RDM System prototype was executed on a Web engine 7.0.26 and a Drools engine 6.1. RDMDB was executed on a MySQL 5.5. The RDM System and RDMDB were deployed on a machine with Linux Ubuntu O.S., 2.53 GHz Intel Core i5 processor, 4 GBytes RAM, and 250 GBytes hard disk. In turn, the applications used to evaluate the proposed approach were executed on a Test Client with 2 GBytes RAM and 2.53 GHz core 2 duo processor.

In the test environment, we managed three SDN-based networks controlled by Beacon 1.0.2, POX 1.0.0, and Floodlight 0.9. These controllers handled a lot of Open vSwitches 1.4; later in each evaluation, we will define the exact quantity of switches per network. Each OpenFlow controller was executed on a machine with 2.33 GHz Core 2 Duo processor, 2 GBytes RAM, and 160 GBytes hard disk. In turn, the Open vSwitches were executed on Mininet 2.2.1 and deployed on a server with 8 GBytes RAM and 3.4 GHz Core i7 processor. Mininet [58] is a software useful for emulating OpenFlow-based networks.

Fig. 7 presents the high-level operation of our test environment: (*i*) the RDM System gets the network information of SDN-based networks by the modules Sensing and SMRS, (*ii*) the above-mentioned modules return the network information to the RDM System, (*iii*) it retrieves the *nmsit* patterns from RDMDB, (*iv*) the RDM System conducts a match operation among the information and patterns retrieved, (*v*) if there is a match, it retrieves from RDMDB a composition template to address the recognized *nmsit*, (*vi*) it uses the retrieved template to generate the RDM that will address the detected *nmsit*; and (*vii*) it executes network management operations in the SDN-based networks when the network administrator launches the generated RDM by the Mashment Maker GUI.

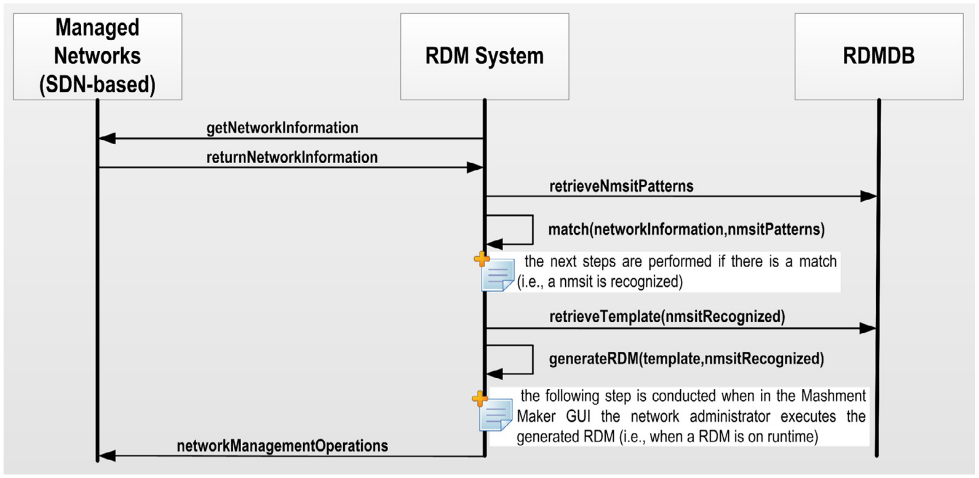

Managed Networks (SDN-based)
RDM System
RDMDB
getNetworkInformation
returnNetworkInformation
retrieveNmsitPatterns
match(networkInformation,nmsitPatterns)
the next steps are performed if there is a match (i.e., a nmsit is recognized)
retrieveTemplate(nmsitRecognized)
generateRDM(template,nmsitRecognized)
the following step is conducted when in the Mashment Maker GUI the network administrator executes the generated RDM (i.e., when a RDM is on runtime)
networkManagementOperations

Fig. 7. Test environment high-level operation.

5.4. Time-recognition

The time-recognition is the time that the RDM System takes for recognizing *nmsit* patterns. To measure time-recognition, we used two OpenFlow-based networks handled by POX and Floodlight (see Fig. 6). Each controller handled a datacenter network topology with 259 switches distributed in 4 levels of depth (*i.e.*, layers of access, aggregation, core, and edge) and 6 servers per rack. Thus, in total, we used 2 controllers, 518 switches, and 3626 ports. In this and the following evaluations involving average time in milliseconds (*ms*), we took 30 measurements with 95% confidence level.

Using the NMSit Designer, we defined a *nmsit* pattern to detect when any port of any switch handled by POX or Floodlight had more than 5% of dropped packets. Fig. 8 depicts such pattern encoded on JSON and DRL. It is important to highlight that, first, the RDM System performs the translation from JSON to DRL. Second, for network administrators and Mashment Creators such translation is always hidden.

In the time-recognition evaluation, first, we loaded only a *nmsit* pattern (*i.e.*, just a rule in the Drools Engine) and varied the number of generated *nmsits* from 250 to 1750 in each OpenFlow-based network. Thus, the total number of generated *nmsits* in each evaluation was from 500 to 3500. Second, we varied the number of loaded rules from 1 to 400 and once again the amount of generated *nmsits* from 500 to 3500.

Fig. 9 presents the time-recognition results. These results reveal that the RDM System is able to recognize *nmsit* patterns in a short time; in the worst behavior approximately 30.3 *ms* to identify 3500 *nmsits* having 400 loaded rules/patterns. Furthermore, the time-recognition is increased linearly in a negligible way with the growth of *nmsits* and loaded rules. Consequently, we can state that, in terms of time-recognition, it is feasible to use our approach to dealing effectively with *nmsits*.

5.5. Time-composition

The time-composition is the time that the RDM System takes for dynamically customizing composition templates and, so, generating RDMs. To measure time-recognition, we use three OpenFlow-based networks handled by POX, Floodlight, and Beacon (see Fig. 6). Each controller was in charge of handling a datacenter network topology with 259 switches distributed in 4 levels of depth and 6 servers per rack. Thus, in total, we used 3 controllers, 777 switches, and 4662 ports.

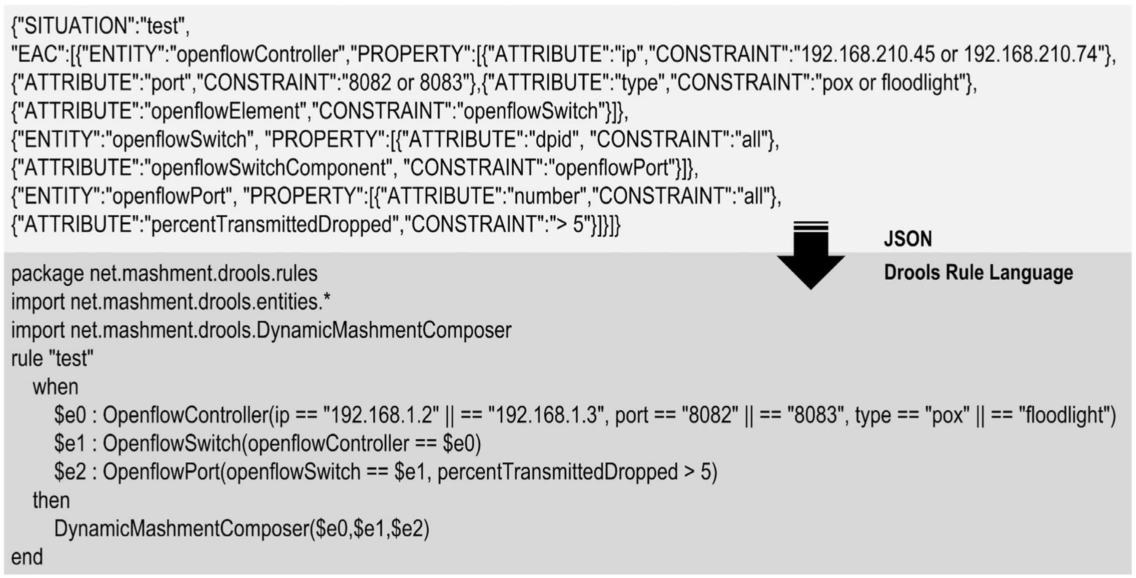
Using the Template Designer, we defined composition templates for monitoring when ports of switches handled by POX, Floodlight, or Beacon had more than 5% of dropped packages. Fig. 10 depicts a snippet of a composition template and the corresponding dynamic *mashment* generated by the RDM Maker.

In the time-composition evaluation, first, we varied the number of resources forming the composition templates from 2 to 8. It is noteworthy that more than 60% of mashups consist of $3-8$ components/resources [66]. Second, we modified the number of templates from 10 to 50; note that the number of templates defines the number of simultaneously generated RDMs.

Fig. 11 presents the time-composition results. These results reveal that the RDM System generates RDMs by customizing composition templates in a short time. In the worst behavior, approximately 14500 *ms* to dynamically compose 50 RDMs formed by 8 resources. Furthermore, the time-composition increases linearly with the growth of generated RDMs and resources per composition template. Considering, first, the above results. Second, the mechanism for generating RDMs has similar time-composition behavior than the composition proposals introduced on other application domains [67]. We can state that, in terms of time-composition, it is feasible to use the proposed approach to coping effectively with *nmsits*.

5.6. Time-consuming

The time-consuming (*i.e.*, T_{cons}) is the time that network administrators spend to cope with *nmsits*. In this evaluation, we raise a *nmsit* called NMSit-AS. Let's consider that a network administrator manages three Autonomous Systems, called AS_1, AS_2, and AS_3. AS_1, AS_2, and AS_3 are handled by Beacon, Floodlight, and POX, respectively. When there is



```
{"SITUATION":"test",
"EAC":[{"ENTITY":"openflowController","PROPERTY":[{"ATTRIBUTE":"ip","CONSTRAINT":"192.168.210.45 or 192.168.210.74"},
{"ATTRIBUTE":"port","CONSTRAINT":"8082 or 8083"},{"ATTRIBUTE":"type","CONSTRAINT":"pox or floodlight"},
{"ATTRIBUTE":"openflowElement","CONSTRAINT":"openflowSwitch"}]},
{"ENTITY":"openflowSwitch", "PROPERTY":[{"ATTRIBUTE":"dpid", "CONSTRAINT":"all"},
{"ATTRIBUTE":"openflowSwitchComponent", "CONSTRAINT":"openflowPort"}]},
{"ENTITY":"openflowPort", "PROPERTY":[{"ATTRIBUTE":"number","CONSTRAINT":"all"},
{"ATTRIBUTE":"percentTransmittedDropped","CONSTRAINT":"> 5"}]}]}
```

```
package net.mashment.drools.rules
import net.mashment.drools.entities.*
import net.mashment.drools.DynamicMashmentComposer
rule "test"
  when
    $e0 : OpenflowController(ip == "192.168.1.2" || == "192.168.1.3", port == "8082" || == "8083", type == "pox" || == "floodlight")
    $e1 : OpenflowSwitch(openflowController == $e0)
    $e2 : OpenflowPort(openflowSwitch == $e1, percentTransmittedDropped > 5)
  then
    DynamicMashmentComposer($e0,$e1,$e2)
end
```

Fig. 8. Test nmsit.

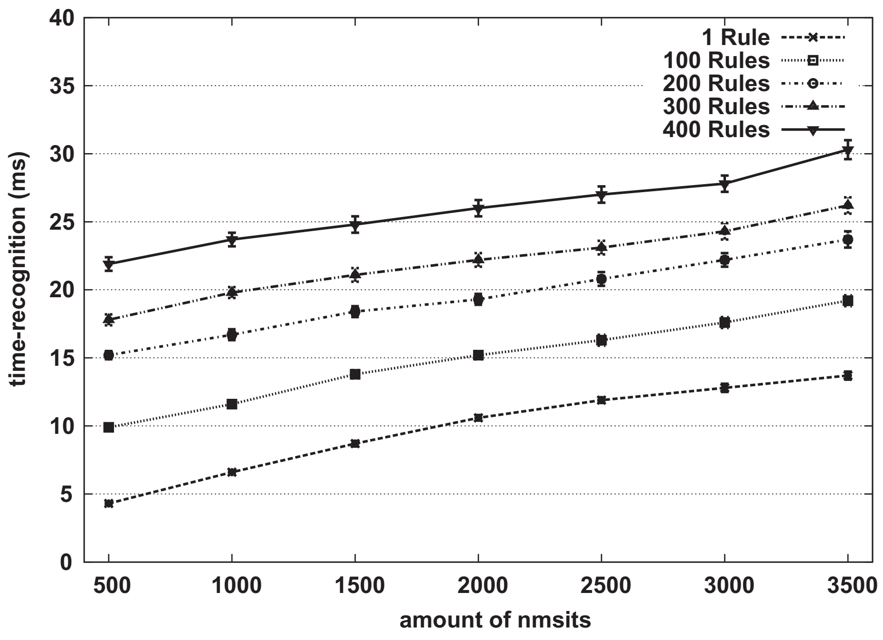
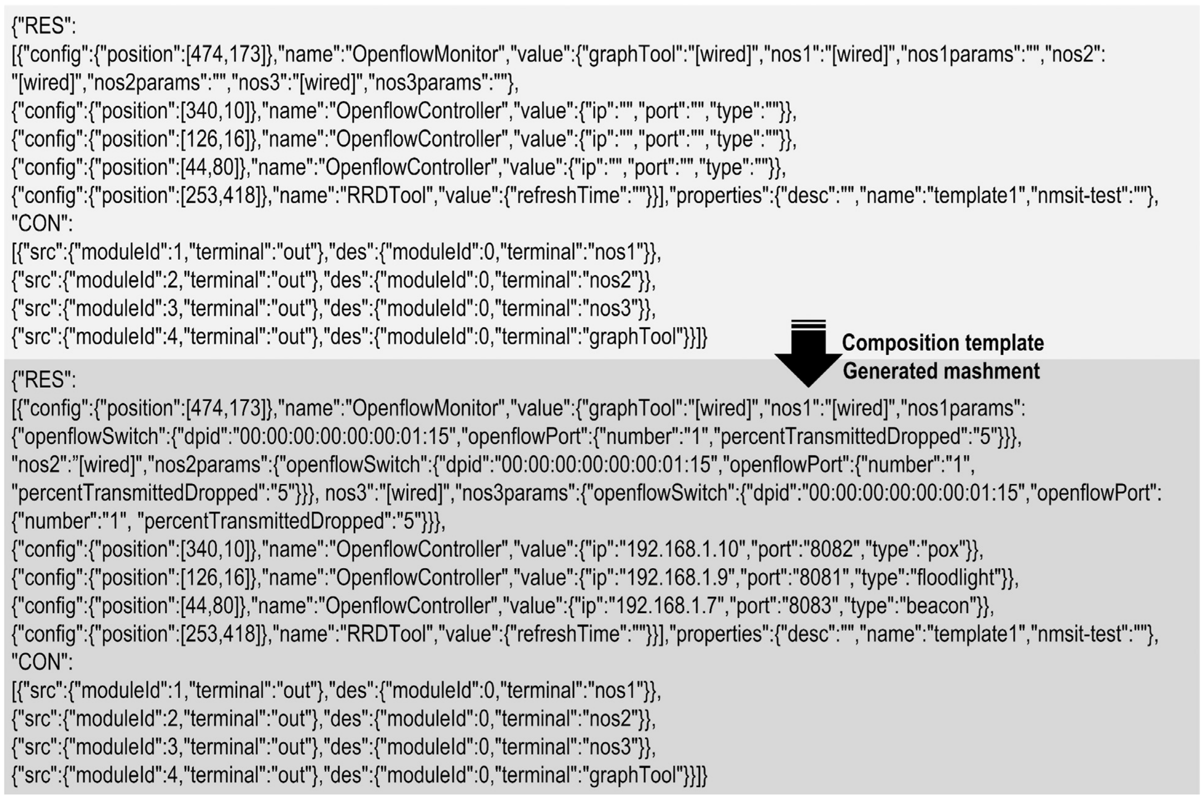


Fig. 9. Time-recognition behavior.

unexpected degradation in the performance of links that connect these *ASs*, he/she needs to identify in each *AS* which are the Open vSwitches that are causing such performance. In this way, he/she requires to rapidly and easily obtain a situational solution that presents, in an integrated, visual, and intelligible way, information about links and Open vSwitches.

To deal with NMSit-AS, the network administrator tests several options: (*i*) without RDM by the *Situational Script*, (*ii*) without RDM by the Performance Monitoring Mashment (*PMM*); and (*iii*) with the proposed approach by the RDM of Performance (*RDMP*). In general terms, the *Situational Script* is an application programmed and executed by the network administrator in a low-abstraction level. *PMM* is a composite solution developed and launched by the network administrator in the Mashment Maker [11] [26]. *RDMP* is a *mashment* automatically generated by the RDM System and offers the same functionalities as *PMM*.

To assess T_{cons}, we use the Keystroke–Level Model (KLM) [68] because it is useful to estimate the time that network



```
{"RES":
[{"config":{"position":[474,173]},"name":"OpenflowMonitor","value":{"graphTool":"[wired]","nos1":"[wired]","nos1params":"","nos2":
"[wired]","nos2params":"","nos3":"[wired]","nos3params":""},
{"config":{"position":[340,10]},"name":"OpenflowController","value":{"ip":"","port":"","type":""}},
{"config":{"position":[126,16]},"name":"OpenflowController","value":{"ip":"","port":"","type":""}},
{"config":{"position":[44,80]},"name":"OpenflowController","value":{"ip":"","port":"","type":""}},
{"config":{"position":[253,418]},"name":"RRDTool","value":{"refreshTime":""}}],"properties":{"desc":"","name":"template1","nmsit-test":""},
"CON":
[{"src":{"moduleId":1,"terminal":"out"},"des":{"moduleId":0,"terminal":"nos1"}},
{"src":{"moduleId":2,"terminal":"out"},"des":{"moduleId":0,"terminal":"nos2"}},
{"src":{"moduleId":3,"terminal":"out"},"des":{"moduleId":0,"terminal":"nos3"}},
{"src":{"moduleId":4,"terminal":"out"},"des":{"moduleId":0,"terminal":"graphTool"}}]}
```

```
{"RES":
[{"config":{"position":[474,173]},"name":"OpenflowMonitor","value":{"graphTool":"[wired]","nos1":"[wired]","nos1params":
{"openflowSwitch":{"dpid":"00:00:00:00:00:00:01:15","openflowPort":{"number":"1","percentTransmittedDropped":"5"}}},
"nos2":"[wired]","nos2params":{"openflowSwitch":{"dpid":"00:00:00:00:00:00:01:15","openflowPort":{"number":"1",
"percentTransmittedDropped":"5"}}}, nos3":"[wired]","nos3params":{"openflowSwitch":{"dpid":"00:00:00:00:00:00:01:15","openflowPort":
{"number":"1", "percentTransmittedDropped":"5"}}},
{"config":{"position":[340,10]},"name":"OpenflowController","value":{"ip":"192.168.1.10","port":"8082","type":"pox"}},
{"config":{"position":[126,16]},"name":"OpenflowController","value":{"ip":"192.168.1.9","port":"8081","type":"floodlight"}},
{"config":{"position":[44,80]},"name":"OpenflowController","value":{"ip":"192.168.1.7","port":"8083","type":"beacon"}},
{"config":{"position":[253,418]},"name":"RRDTool","value":{"refreshTime":""}}],"properties":{"desc":"","name":"template1","nmsit-test":""},
"CON":
[{"src":{"moduleId":1,"terminal":"out"},"des":{"moduleId":0,"terminal":"nos1"}},
{"src":{"moduleId":2,"terminal":"out"},"des":{"moduleId":0,"terminal":"nos2"}},
{"src":{"moduleId":3,"terminal":"out"},"des":{"moduleId":0,"terminal":"nos3"}},
{"src":{"moduleId":4,"terminal":"out"},"des":{"moduleId":0,"terminal":"graphTool"}}]}
```

Fig. 10. Snippet of template and generated mashment.

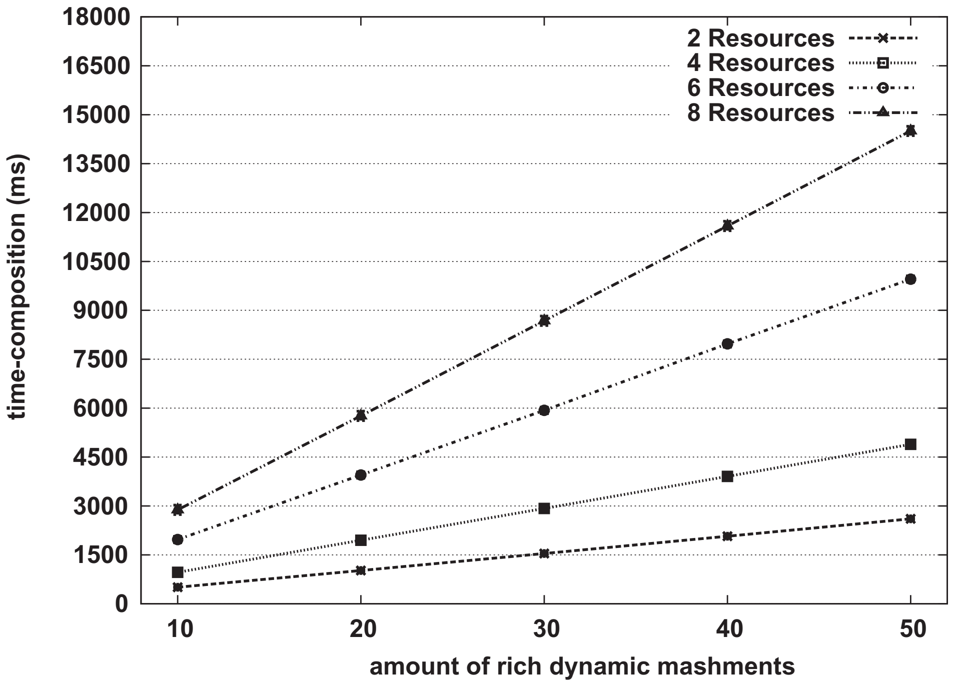
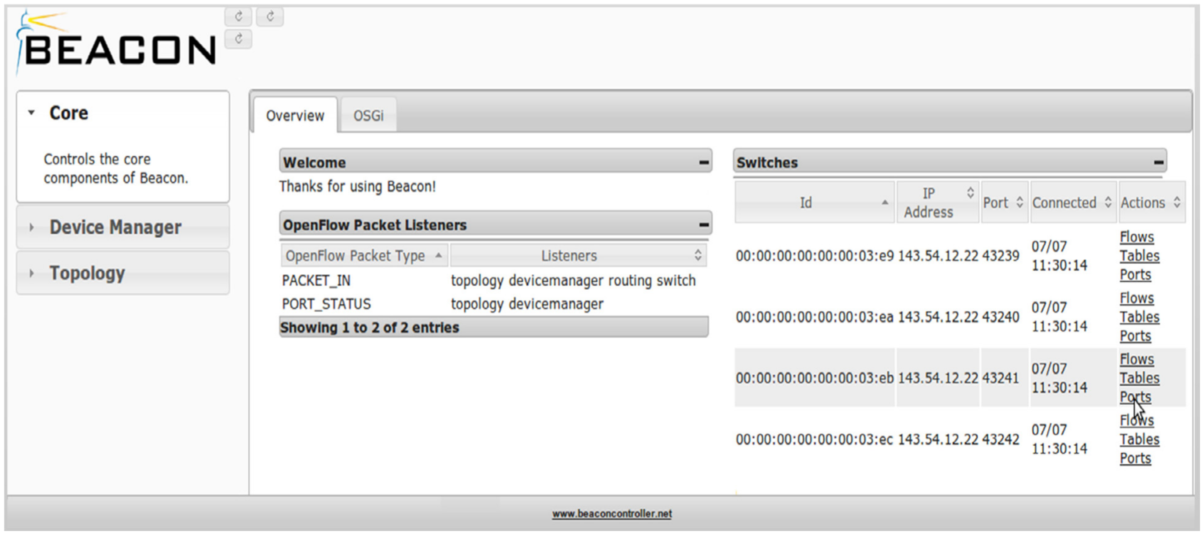


Fig. 11. Time-composition behavior.



Controls the core components of Beacon.

Thanks for using Beacon!

OpenFlow Packet Type	Listeners
PACKET_IN	topology devicemanager routing switch
PORT_STATUS	topology devicemanager

Showing 1 to 2 of 2 entries

Id	IP Address	Port	Connected	Actions
00:00:00:00:00:00:03:e9	143.54.12.22	43239	07/07 11:30:14	Flows Tables Ports
00:00:00:00:00:00:03:ea	143.54.12.22	43240	07/07 11:30:14	Flows Tables Ports
00:00:00:00:00:00:03:eb	143.54.12.22	43241	07/07 11:30:14	Flows Tables Ports
00:00:00:00:00:00:03:ec	143.54.12.22	43242	07/07 11:30:14	Flows Tables Ports

Fig. 12. Beacon Web Tool [64].

administrators spend to carry out tasks supported on computer keyboard and mouse. In KLM, each task is modeled as a sequence of actions. The original KLM-actions [68] and some helpful extensions [69] are: (i) press and release a key – $k = 0.2$ s, (ii) type a string – $nk * 0.2$ s, (iii) point the mouse – $p = 1.1$ s, (iv) hold or release the mouse – $b = 0.1$ s, (v) move the hand from mouse to keyboard – $h = 0.4$ s, (vi) drag-and-drop a visual element – $dnd = 1.3$ s; and (vii) wire two visual elements – $wire = 4.1$ s.

Addressing with Situational Script. $T_{cons:\,script}$ is the spent time by the network administrator for addressing NMSit-AS with the *Situational Script*. Without the mashment-based approach, for retrieving situational information, he/she uses a monitoring Web-based tool per controller, such as Beacon Web Tool [64], POX Web Tool [63], and Floodlight Web Tool [65]. As these tools operate similarly, $T_{cons:beacon} = T_{cons:pox} = T_{cons:\,floodlight}$.

The network administrator retrieves information in HTML tables about the packet traffic of a switch from the Beacon Web Tool [64], by the actions (see Fig. 12): (i) point the mouse to Core Components tab, (ii) press and release the mouse to select the Core Components tab, (iii) point the mouse to the Overview tab that presents a switches list, (iv) press and release the mouse to select the Overview tab, (v) point the mouse to select a switch, (vi) press and release the mouse to select a switch, (vii) point the mouse to the Ports link; and ($viii$) press and release the mouse to select Ports of switch. Considering these actions, $T_{cons:beacon} = h + 4p + 8b = 5.6$ s and the time of separately use the above mentioned tools is $T_{cons:nonIntegrated} = 16.8$ s.

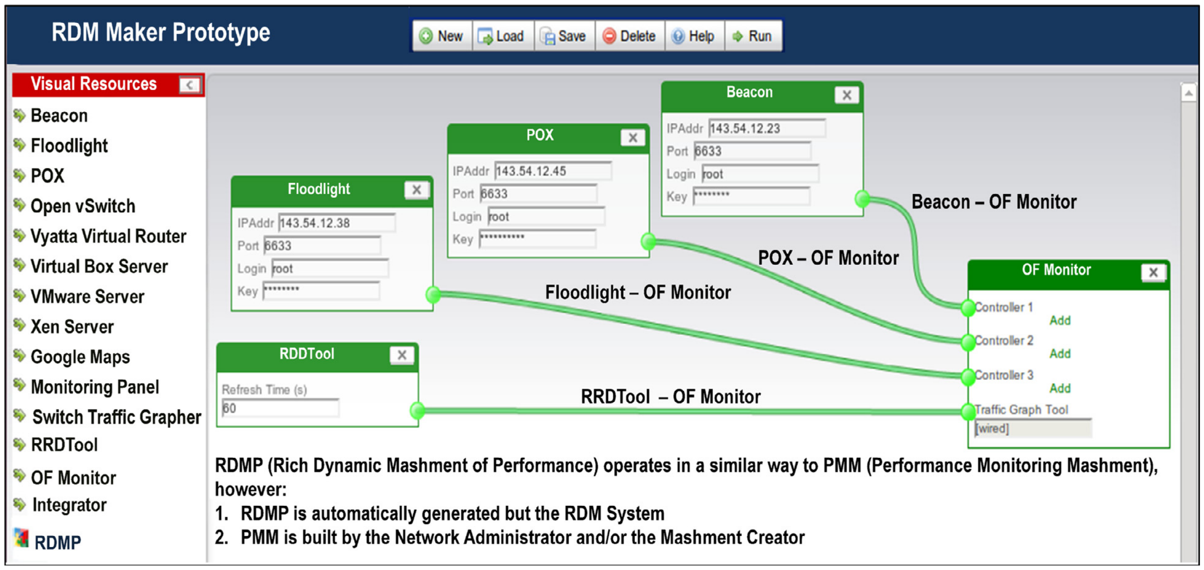
To obtain in a unique GUI, the retrieved information by the Web-based tools, the network administrator develops and launches the *Situational Script*. As this script generates HTML tables and RRDTool images, $T_{cons:dev} = T_{table} + T_{rrdImages}$. Considering only the time to type the code for generating the tables and images, $T_{cons:dev} = (h + 290k) + (h + 1200k) = 298.8$ s. In turn, $T_{cons:lau} = h + p + 2b + 11k = 3.9$ s. Thus, $T_{cons:script} = T_{cons:nonIntegrated} + T_{cons:dev} + T_{cons:lau} = 319.5$ s.

Addressing with PMM. $T_{cons:\,pmm}$ is the time spent by the network administrator for handling NMSit-AS with *PMM*. Here, $T_{cons:pmm} = T_{cons:dev} + T_{cons:lau} + T_{cons:use}$. As *PMM* is developed in the Mashment Maker [11] [26], $T_{cons:dev} = T_{sel} + T_{con} + T_{com}$.

To develop *PMM* (see Fig. 13), the network administrator initially selects Visual Resources by dragging-and-dropping *Beacon*, *POX*, *Floodlight*, *RRDTool*, and *OF Monitor*. Thus, $T_{sel} = 5 * dnd = 6.5$ s. Afterwards, he/she configures the selected resources by providing the functioning parameters of *Beacon*, *POX*, *Floodlight*, and *RRDTool*. As he/she manually writes these parameters, $T_{con:beacon} = T_{con:pox} = T_{con:\,floodlight} = [4 * (p + h + 2b) + (16 + 8 + 8 + 5) * k] = 14.2$ s and $T_{con:rrd} = p + h + 2b + 3k = 2.3$ s. Therefore, $T_{con} = 44.9$ s. Finally, he/she creates connections among the selected and configured resources by wiring *Beacon - OF Monitor*, *POX - OF Monitor*, *Floodlight - OF Monitor*, and *RRDTool - OF Monitor*. Thus, $T_{com} = 4 * wire = 16.4$ s.

Before requesting the execution of *PMM*, the network administrator saves it: (i) point the mouse in the Save button and click it, (ii) point the mouse in the dialog that asks for the *mashment* name and click it; and (iii) type the string “*PMM*”. Once *PMM* has been saved, he/she launches it by clicking the button Run. Thus, $T_{cons:lau} = 3(h + p + 2b) + 3k = 5.7$ s.

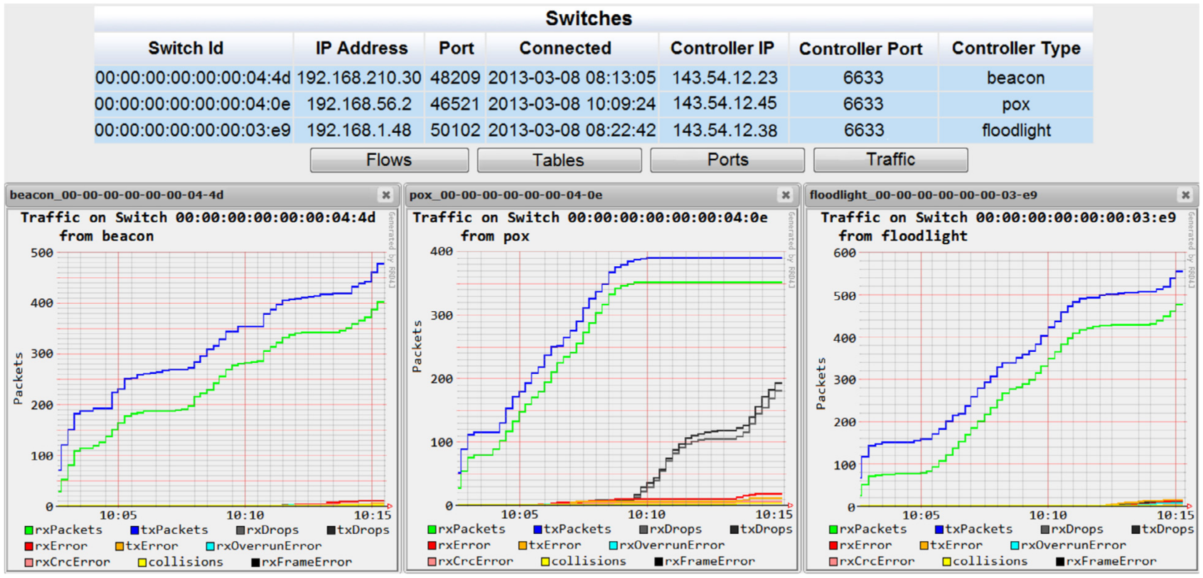
On runtime, *PMM* (see Fig. 14) allows the network administrator to retrieve information about NMSit-AS. He/she carries out in *PMM* the following actions to retrieve generic information of switches/links in three different controllers: (i) points the mouse to the Controllers list, (ii) presses and releases the mouse to select three distinct controllers, (iii) points the mouse to the button Switches/Links; and (iv) presses and releases the mouse to click the button Switches/Links. Furthermore, to retrieve information about flows, tables, ports, or traffic of three switches: (i) presses and releases the mouse to select three switches in



RDMP (Rich Dynamic Mashment of Performance) operates in a similar way to PMM (Performance Monitoring Mashment), however:

1. RDMP is automatically generated but the RDM System
2. PMM is built by the Network Administrator and/or the Mashment Creator

Fig. 13. Rich Dynamic Mashment of Performance.



Switches						
Switch Id	IP Address	Port	Connected	Controller IP	Controller Port	Controller Type
00:00:00:00:00:00:04:4d	192.168.210.30	48209	2013-03-08 08:13:05	143.54.12.23	6633	beacon
00:00:00:00:00:00:04:0e	192.168.56.2	46521	2013-03-08 10:09:24	143.54.12.45	6633	pox
00:00:00:00:00:00:03:e9	192.168.1.48	50102	2013-03-08 08:22:42	143.54.12.38	6633	floodlight

Flows
Tables
Ports
Traffic
beacon_00-00-00-00-00-00-04-4d
Traffic on Switch 00:00:00:00:00:00:04:4d from beacon
pox_00-00-00-00-00-00-04-0e
Traffic on Switch 00:00:00:00:00:00:04:0e from pox
floodlight_00-00-00-00-00-00-03-e9
Traffic on Switch 00:00:00:00:00:00:03:e9 from floodlight
Packets
10:05
10:10
10:15
🟩rxPackets ■txPackets ■rxDrops ■txDrops
■rxError ■txError ■rxOverrunError
■rxCrcError ■collisions ■rxFrameError

Fig. 14. RDMP and PMM on runtime.

different controllers, (*ii*) points the mouse to the button Flows, Tables, Ports, or Traffic; and (*iii*) presses and releases the mouse to click the button Flows, Tables, Ports, or Traffic. Thus, $T_{cons:use} = h + 3p + 16b = 5.3$ s.

Since we have already calculated $T_{cons:dev}$, $T_{cons:\,lau}$, and $T_{cons:\,use}$, we can calculate $T_{cons:\,pmm}$. As a result, it is expected that a network admistrator takes 78.8 s to deal with NMSit-AS by using *PMM*.

Addressing with RDMP. $T_{cons:\,rdmp}$ is the time spent by the network administrator for facing NMSit-AS with *RDMP*. As *RDMP* is generated by the RDM System (see Figs. 10 and 13), $T_{cons:rdmp} = T_{cons:lau} + T_{cons:use}$. As *RDMP* is a *mashment* dynamically composed, the network administrator launches it by clicking the button Run. Therefore, $T_{lau} = h + p + 2b = 1.7$ s. Furthermore, since on runtime *RDMP* and *PMM* provide identical functionalities and show the same GUI (see Fig. 14), $T_{cons:use} = 5.3s$. Thus, $T_{cons:rdmp} = 6s$.

Fig. 15 depicts the time-consuming results that reveal: (*i*) the time that the network administrator spends for developing *RDMP* is zero, attained by mechanisms for automatic recognition of *nmsits* and dynamic composition of *mashments*; and (*ii*) because every RDM is ready to be launched by

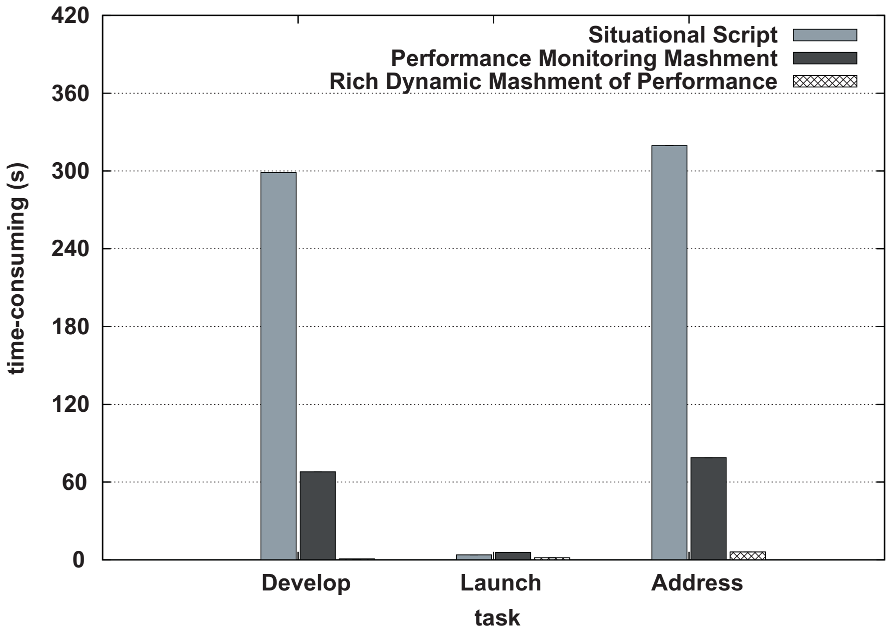


Fig. 15. Time-consuming behavior.

the RDM Designer, the time for launching *RDMP* ($T_{cons:lau} = 1.7$ s) is less than for *SituationalScript* ($T_{cons:lau} = 3.9$ s) and *PMM* ($T_{cons:lau} = 5.7$ s).

Summing up, the time for addressing NMSit-AS with RDM ($T_{cons:rdmp} = 6$ s) is less (about 92.3%–98.1%) than without our approach ($T_{cons:pmm} = 78.8$ s and $T_{cons:script} = 319.5$ s). This global result and the results per task demonstrate that, in terms of time-consuming, it is feasible to use our approach to handling effectively *nmsits*.

5.7. Time-response

To continue the evaluation, it is measured the time-response of *RDMP* and *Beacon Web Tool* when conducting the operations *SwitchesList* and *LinksList* over the networks of the test environment (see Fig. 6). These operations offer visual information of Open vSwitches and their links, which is useful to tackle the NMSit-AS.

In the time-response evaluation of *SwitchesList*, the number of Open vSwitches was varied from 20 to 100 per OpenFlow-based network. Thus, the total number of switches in each evaluation was 60, 120, 180, 240, and 300. Fig. 16 presents the corresponding results. Considering that the time-response (*r* in *ms*) of Web systems can be ranked as optimal ($r \leq 100$), good ($100 < r \leq 1000$), admissible ($1000 < r \leq 10000$), and deficient ($r > 10000$) [70], the time-response results reveal: (*i*) *SwitchesList* of *RDMP* has a good *r* that grows negligibly (less than 1 *ms* per switch) when the number of switches is increased in linear and tree topologies; and (*ii*) *r* is ranked as optimal for *Beacon Web Tool* and as good for *RDMP*; this result was expected because *Beacon Web Tool* operates with one type of controller and *RDMP* with three different types of controller.

In the time-response evaluation of *LinksList*, the number of links was varied from 50 to 250 per OpenFlow-based network. Therefore, the total number of links in each evaluation was 150, 300, 450, 600, and 750. Fig. 17 depicts the corresponding results that reveal: (*i*) *LinksList* of *RDMP* has a good *r* that grows negligibly (less than 1 *ms* per link) when the number of links is increased in linear and tree topologies; and (*ii*) *r* is ranked as optimal for *Beacon Web Tool* and as good for *RDMP*; again, this result was expected because *Beacon Web Tool* works with one type of controller and *RDMP* with three different types.

Although, at runtime, *RDMP* uses several software modules (*e.g.*, NMRS like BeaconService and Visual Resources like OF Monitor) to integrate and present monitoring information from different controllers, its behavior on time-response is good for the most of operations and regardless of controllers, topologies, and number of switches and links. Such behavior is because the Adaptation Layer hides the heterogeneity of controllers and, in turn, their centralized nature handles the number of network elements. Summing up, the time-response evaluation results demonstrate that, in terms of such metric, it is feasible to use the proposed approach to dealing with *nmsits* like the raised NMSit-AS.

5.8. Network traffic

To continue the evaluation, we measured the network traffic generated by *RDMP* and *Beacon Web Tool* when carrying out *SwitchesList* and *LinksList* in the networks of the test environment (see Fig. 6). In this and the following evaluations, the network traffic is expressed in *Bytes* or *KBytes*.

In the traffic evaluation of *SwitchesList*, the number of Open vSwitches was varied from 20 to 100 per OpenFlow-based network. Thus, the total number of switches in each

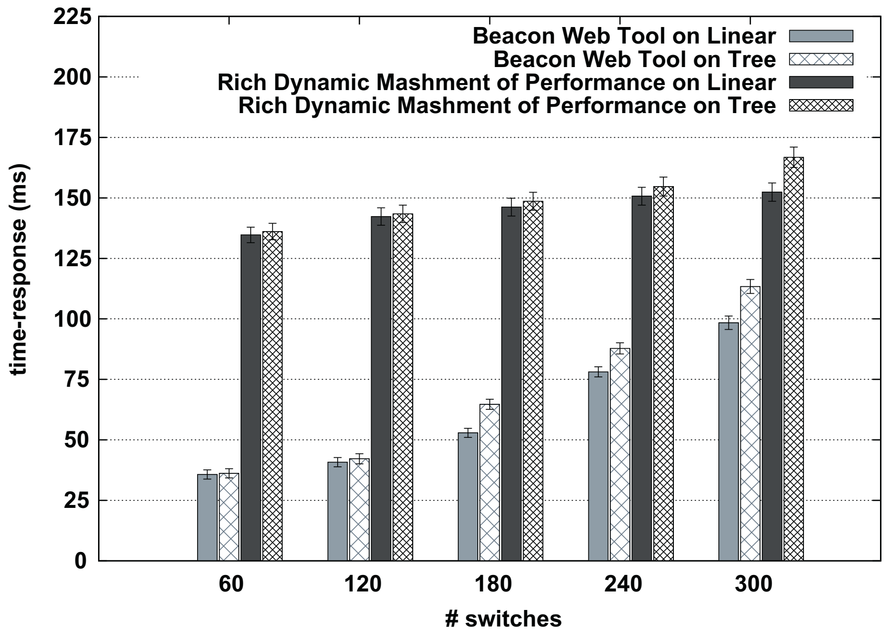


Fig. 16. Time-response on SwitchesList.

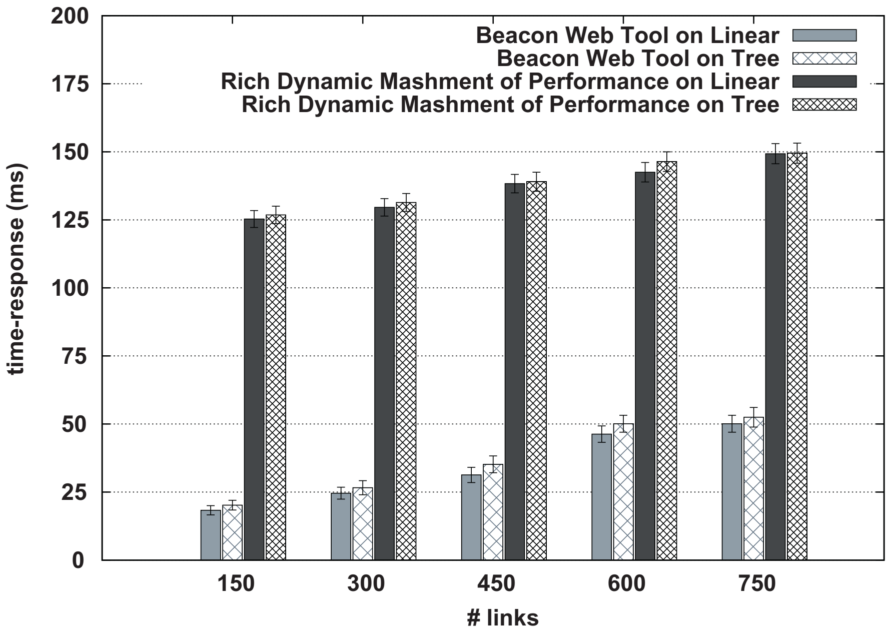


Fig. 17. Time-response on LinksList.

evaluation was 60, 120, 180, 240, and 300. Fig. 18 presents the corresponding results in which there is not discrimination by topology because, the traffic generated by *SwitchesList* of *RDMP* and *Beacon Web Tool* was independent of topologies (linear and tree) tested. In addition, these results reveal: (*i*) the traffic generated by *SwitchesList* of *RDMP* grows negligibly (approx 112 *Bytes* per switch) when the number of switches is increased, (*ii*) in relation to this operation, *RDMP* generates more traffic than *Beacon Web Tool*; and (*iii*) the additional traffic generated by *RDMP* is always less than 10%. Considering that *Beacon Web Tool* operates with just one type of controller and *RDMP* integrates data from three different types, the above facts corroborate that *SwitchesList* of *RDMP* has a good behavior on network traffic.

In the traffic evaluation of *LinksList*, the number of links was varied from 50 to 250 per OpenFlow-based network.

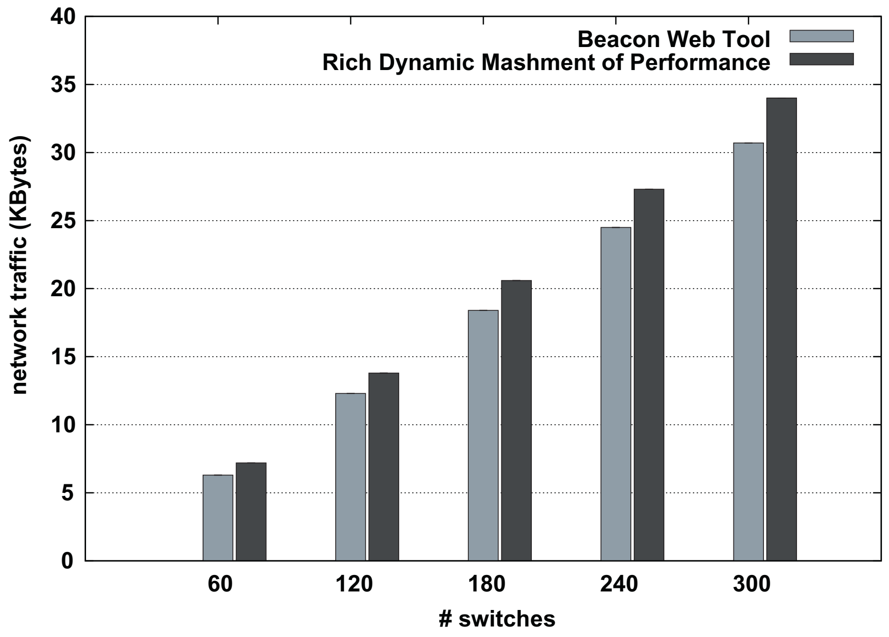


Fig. 18. Traffic on SwitchesList.

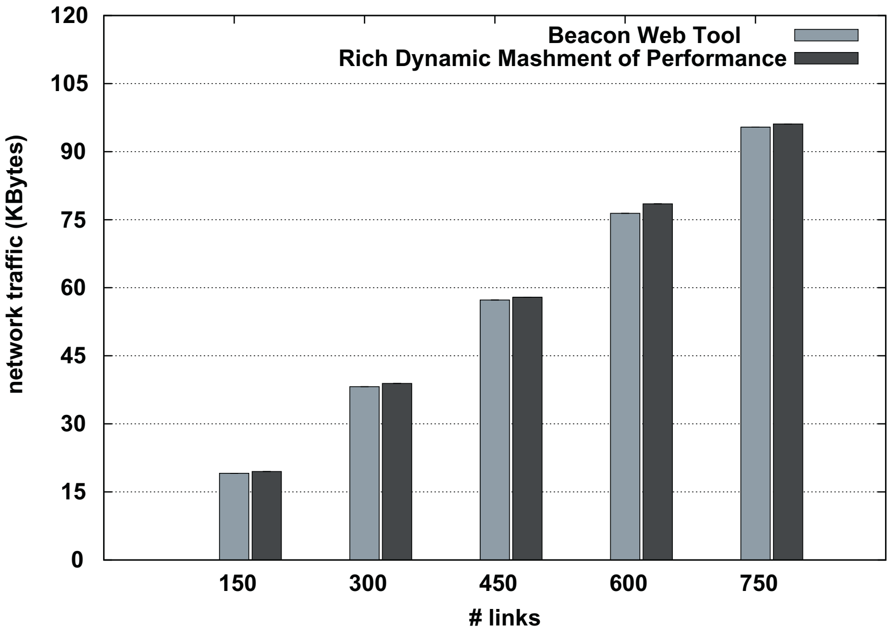


Fig. 19. Traffic on LinksList.

Therefore, the total number of links in each evaluation was 150, 300, 450, 600, and 750. Fig. 19 depicts the corresponding results in which there is not discrimination by topology because the traffic generated by *LinksList* of *RDMP* and *Beacon Web Tool* was independent of topologies tested. Furthermore, these results reveal: (*i*) the traffic generated by *LinksList* of *RDMP* grows negligibly (approx 129 *Bytes* per link) when the number of links is increased, (*ii*) regarding this operation, *RDMP* generates more traffic than *Beacon Web Tool*; and (*iii*) the additional traffic generated by *RDMP* is always less than 5%. Since the *Beacon Web Tool* operates with just one type of controller and *RDMP* integrates data from three different types, the above facts corroborate that *LinksList* of *RDMP* has a good behavior on network traffic.

Regarding the results of the network traffic evaluation of *RDMP*, it is important to mention: (*i*) JSON was used to decrease the size of information exchanged between the layers of Adaptation, Dynamic Composition, and Presentation because JSON is less verbose than XML; and (*ii*) the size of GUIs is too small to impact the quantity of traffic generated by *RDMP*. Furthermore, although *RDMP* integrates monitoring information from different controllers by using several additional software modules, its extra traffic is always less than 10% (worst operation - *SwitchesList*). Summing up, the above results corroborate that, in terms of network traffic, it is feasible to use our approach to coping with nmsits like the raised NMSit-AS.

5.9. Final remarks

The addressing with and without RDM of several *nmsits* was evaluated and analyzed in terms of: (*i*) time-recognition and time-composition related to mechanisms of automatic recognition of *nmsits* and dynamic composition of *mashments*, (*ii*) time-consuming related to the process followed by network administrators to face *nmsits*; and (*iii*) time-response and network traffic associated with the runtime behavior of solutions used to handle *nmsits*.

The evaluation results revealed several facts. First, the RDM Architecture allows recognizing automatically and composing dynamically rich dynamic*mashments* in a short time. Second, if network administrators cope with *nmsits* (*e.g.*, *NMSit-AS*) by developing and launching static *mashments* (*e.g.*, *PMM*), the time-consuming decreases. Third, such decreasing is greater when mechanisms to automatically recognize nmsits and dynamically compose *mashments* are used (*e.g.*, *RDMP*). Fourth, although an RDM uses extra software layers to face *nmsits*, such layers generate few additional time-response and network traffic in relation to Web-based network management tools (*e.g.*, Beacon Web Tool).

Summing up, the evaluation results demonstrate that, in terms of time-recognition, time-composition, time-consuming, time-response, and network traffic, it is feasible to use the proposed approach to deal with *nmsits* in an effective way. Therefore, such results confirm the relevance of the models of rich dynamic *mashments* and *nmsits*, the mechanisms to automatically recognize *nmsits* and dynamically compose *mashments*, the RDM Maker, and the RDM Architecture as a whole.

From a qualitative point of view, our approach provides mainly flexibility and extensibility. The flexibility refers to that RDM allows network administrators by themselves to customize and improve their workspace. They do not require a lot of Web programming skills to create situational management capabilities (*e.g.*, *PMM* and *RDMP*) because RDM provides a high-level abstraction (*i.e.*, mashable components) of network management technologies as well as of situational management operations.

The extensibility refers to that with our approach, network administrators can create (by conducting a simple process and using existing *mashments*) novel, advanced, and complex situational composite services targeted to overcome *nmsits*. It is possible because RDM leverages the composition, abstraction, and reusing models from mashups as well as allows implementing the investigative and control aspects of SM. Furthermore, in our approach the Mashment Creators can extend the RDM Maker by aggregating *mashments*, *nmsit* patterns, and composition templates. Such extension leads to improving the workspace of network administrators.

According to the evaluation results and the qualitative characteristics of RDM, it can be considered as a step forward in the network management, the SM discipline, and the mashup technology. In this regard, the network management is driven towards an environment focused on situations, composite situational solutions, and network administrators. The mashup foundations are brought up to SM to carrying out its investigative and control aspects. The mashup technology is led to a novel application domain located at the intersection of SM and network management.

6. Conclusions and future work

In this paper, we investigated the feasibility of using SM and mashups as an effective approach to facilitate the daily work of network administrators. The more significant contributions achieved by such investigation are: (*i*) the *nmsits* model that presented a way to characterize unexpected, dynamic, and heterogeneous situations in the network management domain by SM, (*ii*) the rich dynamic *mashments* model that introduced how to use mashups to conduct the investigative and control aspects of SM on network management, (*iii*) the mechanism to automatically recognize *nmsits* that presented how to detect such situations by rules and matching algorithms, (*iv*) the mechanism to dynamically compose *mashments* that introduced how to generate situational solutions by composition templates, and (*v*) the architecture that supported the proposed approach as a whole.

We also presented a prototype that carried out our approach and its evaluation in an SDN-based realistic scenario. In this scenario, we raised diverse *nmsits* and analyzed the feasibility of using rich dynamic *mashments* as an effective approach for network management. Considering the evaluation results, we can state about our approach: (*i*) it recognizes *nmsits* and composes rich dynamic *mashments* in a short time, (*ii*) it decreases the consumption of time of daily tasks performed by network administrators when facing *nmsits*, (*iii*) it has a good behavior in terms of time of response; and (*iv*) it has a good behavior in terms of network traffic because its additional entities and layers generate few extra traffic (less than 10%) in relation to the solutions currently used to cope with *nmsits*.

In the next research steps, we plan to extend and enhance the prototype to support other management tasks (*e.g.*, configuration and accounting) on traditional, SDN-based, and virtual networks. Finally, we also are interested in evaluating the productivity of network administrators that use rich dynamic *mashments*.

Acknowledgments

The CAPES (Brazil) and the University of Cauca (Colombia) supported the research of the professor Caicedo.

References

[1] G. Jakobson, L. Lewis, C. Matheus, M. Kokar, J. Buford, Overview of situation management at SIMA 2005, in: MILCOM, 3, IEEE, Atlantic City, USA, 2005, pp. 1630–1636.

[2] G. Jakobson, On modeling context in situation management, in: CogSIMA, IEEE, San Antonio, USA, 2014.1600–166
[3] G. Jakobson, On conceptualization of eventualities in situation management, in: CogSIMA, IEEE, San Diego, USA, 2013, pp. 75–82.
[4] S. George, W. Zhou, H. Chenji, M. Won, Y.O. Lee, A. Pazarloglou, R. Stoleru, P. Barooah, DistressNet: a Wireless ad hoc and Sensor Network Architecture for Situation Management in Disaster Response, IEEE Commun. Mag. 48 (3) (2010) 128–136.
[5] B. Magoutas, G. Mentzas, D. Apostolou, Proactive situation management in the future internet: the case of the smart power grid, in: DEXA, IEEE, Toulouse, France, 2011, pp. 267–271.
[6] D.M. Hein, R. Toegl, M. Pirker, E. Gatial, Z. Balogh, H. Brandl, L. Hluchý, Securing mobile agents for crisis management support, in: STC, ACM, New York, USA, 2012, pp. 85–90.
[7] R. Koelle, A. Tarter, Towards a distributed situation management capability for SESAR and NextGen, in: ICNS, IEEE, Herndon, USA, 2012 061–O6–12.
[8] I. Pereira, P. Costa, J. Almeida, A rule-based platform for situation management, in: CogSIMA, IEEE, San Diego, USA, 2013, pp. 83–90.
[9] H. Krohns-Valimaki, J. Stranden, J. Sarsama, Improving shared situation awareness in disturbance management, in: CIRED, IET, Stockholm, Sweden, 2013, pp. 1–4.
[10] R. Bruns, J. Dunkel, H. Billhardt, M. Lujak, S. Ossowski, Using complex event processing to support data fusion for ambulance coordination, in: FUSION, 2014, pp. 1–7.
[11] O. Caicedo Rendon, F. Estrada-Solano, L. Zambenedetti Granville, A mashup ecosystem for network management situations, in: GLOBECOM, IEEE, New York, USA, 2013, pp. 2249–2255.
[12] C. Cappiello, F. Daniel, M. Matera, C. Pautasso, Information quality in mashups, IEEE Internet Comput. 14 (4) (2010) 14–22.
[13] N. Laga, E. Bertin, R. Glitho, N. Crespi, Widgets and composition mechanism for service creation by ordinary users, IEEE Commun. Mag. 50 (3) (2012) 52–60.
[14] A. Majchrzak, P.H.B. More, Emergency! Web 2.0 to the rescue!, Commun. ACM 54 (2011) 125–132.
[15] H. Gebhardt, M. Gaedke, F. Daniel, S. Soi, F. Casati, C. Iglesias, S. Wilson, From mashups to telco mashups: a survey, IEEE Internet Comput. 16 (3) (2012) 70–76.
[16] V. Stirbu, Y. You, K. Roimela, V. Mattila, A lightweight platform for web mashups in immersive mirror worlds, IEEE Pervasive Comput. 12 (1) (2013) 34–41.
[17] C. dos Santos, R. Bezerra, J. Ceron, L. Granville, L. Rockenbach Tarouco, On using mashups for composing network management applications, IEEE Commun. Mag. 48 (12) (2010) 112–122.
[18] O.M.C. Rendon, C.R.P. dos Santos, A.S. Jacobs, L.Z. Granville, Monitoring virtual nodes using mashups, Comput. Netw. 64 (2014) 55–70.
[19] X. Chen, Y. Mao, Z.M. Mao, J. Van der Merwe, Declarative Configuration Management for Complex and Dynamic Networks, in: Co-NEXT, ACM, New York, USA, 2010, pp. 6:1–6:12.
[20] N. Kim, J. Kim, Building NetOpe Networking Services over OpenFlow-based Programmable Networks, in: ICOIN, 2011, pp. 525–529.
[21] D. Mattos, N. Fernandes, V. da Costa, L. Cardoso, M. Campista, L. Costa, O. Duarte, OMNI: OpenFlow MaNagement Infrastructure, in: NOF, IEEE, Paris, France, 2011, pp. 52–56.
[22] J. de Santanna, J. Wickboldt, L. Granville, A BPM-based Solution for Inter-domain Circuit Management, in: NOMS, 2012, pp. 385–392.
[23] C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker, Composing Software-defined Networks, in: NSDI, USENIX Association, Berkeley, CA, USA, 2013, pp. 1–14.
[24] H. Kim, N. Feamster, Improving Network Management with Software Defined Networking, IEEE Commun. Mag. 51 (2) (2013) 114–119.
[25] P. Smith, A. Schaeffer-Filho, D. Hutchison, A. Mauthe, Management patterns: SDN-enabled network resilience management, in: NOMS, 2014, pp. 1–9.
[26] O. Caicedo Rendon, F. Estrada Solano, L. Zambenedetti Granville, An Approach to Overcome the Complexity of Network Management Situations by Mashments, in: AINA, IEEE, Victoria, Canada, 2014, pp. 875–883.
[27] M. Badger, Zenoss Core Network and System Monitoring, Packt, Birmingham, UK, 2008.
[28] C. Chiang, G. Levin, S. Li, C. Serban, M. Wolberg, R. Chadha, G. Hadynski, L. LaBarre, Enabling Distributed Management for Dynamic Airborne Networks, in: POLICY 2009, IEEE, London, UK, 2009, pp. 102–105.
[29] W. Barth, Nagios: System and Network Monitoring, 2nd, No Starch Press, San Francisco, USA, 2008.
[30] G. Jakobson, J. Buford, L. Lewis, Situation Management: Basic Concepts and Approaches, in: Information Fusion and Geographic Information Systems, Lecture Notes in Geoinformation and Cartography, Springer Berlin Heidelberg, New Yor, USA, 2007, pp. 18–33.
[31] E.M. Maximilien, A. Ranabahu, S. Tai, Swashup: Situational Web Applications Mashups, in: OOPSLA, ACM, Montreal, Canada, 2007, pp. 797–798.
[32] M. Davies, P. Hamel, K. Yoshii, M. Goto, AutoMashUpper: Automatic Creation of Multi-Song Music Mashups, IEEE/ACM Trans. Audio, Speech, Language Proces. 22 (12) (2014) 1726–1737.
[33] C. dos Santos, R. Bezerra, J. Ceron, L. Granville, L. Tarouco, Botnet master detection using a mashup-based approach, in: CNSM, 2010, pp. 390–393.
[34] R. Bezerra, C. dos Santos, L. Bertholdo, L. Granville, L. Tarouco, On the Feasibility of Web 2.0 Technologies for Network Management: A Mashup-based Approach, in: NOMS, IEEE, Osaka, Japan, 2010, pp. 487–494.
[35] C. Pautasso, O. Zimmermann, F. Leymann, Restful Web Services vs. Big Web Services: Making the Right Architectural Decision, in: Proceedings of International Conference on World Wide Web, ACM, New York, USA, 2008, pp. 805–814.
[36] M.L. Massie, B.N. Chun, D.E. Culler, The ganglia distributed monitoring system: design, implementation and experience, Paral. Comput. 30 (2003) 2004.
[37] J. Wang, L. Yang, M. Yu, S. Wang, Application of server virtualization technology based on CitriX XenServer in the information Center of the Public Security Bureau and Fire Service Department, in: ISCCS, IEEE, Kota Kinabalu, Malaysia, 2011, pp. 200–202.
[38] T. Oetiker, MRTG: the multi router traffic grapher, in: LISA, USENIX, 1998, pp. 141–148.
[39] T. Oetiker, Monitoring your it gear: the MRTG story, IT Professional 3 (6) (2001) 44–48.
[40] S. Lin, Z. Gao, K. Xu, Web 2.0 Traffic Measurement: Analysis on Online Map Applications, in: Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video, ACM, New York, USA, 2009, pp. 7–12.
[41] T.L. Ruthkoski, Google Visualization API Essentials, Packt, Birmingham, UK, 2013.
[42] E. Salvador, L. Granville, Using Visualization Techniques for SNMP Traffic Analysis, in: ISCC, IEEE, Marrakech, Morocco, 2008, pp. 806–811.
[43] I. Juniper Networks, Juniper Home, 2014, [Accessed November 2015].
[44] S.I. ULC, SandvineHome, 2014, [Accessed November 2015].
[45] E.F. Hill, Jess in Action: Java Rule-Based Systems, Manning Publications Co., Greenwich, USA, 2003.
[46] P. Browne, JBoss Drools Business Rules, Packt, Birmingham, UK, 2009.
[47] C. Ibsen, J. Anstey, Camel in Action, 1st, Manning Publications Co., Greenwich, CT, USA, 2010.
[48] R.T. Fielding, R.N. Taylor, Principled Design of the Modern Web Architecture, ACM Trans. Internet Technol. 2 (2) (2002) 115–150.
[49] D. Grigori, J. Corrales, M. Bouzeghoub, A. Gater, Ranking BPEL Processes for Service Discovery, IEEE Trans. Services Comput. 3 (3) (2010) 178–192.
[50] R.C. Angell, G.E. Freund, P. Willett, Automatic Spelling Correction Using a Trigram Similarity Measure, Infor. Proces. Manag. 19 (4) (1983) 255–261.
[51] G.A. Miller, WordNet: a Lexical Database for English, Commun. ACM 38 (11) (1995) 39–41.
[52] A.A. Patil, S.A. Oundhakar, A.P. Sheth, K. Verma, Meteors Web Service Annotation Framework, in: Proceedings of International Conference on World Wide Web, ACM, New York, USA, 2004, pp. 553–562.
[53] M. Bostock, V. Ogievetsky, J. Heer, D^3 data-driven documents, IEEE Trans,. Visual. Comput. Graphics 17 (12) (2011) 2301–2309.
[54] P. Clements, R. Kazman, M. Klein, Evaluating Software Architectures: Methods and Case Studies, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
[55] P. Shanmugapriya, R.M. Suresh, Article: software architecture evaluation methods - a survey, Int. J. Comput. Appl. 49 (16) (2012) 19–26.
[56] R.T. Fielding, R.N. Taylor, Principled design of the modern web architecture, ACM Trans. Internet Technol. 2 (2) (2002) 115–150.
[57] G. Natasha, K. Teemu, P. Justin, P. Ben, C. Martin, M. Nick, S. Scott, NOX: towards an operating system for networks, Comput. Commun. Rev. ACM 38 (3) (2008) 105–110.
[58] B. Lantz, B. Heller, N. McKeown, A network in a laptop: rapid prototyping for software-defined networks, in: Proceedings of ACM SIGCOMM Workshop on Hot Topics in Networks, ACM, New York, USA, 2010, pp. 19:1–19:6.
[59] A. Lara, A. Kolasani, B. Ramamurthy, Network innovation using openflow: a survey, IEEE Commun. Surveys Tutorials PP (99) (2013) 1–20.

[60] O.M.C. Rendon, F. Estrada-Solano, L.Z. Granville, A mashup-based approach for virtual SDN management, in: Proceedings of Annual International Computer, Software & Applications Conference (COMPSAC), 2013, pp. 143–152.
[61] A. Doria, J. Hadi, R. Salim, H. Haas, W. Khosravi, W. Wang, L. Dong, J. Gopal, J. Halpern, Forwarding and control element separation (ForCES) protocol specification, 1 (2010) 5–29. (RFC 5810)
[62] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, Comput. Commun. Rev. ACM 38 (2) (2008) 69–74.
[63] POX, POX Home, 2014, [Accessed November 2015].
[64] D. Erickson, Beacon Home, 2013, [Accessed November 2015].
[65] FloodLight, Floodlight Home, 2014, [Accessed November 2015].
[66] G. Huang, Y. Ma, X. Liu, Y. Luo, X. Lu, B. Blake, Assisting navigation and complementary composition of complex service mashups, IEEE Trans. Services Comput. PP (99) (2014).1–1
[67] A. Ordonez, V. Alcazar, J.C. Corrales, P. Falcarin, Automated context aware composition of advanced telecom services for environmental early warnings, Expert Syst. Appl. 41 (13) (2014) 5907–5916.
[68] D. Kieras, Using the Keystroke-Level model to estimate execution times, in: University of Michigan, 2001.
[69] S. Tian, G. Weber, C. Lutteroth, A tuplespace event model for mashups, in: OzCHI, ACM, New York, USA, 2011, pp. 281–290.
[70] S. Joines, R. Willenborg, K. Hygh, Performance Aanalysis for Java Websites, Addison–Wesley Longman Publishing Co., Inc., Boston, USA, 2002.

Oscar Mauricio Caicedo Rendon is full professor at the Telematics Department of the University of Cauca, Colombia. He received his Ph.D. degree in Computer Science from the Institute of Informatics (INF) of the Federal University of Rio Grande do Sul (UFRGS), Brazil, in 2015. He also received a Master degree in Telematics (2006) and a degree in Electronics and Telecommunications Engineering (2001) from the University of Cauca (UNICAUCA). His research interests include network management, Software-Defined Networking, and Network Function Virtualization.

Felipe Estrada-Solano is a Master student in Telematics at the University of Cauca (UNICAUCA). He received his degree in Electronics and Telecommunications Engineering (2010) from UNICAUCA. His topics of interest include network and service management, Web 2.0/3.0 technologies, and Software-Defined Networking.

Vinicius Guimarães is professor at the Sul-Rio-Grandense Federal Institute of Education, Science, and Technology (IFSul), Brazil, and a Ph.D. student in Computer Science at the Institute of Informatics of the Federal University of Rio Grande do Sul (UFRGS), Brazil. He received his B.Sc. (2005) degree in Computer Science from Catholic University of Pelotas (UCPEL), Brazil and his M.Sc degree from Pontifical Catholic University of Rio Grande do Sul (PUCRS), Brazil. His research interests include network and service management, Software-Defined Networking, and information visualization.

Liane Margarida Rockenbach Tarouco is full professor at the Center for New Technologies on Education of the Federal University of Rio Grande do Sul (UFRGS), Brazil. She received his M.Sc. degree in computer science, from UFRGS in 1976 and Ph.D. degree from USP in 1990. She was member of IFIP WG 6.5 (Messaging Systems) and WG 6.6 (Network Management) and has served as a TPC member for many important events in the area of computer networks.

Lisandro Zambenedetti Granville received the M.Sc. and Ph.D. degrees in Computer Science from the Institute of Informatics (INF) of the Federal University of Rio Grande do Sul (UFRGS), Brazil, in 1998 and 2001, respectively. Currently, he is a professor at INF-UFRGS. Lisandro is co-chair of the Network Management Research Group (NMRG) of the Internet Research Task Force (IRTF) and vice-chair of the Committee on Network Operations and Management (CNOM) of the IEEE Communications Society (COMSOC). He was also technical program committee co-chair of the 12th IEEE/IFIP Network Operations and Management Symposium (NOMS 2010) and 18th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2007). His research interests include network and services management, Software-Defined Networking, network virtualization, information visualization, and network programmability.

An approach to Overcome the Complexity of Network Management Situations by Mashments

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 work performed by network administrators to address sudden, dynamic, heterogeneous, and time specific situations that happen in the network management domain is complex. In this paper, we introduce an approach that allows network administrators to overcome the complexity of handling these network management situations (called NMSits). The approach is made up of Mashments that are special mashups used to cope with NMSits, the process to develop and execute Mashments, and the Mashment Maker that supports such model and process. We use IT Service Management metrics to evaluate our approach, measuring the complexity of facing, with and without the Maker, a specific NMSit that occurs in several networks based on the Software Defined Networking paradigm. The evaluation results demonstrate that the complexity decreases when network administrators use our approach to handle NMSits.**

***Keywords*-Complexity; Mashment; Mashup; NMSit, Situation Management; Web-based Network Management;**

I. Introduction

The Situation Management (SM) discipline provides solutions that enable analyzing, correlating, and coordinating interactions among people, information, technologies, and actions intended to overcome situations happening or that might happen in dynamic systems [1] [2]. SM foundations are [3]: (*i*) a *Situation* that is modeled as a collection of entities in a domain, their attributes, and relationships in a time interval, (*ii*) the investigative aspect related to retrospective cause analysis of *Situations*, (*iii*) the control aspect devised to change or preserve *Situations*; and (*iv*) the predictive aspect aimed to predict *Situations*.

SM has been used in domains such as disaster response [4], smart power grid networks [5], security crisis management [6], and public health [7]. However, to the best of our knowledge, there is no SM-based approach to address the complexity of sudden, dynamic, heterogeneous, and time specific *Situations* that network administrators face in their daily work. An example of network management *Situation* is the unexpected failures in the packet transmission of virtual routers belonging to a slice made up of SDN (Software Defined Networking) networks handled by different NOS (Network Operating Systems). Hereinafter, we will refer to this type of *Situations* in the network management domain as NMSits [8].

We argue that the work conducted by network administrators to address NMSits is complex. This complexity exists because, first, although large research efforts have been made to deal with the intricacy of network management [9] [10] [11] [12] [13], they do not focus on handling such a complexity when *Situations* arise unexpectedly. Thus, they have a constrained response capacity to meet NMSits. Second, to face sudden *Situations*, network administrators must handle and rely on a vast amount of non-integrated tools (*e.g.*, traceroute, ZenOSS, OpenNMS, and so on), which hinders their work. Third, to cope with situational requirements, network administrators are usually forced to develop low-level scripts; developing these scripts itself is also complex because network administrators may not be experienced programmers.

Mashups are Web applications built up by end-users through the combination of Web resources available along the Internet [14] [15]. The mashup technology has been employed to manage *Situations* in many domains, such as project management [16], telco services [17], immersive mirror worlds [18], and data integration [19]. In our previous work, we analyzed mashups as a feasible mechanism to accomplish specific tasks for network management in both traditional [20] and SDN-based networks [21]. Nevertheless, we have not addressed how to overcome the complexity of tasks fulfilled by network administrators dealing with NMSits. We refer to mashups used to cope with one or more NMSits as Mashments [8].

In this paper, we take a step further, proposing a novel Mashment-based approach that assists network administrators to overcome the complexity of NMSits and, consequently, facilitates their work. The key contributions from this paper are: (*i*) a conceptual model that presents how to address NMSits by Mashments, (*ii*) a process to develop and execute Mashments, targeted to surpass the intrincacy of tasks carried out by network administrators to handle NMSits, (*iii*) a Mashment Maker prototype that supports the model and process abovementioned; and (*iv*) a Mashment that faces a NMSit on SDN, demonstrating the decrease of complexity when network administrators use our approach to deal with NMSits.

The remainder of this paper is organized as follows. In Section II, we review both the background and the related work. In Section III, we introduce the Mashment-based approach. In Section IV, we describe and discuss the case study raised to evaluate our approach. In Section V, we provide conclusions and implications for future work.

II. Background and Related Work

In this section, we describe research concerning SM, mashups, and mashups on network management. We also present related work with handling the complexity of network management.

A. Situation Management

The goal of SM is to provide solutions aimed to investigate, control, and predict *Situations* that are composite entities whose components are other entities, their attributes, and relationships in a time interval [1] [2]. To accomplish such a goal, SM-based solutions offer a global vision of *Situations* by collecting, correlating, and merging information from multi-entities, seeking to maximize the user comprehension and, so, supporting the opportune and correct decision making.

SM has been used in diverse domains. In the disaster response [4], a situation-aware architecture and a set of protocols support the timely delivery of high volumes of accurate data that the disaster responders need to make correct decisions. In the smart grid power networks [5], an architecture, based on semantics, linked open data, and complex event processing, enables to respond intelligently to the active power demand of end-users. In the security crisis management [6], a mobile agents platform allows to provide timely information to the on-site personnel, the tactical crisis command, and the off-site strategic command centre. In the public health [7], a platform permits the development of situation-aware applications targeted to monitor suspicious cases of tuberculosis. To the best of our knowledge, up to now, SM has been not used to handle the complexity of *Situations* in the network management domain.

B. Mashups

Mashups are Web applications formed by combining Web resources available on the Internet [14] [15]. This combination is mainly achieved by a simple composition model, which allows end-users the development of customized applications, in an easy and rapid way [22]. Furthermore, mashups encourage the reuse by permitting end-users to generate more advanced applications through extending the existing compositions.

The mashup technology has been employed to manage *Situations* in several domains. In the project management [16], a mashup system allows managers to easily compose small solutions for displaying and filtering information about their projects. In the telco services [17], a reference architecture was defined to facilitate the provisioning of telco-mashups for end-users; a telco-mashup is a composite service that combines functionalities from telecom networks like streaming, quality of service, and billing. In the immersive mirror worlds [18], the Cloud City Scene platform enables end-users to create, in a mashup manner, realistic and immersive street-level representations of the physical world. In the data integration [19], Mashroom, a spreadsheet-like programming environment allows non-developers to create composite services, by aggregating data sources on the fly and in an interactive way.

In our previous work, we analyzed the mashup technology as a feasible mechanism to carry out specific tasks in the network management domain. Initially, we used mashups to accomplish the botnet detection [23] and the traffic monitoring of the border gateway protocol among two autonomous systems [24]. Afterwards, we introduced a generic architecture to support the composing of network management applications [20]. Recently, we leveraged the features of the mashup technology to conduct the integrated monitoring of SDN-based networks [21] and identified a mashup ecosystem around NMSits [8]. However, we have not addressed how to overcome the complexity of coping with NMSits.

C. Complexity of Network Management

There is a lot of research works about addressing the complexity of network management. COOLAID [9] automates network configuration by queries performed on an abstract database containing network information. NetOpen [10] allows to build up SOA services for monitoring and configuring OpenFlow networks by networking primitives. MEICAN [11] uses the business process management for permitting network administrators take part in the decision-making process of provisioning virtual inter-domain circuits. Pyretic [12] enables network programmers to build SDN applications using an abstract packet model, parallel and sequential composition operators, and topology abstraction. Procera [13] permits to manage SDN networks by expressing event-driven and reactive policies based on control domains. In the aforecited works, the network administrator is responsible for manually writing policies, queries, rules, or primitives in specific languages and/or controllers. Thus, his/her daily work to overcome sudden *Situations* of network management remains complex.

Unlike the above works, we consider concepts from SM and mashups, to propose an approach (section III) that acts in a more high-abstraction level and focuses on decrease the complexity of tasks fulfilled by network administrators when facing NMSits. Furthermore, as opposed to the aforementioned works that have been evaluated using metrics, such as bandwidth, response time, and code lines, we concern about the complexity perceived by network administrators (section IV).

III. Mashments Complexity & NMSits

To better explain our approach, at start, we present a conceptual model about how to address NMSits by Mashments. Afterwards, we introduce the process and complexity to develop and execute Mashments. At last, we present the Mashment Maker.

A. Addressing NMSits by Mashments

A NMSit is a sudden, dynamic, heterogeneous, and time specific *Situation* happening or that might happen in the network management domain. Examples of NMSits, that may be faced by network administrators in their daily work, are: in Faults, to find the cause root of unexpected and multiple packet transmission failures in network slices formed by several OpenFlow networks. In Performance, (*i*) to control the abrupt performance degradation of one or more nodes (switches, routers, and so on) on networks that use diverse virtualization environments; and (*ii*) to monitor, in a near real time way, sudden violations in service level agreements. NMSits as the aforementioned may be addressed by using mismatched tools but it overloads and becomes complex the work of network administrators. Also, network administrators may develop home-brew situational scripts. However, such a development is daunting and complex for non-programmers.

The Figure 1 depicts the conceptual model for addressing NMSits by Mashments. If a NMSit happens, the network administrator: (*i*) orchestrates a Mashment (*i.e.*, combines services, processes, user interfaces, and mashup operations to define a plan and deal with such a NMSit), or (*ii*) reuses a Mashment (*i.e.*, takes advantage of existing plans to face the NMSit). Then, he/she executes the Mashment; on runtime, it performs Network Management Operations targeted to investigate/resolve the NMSit. These Operations are internally conducted via Network Situational Processes, Situation Management Resource as a Service (SMRS), Mediating, and Situation Management Resources (SMR).

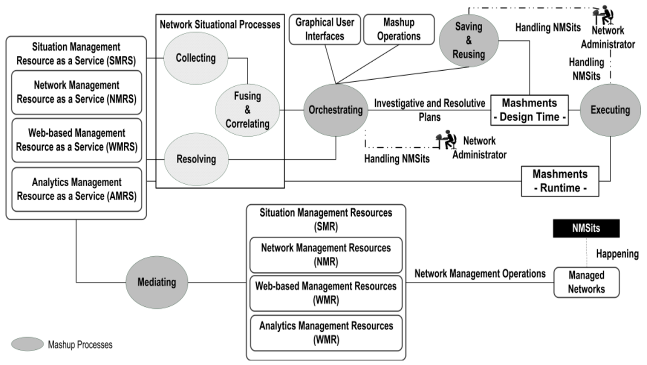

Situation Management Resource as a Service (SMRS)
Network Management Resource as a Service (NMRS)
Web-based Management Resource as a Service (WMRS)
Analytics Management Resource as a Service (AMRS)
Network Situational Processes
Collecting
Fusing & Correlating
Resolving
Graphical User Interfaces
Mashup Operations
Saving & Reusing
Handling NMSits
Network Administrator
Handling NMSits
Orchestrating
Investigative and Resolutive Plans
Mashments - Design Time -
Executing
Handling NMSits
Network Administrator
Mashments - Runtime -
Situation Management Resources (SMR)
Network Management Resources (NMR)
Web-based Management Resources (WMR)
Analytics Management Resources (WMR)
NMSits
Happening
Mediating
Network Management Operations
Managed Networks
Mashup Processes

Figure 1: Addressing NMSits by Mashments: Conceptual Model

A SMR is any solution that provides access and communication to and from network elements or entire networks involved in NMSits. There are three types of SMR: Network Management Resources (NMR) that are solutions, like ZenOSS, Citrix Center, OpenFlow MaNagement Infraestructure (OMNI), and Nagios, intended to conduct network management operations. Web-based Network Management Resources (WMR) that are tools available in the Internet, such as the Multi Router Traffic Grapher (MRTG) and RRDTool, useful to perform network management tasks. Analytics Management Resources (AMR) that are solutions, like Junos Network Analytics Suite, Management Traffic Analyzer, and Sandvine Network Analytics, suitable to analyze network management information.

A SMRS is a software entity that offers the network management operations of a SMR to the Network Situational Processes, aiming to hide the complexity of NMR, WMR, and AMR. Specifically, a Network Management Resource as a Service (NMRS) is responsible for providing functionalities of NMR, a Web-based Management Resource as a Service (WMRS) is in charge of offering capabilities of WMR, and an Analytics Management Resource as a Service (AMRS) is responsible for supplying functionalities of AMR.

The Network Situational Processes help to automate and carry out the investigative/control aspects of SM by using NMRS, WMRS, and/or AMRS. There are three Network Situational Processes: Collecting, Fusing&Correlating, and Resolving. Collecting allows to retrieve information about NMSits through SMRS. Fusing&Correlating permits to merge and correlate the information retrieved by Collecting. Fusing&Correlating and Collecting aid in the creation of investigative plans that are useful to determine the cause of NMSits. Resolving enables to perform, by using SMRS, network management operations aimed to control (change/preserve) NMSits. Consequently, Resolving supports the building up of resolutive plans.

The Mashup Processes provide the automation needed to orchestrate, save&reuse, and execute Mashments that are composite and customisable situational solutions which allow network administrators to deal with NMSits. In particular, Orchestrating includes selecting, configuring, and connecting the resources that form a Mashment: SMRS, Network Situational Processes, GUI, Mashup Operations, and even Mashments.

The GUI are internal and external libraries helpful to generate advanced and integrated Mashment interfaces targeted to network administrators. An external GUI is an Application Program Interface (API), such as Yahoo Maps and Google Chart, provided by a third-party and that may be used to illustrate composed information about networks and their devices. An internal GUI is, for instance, a specific user interface developed to show, in a correlated way, network traffic information.

The Mashup Operations are: (*i*) control patterns (*e.g.*, sequential, parallel, and conditional) that allow to define the process flow of Mashments and, consequently, investigative and resolutive plans, (*ii*) structures for configuring and invoking the resources that form Mashments; and (*iii*) structures for receiving, sorting, and filtering information from any SMRS.

Executing enables network administrators to run Mashments. It is noteworthy that, on run time, all Mashments delegates their management operations to one or more SMRS via Network Situational processes. In turn, each SMRS carries out its operations through Mediating, which is a process always hidden for network administrators. To assist Executing and Orchestrating, Mediating offers SMRS to Network Situational Processes and delegates network management operations to SMR. This mediation is needed because there is not a common format neither a standardized interface/protocol to retrieve and/or bidirectionally interact with data, application logic, and user interfaces of SMR involved in NMSits. Saving&Reusing permits network administrators to store Mashments for their later reuse. Thus, Mashments can be extended and improved to create other ones or customized to handle analogous NMSits.

Leveraging the automation of Network Situational Procesess and Mashup Processes, our approach enables network administrators to: (*i*) collect, correlate, and fuse information about NMSits, (*ii*) present information related to NMSits, in a visual and comprehensible way, (*iii*) perform network management operations to resolve (change or preserve) NMSits, (*iv*) build up composite situational solutions, in a Mashment manner, targeted to address NMSits; and (*v*) as a global result, to overcome the complexity of network management tasks in front of NMSits.

B. Process and Complexity in the Development and Execution of Mashments

Considering the conceptual model to address NMSits by Mashments, the set of Mashments is formally expressed as: $Mashment = \{mashment_x | mashment_x = (R_{used}, r_{root}, \delta, NMSit_{addr})\}$. Where, R_{used} is the set of resources (SMRS, GUIs, Mashup Operations, and Mashments) used in the $mashment_x$ creation, r_{root} is the root resource ($\in R_{used}$) that starts the $mashment_x$ execution, δ is the execution flow (*i.e.*, investigative and resolutive plans) of resources that make up the $mashment_x$, and $NMSit_{addr}$ is the set of one or more $nmsits$ addressed by the particular $mashment_x$. It is noteworthy to mention that $NMSit$ is the set of *Situations* happening in the network management domain and $NMSit_{addr} \subseteq NMSit$.

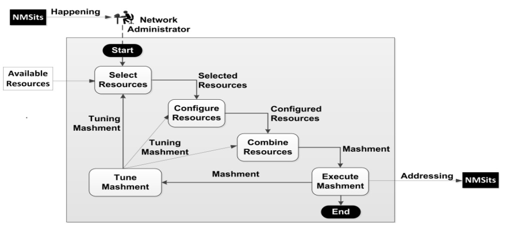


Figure 2: Process to Develop and Execute Mashments

In our approach, network administrators are able to tackle $nmsits$ by performing the following process (see Figure 2) that allows to develop and execute $mashments$. Such a process is formed by the tasks: Select, Configure, Combine, Execute, and Tune.

Select Resources. The network administrator defines the R_{used} from Available Resources. This task is divided in two: ***(i)*** The network administrator selects the SMRS, GUIs, and Mashup Operations needed to create the $mashment_x$. ***(ii)*** If it is feasible (there is a $mashment_y$ that addresses similar $nmsits$), the network administrator chooses one or more elements of the set $Mashment$ (it is part of available resources) to reuse them. ***Configure Resources***. The network administrator provides the functioning settings of one or several elements belonging to R_{used}, defining the set of resources configured $R_{conf} \subseteq R_{used}$. ***Combine Resources***. The network administrator defines the δ of $mashment_x$ that is formed by combining (connecting/linking) the selected and configured resources. It is important to highlight that the δ creation includes the definition of r_{root}. ***Execute Mashment***. The network administrator launches the $mashment_x$. ***Tune Mashment***. If it is needed, the network administrator tunes the δ of $mashment_x$ under construction, which may imply the selection, configuration, and combination of new resources or simply the re-arrangement of R_{used}.

The complexity of $mashment_x$ (*i.e.*, ζ) is calculated by computing the individual complexity of tasks forming the process aforedescribed. In this way:

$$\zeta = \sum_{1}^{i} \zeta_{sel} + \sum_{1}^{j} \zeta_{con} + \sum_{1}^{k} \zeta_{com} + \sum_{1}^{e} \zeta_{exe} \quad (1)$$

Where, ζ_{sel}, ζ_{con}, ζ_{com}, and ζ_{exe} represent the complexity of Select, Configure, Combine, and Execute, respectively. In turn, i, j, k, and e denote the number of times that such tasks are conducted, allowing to consider the complexity of Tune. In the next paragraphs, ζ_{sel}, ζ_{con}, ζ_{com}, and ζ_{exe} are expressed by using per-task metrics defined for IT Service Management processes [25].

The complexity of Select is expressed as:

$$\zeta_{sel} = \sum_{m=1}^{M} \varsigma_m + (nAvailableResources - 1) * gF * cF \quad (2)$$

Where, M is the total number of elements on R_{used} and $\varsigma_m = selType(m)$ is the complexity of selecting the m-resource. Here, $selType(m)$ can take one of three values depending on the automation of m-selection: 0 - if fully automated, 1 - if manual but tool-assisted, or 2 - if manual. In turn, $nAvailableResources$ is the number of resources available to build up the $mashment_x$ (*i.e.*, more available resources result in higher complexity of selection). gF is the grade of guidance provided to select the resources needed to form the $mashment_x$. gF can take one of three values: 1 - if correct recommendation about resources to be selected is offered, 2 - if general information about each available resource is supplied, or 3 - if non information is provided. cF represents the impact of wrong selection of resources and can take one of three values: 0 - if negligible impact, 1 - if moderate impact, or 2 - if severe impact.

The complexity of Configure is defined as:

$$\zeta_{con} = \sum_{n=1}^{N} \varsigma_n. \quad (3)$$

Where, N is the total number of resources on R_{conf} and ς_n is the complexity of configuring the n-resource. Note that as $R_{conf} \subseteq R_{used}$, so, $N \leq M$. $\varsigma_n = \sum_{p=1}^{P} sourceParameter(p)$. Here, P is the total number of parameters to be configured in the n-resource and $sourceParameter(p)$ can take one of seven values: 0 - if the p-parameter value is produced from automation, 1 - if the p-parameter value may be chosen freely (*e.g.*, a new password), 2 - if the p-parameter value is taken from task documentation (*e.g.*, set up port=8080 for a HTTP server), 3 - if the p-parameter value is extrapolated from task documentation (*e.g.*, define a range of IP addresses), 4 - if the p-parameter value is not trivial for unexperienced network administrators (*e.g.*, set up the URL=$http : //IPAddressOfXenServer/rrdUpdates?host = true$ to retrieve statistics of virtual machines running on a determined XenServer), 5 - if the p-parameter is fixed by the environment to a specific value that is defined after additional research (*e.g.*, set up the SNMP OID=1.3.6.1.4.1.9.9.91.1.1.1.1.4 to obtain the temperature of Catalyst Cisco Switch), or 6 - if the p-parameter value is constrained by the environment to a limited set of possible choices where network administrators need to infer the right choice (*e.g.*, set up the type of server virtualization technology to be monitored: virtTech=VMware).

The complexity of Combine is expressed as:

$$\zeta_{com} = \sum_{l=1}^{L} linkType(l) + (M-1) * goF * coF \qquad (4)$$

Where, L is the total number of links (logical connections) created to build up the $mashment_x$. $linkType(l)$ represents the complexity of creating the l-link that connects two elements of R_{used} and can take one of four values: 0 - if the l-link is automatically created, 1 - if the l-link is manually created by a support tool and data transferred among resources connected must not be adapted, 2 - if the l-link is manually created and data transferred among resources connected must not be adapted, and 3 - if the l-link is manually built and data transferred among resources connected must be adapted. In turn, M is the total number of R_{used} (*i.e.*, more selected resources result in higher complexity of combination). goF is the grade of guidance provided to link the selected resources and can take one of three values: 1 - if correct recommendation to link the selected resources is supplied, 2 - if general information about the links that can be established is offered, or 3 - if non information is provided. coF represents the impact of wrong combination of resources and can also take one of three values: 0 - if negligible impact, 1 - if moderate impact, or 2 - if severe impact.

ζ_{exe} can take one of three values depending on the automation of $mashment_x$ execution: 0 - if entirely automated (*e.g.*, an autonomous mashment system that executes $mashments$ on demand), 1 - if manual but tool-assisted (*e.g.*, using an execution environment to start the $mashment_x$), or 2 - if manual (*e.g.*, programming and customizing a script every time that the $mashment_x$ needs to be executed).

C. Mashment Maker Architecture

The Mashment Maker is defined to accomplish the following goal: to support the conceptual model of Mashments and, as a consequence, their developing and executing process. Therefore, the Maker is targeted to decrease the intricacy of Select (ζ_{sel}), Configure (ζ_{con}), Combine (ζ_{com}), Execute (ζ_{exe}), and Tune (*i.e.*, the complexity of repeating once or more times the other tasks). The Figure 3 depicts the Mashment Maker Architecture that is formed by: SMRS, Mashment Operations, Mediator Bus, Visual Resources (*i.e.*, Visual-SMRS, Visual-BI, Visual-MO, and Visual-Mashment), Designer, Contextual Help System (CHS), Mashment Router, Mashment Engine, Mashment Repository, and Users Repository.

The Mediator Bus provides as a service the Mediating Process and enables the communication among all elements

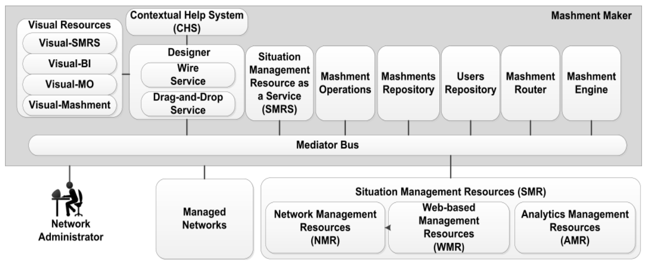


Figure 3: Mashment Maker Architecture

of the Maker Architecture. Mashment Operations are services that supply the functionalities of both Network Situational Processes and Mashup Operations. The functioning of SMRS (NMRS, WMRS, and AMRS), Network Situational Processes, Mashup Operations, and Mediating Process was already described in the subsection III-A. Here, it is important to point out that, first, the Bus, Mashment Operations, and SMRS are key to achieve the Maker goal because these architectural elements drive the intricacy of underlying technologies involved in the investigation and resolution of NMSits. Second, network administrators never have direct access to these three elements. This access is always conducted through Visual Resources.

Visual Resources represent SMRS, GUI, Mashments Operations, and Mashments, in a high-level abstraction, in order to hide complexity for network administrators. Visual-SMRS includes three types of resources: *(i)* Visual-NMRS (*e.g.*, a box offering management functionalities of Vyatta Virtual Router) represents NMRS, *(ii)* Visual-WMRS (*e.g.*, a box representing functional features of RRDTool) depicts WRMS; and *(iii)* Visual-AMRS (*e.g.*, a box providing functions of the Management Traffic Analyzer) supplies, in a graphic way, AMRS.

A Visual-BI offers an useful basic user interface to create composite and advanced Mashment GUIs. For instance, a Mashment GUI can be composed by inserting one or more network traffic images (from NMRG) into a map (from Google Maps). In turn, a Visual-MO graphically represents a Mashment Operation. A box offering a dashboard (it hides the collection, correlation, and fusion of network management information) to monitor heterogeneous OpenFlow Controllers is an example of Visual-MO. On design time, to facilitate the reuse, each existing Mashment is depicted as a Visual-Mashment.

The Designer allows network administrators to develop and execute Mashments. Accordingly, the Designer, first, provides services for the δ definition by means of Dragging-and-Dropping and Wiring of Visual Resources. Second, it offers capabilities for saving, deleting, loading, and launching Mashments. In this sense, on design time, Saving permits to write in the Mashment Repository the δ of Mashments. Deleting allows to remove a specific δ. Loading is responsible for reading the δ of Mashments and visually presenting them (*i.e.* generate Visual-Mashments). Launching permits to request to the Engine the execution of a determined Mashment. Third, the Designer uses CHS to offer guidance about Visual Resources, Mashments, and the Maker as a whole. All Designer functionalities are targeted to facilitate the creation, re-usage and execution of Mashments and, as a consequence, to reduce ζ_{sel}, ζ_{con}, ζ_{com}, and ζ_{exe}. Also, such functionalities are key to permit network administrators to customize their workspace when addressing NMSits.

The Mashment Repository stores the metadata of Mashments built in the Designer. The metadata of a specific Mashment is an object containing the information/definition of its δ. If a Mashment is formed by one or more Mashments, its metadata also includes the metadata of these Mashments. This inclusion means that a δ can encompass other δs. In turn, the Users Repository stores the data of Network Administrators; this data is used to perform the access control to the Maker.

The Mashment Router is in charge of performing the δ of Mashments. Thus, on run time, the Router is responsible for: *(i)* receiving Mashments invocations from the Engine, which means that the Router is called by the Engine to select a Mashment to service an initial request, *(ii)* selecting and chaining multiple resources (including Mashments into a Mashment) to attend invocations, by reading the needed information from repositories of Mashments and Users; and *(iii)* calling the Engine to request the instantiation of Mashments and their constitutive elements. It is to point out that the Router is required by the Engine to function, but the Router is a separate architectonic element.

The Mashment Engine is a lifecycle manager, responsible for creating, deleting, and caching instances of Mashments and their associated resources. The Engine is splitted in two: Server-Side Engine and Client-Side Engine. The Server-Side is a Web engine that supports the execution of SMRS, Network Situational Processes, and Mashup Operations. The Client-Side is a Web browser engine that supports Web 2.0 technologies to be able to run the integrated and advanced user interfaces of Mashments.

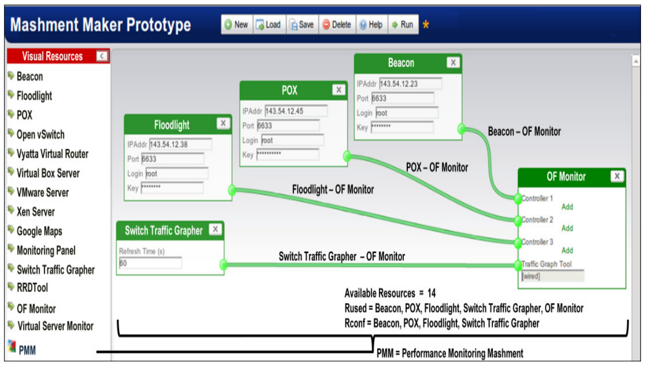
Concerning the Maker, it is important to highlight that: *(i)* the Drag-and-Drop Service assists Select, aiming to reduce $selType(m)$, *(ii)* CHS provides guidelines for supporting Select, Configure, and Combine, which is targeted to diminish, respectively, gF, $sourceParameter(f)$, and goF, *(iii)* the Wire Service, that does not require data mapping, bears Combine, aiming to cut down $linkType(l)$, *(iv)* the high-level launching mechanism, integrated in the Designer, expedites the running of every $mashment$, seeking to decrease ζ_{exe}; and *(v)* high-level Visual Resources allow the flexible construction of Mashments, in a designer-assisted way, which is directed to cut down ζ_{sel}, ζ_{con}, and ζ_{exe}.

IV. Case Study

To assess our approach, first, we performed a test environment make up of the Mashment Maker prototype, three SDN-based networks built using OpenFlow, and a NMSit that happens in these networks. Second, we conducted experiments to measure the complexity of addressing such a NMSit when the network administrator follows the proposed process with and without the Maker. In this section, we describe the test conditions, present the experiments, and analyze the obtained results.

A. Test Environment

Mashment Maker Prototype. The Figure 4 depicts the GUI of Maker. This GUI is formed by the Designer, the Buttons (*i.e.*, New, Load, Save, Delete, Help, and Run), the Visual Resources (*i.e.* Beacon, Floodlight, POX, Open vSwitch, Vyatta Virtual Router, Virtual Box Server, VMware Server, Xen Server, Google Maps, Monitoring Panel, Switch Traffic Grapher, RRDTool, OF Monitor, Virtual Servers Monitor, and the *Performance Monitoring Mashment - after described*), and CHS.



Beacon: IPAddr 143.54.12.23, Port 6633, Login root, Key ********

POX: IPAddr 143.54.12.45, Port 6633, Login root, Key **********

Floodlight: IPAddr 143.54.12.38, Port 6633, Login root, Key ********

Switch Traffic Grapher: Refresh Time (s) 60

OF Monitor: Traffic Graph Tool [wired]

Available Resources = 14

Rused = Beacon, POX, Floodlight, Switch Traffic Grapher, OF Monitor

Rconf = Beacon, POX, Floodlight, Switch Traffic Grapher

PMM = Performance Monitoring Mashment

Figure 4: Mashment Maker Prototype and Performance Monitoring Mashment

The Designer is a Web application built using YUI 2.7 and WireIt 0.5. YUI is an open source, javascript, and cascading style sheets framework, used to implement the Drag-and-Drop Service. WireIt is a set of open source javascript libraries, used to create the Wire Service. The Buttons and Visual Resources (*e.g.*, the Switch Traffic Grapher) are also javascript components, implemented on YUI. CHS is a GUI component developed using cascading style sheets and javascript.

The Mediator Bus, SMRS, and Mashment Operations were developed using the Java Language, like Web Services based on the Representational State Transfer (REST) architectural style. We use REST-based services because they are suitable to achieve integration and interoperability in heterogeneous environments. Each Web Service that interacts with Beacon, Floodlight, and POX was created using the Java Jersey API 2.3, the Floodlight REST API 1.0, and the Java Socket API 1.0, respectively. In turn, each Web Service that communicates with VirtualBox, Xen, and VMware was correspondingly implemented using the VirtualBox SDK API 4.1, the XenSDK API 6.0, and the VMware WebServices SDK 5.1. The Mashment Router was built with Java Servlets and the Asynchronous Javascript and XML (AJAX). We use Servlets and AJAX because they allow the interactive and asynchronous interaction among the Maker GUI and the aforementioned Web Services.

The Mashment Maker prototype was unfolded (see Figure 5) in the Apache-Tomcat Server 7.0 and the MySQL Server 5.1. Specifically, in the Apache-Tomcat were deployed the Designer, Buttons, Visual Resources, CHS, SMRS, Mediator Bus, and Mashment Operations. In the MySQL were installed the Users Repository and Mashments Repository. The browser Mozilla Firefox was the client used to run the Maker GUI.

OpenFlow Networks. Three OpenFlow networks (see Figure 5) were built using, in the control tier, Beacon 1.0, Floodlight 0.9, and POX 1.0. Each OpenFlow controller was deployed to handle 27 Open vSwitches located in the datapath tier. These switches were deployed, in a tree topology, on Mininet that is an emulation platform for OpenFlow networks. The communication among the controllers and their corresponding Open vSwitches was made by the OpenFlow protocol 1.0.

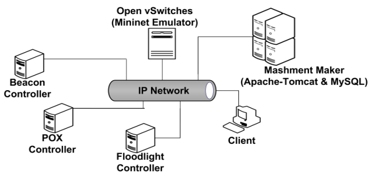


Figure 5: Test Environment

NMSit-SDN. Let's suppose the following *Situation*: the network administrator needs to identify which are the Open vSwitches that are causing sudden performance degradation on the OpenFlow networks of the test environment described earlier. Thereby, he/she requires a situational solution that presents, in an integrated, visual, and intelligible way, network traffic information of Open vSwitches handled by Beacon, POX, and Floodlight. To get such a solution and deal with the *NMSit-SDN*, the network administrator has two options: *(i)* Without the Maker, to create and launch a *Situational Script*; or *(ii)* With the Maker, to develop and execute the *Performance Monitoring Mashment*. The complexity and detail of developing and executing these options is presented in the next subsection.

B. Complexity: Evaluation and Analysis

To evaluate our approach, initially, we measured the complexity of addressing the *NMSit-SDN* when the network administrator follows the proposed process to tackle NMSits but does not use the Maker. In a workspace without the Maker, he/she develops and executes a *Situational Script* that retrieves network traffic information from the switches handled by Beacon, POX, and Floodlight. Such a *Script* presents the information retrieved in a user interface formed by text-plane tables and chart images. According to the equation (1) and considering the no conducting of Tune ($i = j = k = e = 1$), $\zeta_{nomaker} = \zeta_{sel:nomaker} + \zeta_{con:nomaker} + \zeta_{com:nomaker} + \zeta_{exe:nomaker}$.

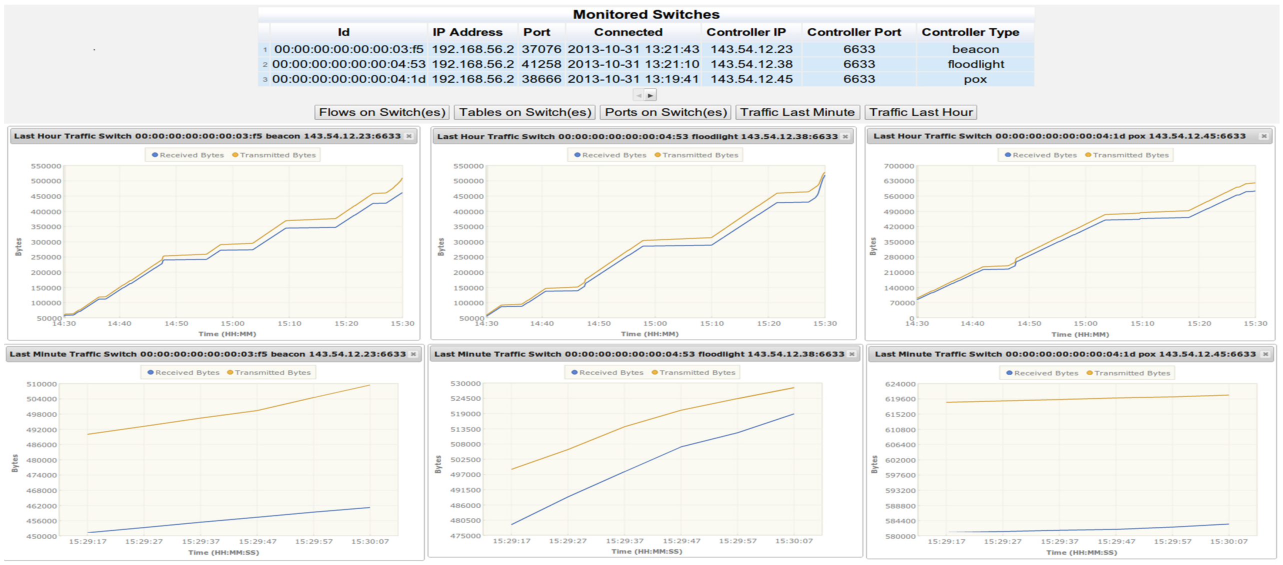
Select without Maker. The network administrator performs the selection of controller tools (*i.e.*, BeaconTool, POXTool, and FloodlightTool) and their specific commands that allow to monitor Open vSwitches. An example of specific command is to retrieve the statistics of an Open vSwitch controlled by Floodlight: curl http://IPAddress:8080/wm/core/switch/switchId/statType/json. The above mentioned selection is complex because it is not tool-assisted and guidelines are scattered on the Internet. In this way, $\varsigma_m = 2$, $gF = 3$, and $cF = 1$. Using these values in the equation (2), $\zeta_{sel:nomaker} = \sum_{m=1}^{4} 2 + (nAvailableResources - 1) * 3 * 1$. Where, considering $nAvailableResources = 14$, we use this value to facilitate the comparison with the Maker prototype, $\zeta_{sel:nomaker} = 47$.

Configure without Maker. The network administrator configures $N = 4$ resources (BeaconTool, POXTool, FloodlightTool, and YUI Chart API) by providing their corresponding functioning parameters. Thus, in accordance to the equation (3), $\zeta_{con:nomaker} = \varsigma_{beaconTool} + \varsigma_{poxTool} + \varsigma_{floodlightTool} + \varsigma_{yc}$. Where, $\varsigma_{beaconTool} = \varsigma_{poxTool} = \varsigma_{floodlightTool} = sourceParameter(login) + sourceParameter(key) + sourceParameter(ip) + sourceParameter(port) + sourceParameter(statisticCommand)$. As he/she takes the configuration information of controller tools from documentation easy to find on the Internet and defines the specific statistic commands after additional search, $sourceParameter(login) = sourceParameter(key) = sourceParameter(ip) = sourceParameter(port) = 2$ and $sourceParameter(statisticCommand) = 5$. Furthermore, since he/she extrapolates the YUI Chart configuration information from documentation simple to find on the Internet, $\varsigma_{yc} = 3$. Using these values, $\zeta_{con:nomaker} = 42$.

Combine without Maker. The network administrator manually develops (writes programming code) one logical link among each of controller tools and the YUI Chart API. Regarding these links, it is to point out that: *(i)* he/she adapts the data retrieved because controller tools, involved in the *NMSit-SDN*, use different data type (*e.g.*, Beacon employs data type in Java and Floodlight uses JSON); and *(ii)* he/she neither has explicit nor centralized guidelines to support the links development. Thus, $linkType(l) = 3$, $goF = 3$, and $coF = 1$. Using these values in the equation (4), $\zeta_{com:nomaker} = 21$.

Execute without Maker. As the network administrator launches the *Situational Script* by typing a specific command in a Linux Command Line, $\zeta_{exe:nomaker} = 2$. After executing this *Script*, the network administrator is able to find the Open vSwitches involved in the *NMSit-SDN*, by analyzing YUI Chart images and text-plane tables.

Once computed the intricacy of facing the *NMSit-SDN* without the Maker, we proceed to evaluate the complexity of developing and executing the *Performance Monitoring Mashment*. In a broad sense, the network administrator builds and launches this Mashment (see Figure 4) by dragging-and-dropping, wiring, and clicking several Visual Resources and Buttons of the Maker. The Maker also assists such a process by providing contextual guidelines for the network administrator. In accordance to the equation (1) and considering that



Monitored Switches

	Id	IP Address	Port	Connected	Controller IP	Controller Port	Controller Type
1	00:00:00:00:00:00:03:f5	192.168.56.2	37076	2013-10-31 13:21:43	143.54.12.23	6633	beacon
2	00:00:00:00:00:00:04:53	192.168.56.2	41258	2013-10-31 13:21:10	143.54.12.38	6633	floodlight
3	00:00:00:00:00:00:04:1d	192.168.56.2	38666	2013-10-31 13:19:41	143.54.12.45	6633	pox

Flows on Switch(es)
Tables on Switch(es)
Ports on Switch(es)
Traffic Last Minute
Traffic Last Hour
Last Hour Traffic Switch 00:00:00:00:00:00:03:f5 beacon 143.54.12.23:6633
Last Hour Traffic Switch 00:00:00:00:00:00:04:53 floodlight 143.54.12.38:6633
Last Hour Traffic Switch 00:00:00:00:00:00:04:1d pox 143.54.12.45:6633
Last Minute Traffic Switch 00:00:00:00:00:00:03:f5 beacon 143.54.12.23:6633
Last Minute Traffic Switch 00:00:00:00:00:00:04:53 floodlight 143.54.12.38:6633
Last Minute Traffic Switch 00:00:00:00:00:00:04:1d pox 143.54.12.45:6633
Received Bytes
Transmitted Bytes
Bytes
Time (HH:MM)
Time (HH:MM:SS)

Figure 6: Performance Monitoring Mashment on Runtime

he/she just one time conducts Select, Configure, Combine, and Execute, $\zeta_{pmm} = \zeta_{sel:maker} + \zeta_{con:maker} + \zeta_{com:maker} + \zeta_{exe:maker}$.

Select on Maker. The network administrator uses the Drag-and-Drop service (*i.e.*, a Maker-assisted way) to select the Visual Resources ($M = 5$) that form the *Performance Monitoring Mashment*. Thus, R_{used} ={*Beacon, POX, Floodlight, Switch Traffic Grapher, OFMonitor*}. Furthermore, to facilitate such a selection, the Maker via CHS provides him/her contextual guidance about each of $nAvailableResources = 14$. Therefore, $\varsigma_m = 1$, $gF = 2$, and $cF = 1$. Using these values in the equation (2), $\zeta_{sel:maker} = 31$.

Configure on Maker. The network administrator configures ($N = 4$) Visual Resources, R_{conf} ={*Beacon,POX,Floodlight,Switch Traffic Grapher*}, by providing their functioning settings. Then, in accordance to the equation (3), $\zeta_{con:maker} = \varsigma_{beacon} + \varsigma_{pox} + \varsigma_{floodlight} + \varsigma_{stg}$. Where, $\varsigma_{beacon} = \varsigma_{pox} = \varsigma_{floodlight} = sourceParameter(login) + sourceParameter(key) + sourceParameter(ip) + sourceParameter(port)$ and $\varsigma_{stg} = sourceParameter(refreshTime)$. Considering that the Maker via CHS offers him/her configuration guidelines about Visual Resources, $\varsigma_{beacon} = 8$ and $\varsigma_{stg} = 2$. Using these values, $\zeta_{con:maker} = 26$.

Combine on Maker. The network administrator uses the Wire Service ($linkType(l) = 1$) to create $L = 4$ links: Beacon - OF Monitor, POX - OF Monitor, Floodlight - OF Monitor, and Switch Traffic Grapher - OF Monitor. Regarding these links, it is to stand out that: *(i)* he/she does not need to adapt the data transferred because the Mediator Bus is responsible for hiding the data mapping; and *(ii)* he/she obtains guidelines about links creation from the Maker via CHS. Therefore, $goF = 2$ and $coF = 1$. Using these values in the equation (4), $\zeta_{com:maker} = 12$.

Execute on Maker. Since the network administrator can run the *Performance Monitoring Mashment* from the Designer by clicking the Run Button, $\zeta_{exe:maker} = 1$. After launching this *Mashment* (see Figure 6), the network administrator can identify the three Open vSwitches implicated in the *NMSit-SDN*, by analyzing, in an integrated GUI, Switch Traffic Grapher images and HTML tables.

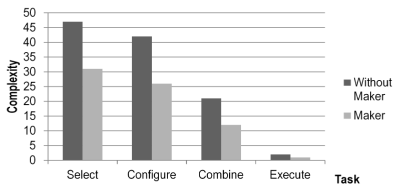


Figure 7: NMSit-SDN: Tasks Complexity

The Figure 7 depicts the obtained results in the complexity assessment when the network administrator faces the *NMSit-SDN* with and without the Maker. In accordance to these results, the use of the Maker to conduct the proposed process: *(i)* diminishes the complexity of Select in 34.04%, $\zeta_{sel:maker} = 31 < \zeta_{sel:nomaker} = 47$, attained by the services Drag-and-Drop and CHS, *(ii)* reduces the complexity of Configure in 38.09%, $\zeta_{con:maker} = 26 < \zeta_{con:nomaker} = 42$, reached by CHS, *(iii)* decreases the complexity of Combine in 42.85%, $\zeta_{com:maker} = 12 < \zeta_{com:nomaker} = 21$, obtained by the Wire Service and the Mediator Bus; and *(iv)* diminishes the complexity of Execute in 50%, $\zeta_{exe:maker} = 1 < \zeta_{exe:nomaker} = 2$, gotten by the Designer.

Since in a Maker-based workspace the complexity of each task is less than the corresponding complexity when the Maker is not used, $\zeta_{pmm} = 70$ is also less than $\zeta_{nomaker} = 112$ and the global reduction is 37.50%. Considering the results of the raised case study, we demonstrated that if network administrators follow the process to develop and execute Mashments in the Maker, the complexity of handling NMSits is decreased. Consequently, we conclude that our approach can be used by network administrators to overcome the complexity of NMSits.

V. Conclusions and Future Work

In this paper, we introduced an approach that allows to overcome the complexity on the work performed by network administrators to face NMSits. The approach is formed by the Mashments conceptual model, the process to develop and execute Mashments, and the Mashment Maker that supports such model and process. Furthermore, we presented the complexity evaluation of process that the network administrator conducts to build up and run two solutions: *Situational Script* and *Performance Monitoring Mashment*. Both solutions were targeted to address the *NMSit-SDN*: identify switches that are suddenly causing performance degradation on several OpenFlow networks handled by different controllers.

Our approach permitted the network administrator to address the complexity involved in overcoming the *NMSit-SDN*, confirming the importance of the Mashment conceptual model, the process to develop and execute Mashments, and the Mashment Maker. In this sense, using per-task metrics, we demonstrated that the complexity decreases when network administrators conduct the following situational tasks: Select, Configure, Combine, and Execute. As a result, we can state that our approach cuts down the complexity on the work carried out by network administrators to cope with NMSits.

We consider the proposed approach as a step forward in the network management, the situation management, and the mashup technology. In this regard, we drive the first towards an environment focused on situations, composite situational solutions, and network administrators. We bring mashup foundations up to the second to perform its investigative and control aspects. We lead the third to a novel application domain located in the intersection of the situation management and the network management.

As future work, we plan to correlate time and complexity metrics, in order to evaluate the productivity of network administrators that face NMSits by Mashments. Furthermore, we are interested in propose the deployment costs model of our approach by considering the heterogeneity of resources to be integrated/combined. Finally, we also pretend to add more resources and services to improve the Maker implementation.

Acknowledgment

The research of PhD(c) Caicedo is supported by the PECPG of the CAPES (Brazil) and the University of Cauca (Colombia).

References

[1] G. Jakobson, J. Buford, and L. Lewis, “Situation Management: Basic Concepts and Approaches,” in *Information Fusion and Geographic Information Systems*, ser. Lecture Notes in Geoinformation and Cartography, W. Cartwright, G. Gartner, L. Meng, M. . Peterson, V. . Popovich, M. Schrenk, and K. . Korolenko, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, ch. 2, pp. 18–33.

[2] Kokar, Mieczyslaw M. and Matheus, Christopher J. and Baclawski, Kenneth, “Ontology-based Situation Awareness,” *Information Fusion*, vol. 10, no. 1, pp. 83–98, january 2009.

[3] G. Jakobson, L. Lewis, C. Matheus, M. Kokar, and J. Buford, “Overview of Situation Management at SIMA,” in *MILCOM*, 2005, pp. 1630 –1636 Vol. 3.

[4] S. George, W. Zhou, H. Chenji, M. Won, Y. O. Lee, A. Pazarloglou, R. Stoleru, and P. Barooah, “DistressNet: a Wireless ad hoc and Sensor Network Architecture for Situation Management in Disaster Response,” *Communications Magazine*, vol. 48, no. 3, pp. 128–136, 2010.

[5] B. Magoutas, G. Mentzas, and D. Apostolou, “Proactive Situation Management in the Future Internet: The Case of the Smart Power Grid,” in *DEXA*, september 2011, pp. 267 –271.

[6] D. M. Hein, R. Toegl, M. Pirker, E. Gatial, Z. Balogh, H. Brandl, and L. Hluchý, “Securing Mobile Agents for Crisis Management Support,” in *STC*. New York, NY, USA: ACM, 2012, pp. 85–90.

[7] Pereira, I.S.A. and Costa, P.D. and Almeida, J.P.A., “A Rule-based Platform for Situation Management,” in *CogSIMA*, 2013, pp. 83–90.

[8] O. Caicedo, F. Estrada, and Granville., “A Mashup Ecosystem for Network Management Situations,” in *Globecom*, december 2013, p. to appear.

[9] X. Chen, Y. Mao, Z. M. Mao, and J. Van der Merwe, “Declarative Configuration Management for Complex and Dynamic Networks,” in *Co-NEXT*. New York, NY, USA: ACM, 2010, pp. 6:1–6:12.

[10] N. Kim and J. Kim, “Building NetOpen Networking Services over OpenFlow-based Programmable Networks,” in *ICOIN*, january 2011, pp. 525 –529.

[11] J. de Santanna, J. Wickboldt, and L. Granville, “A BPM-based Solution for Inter-domain Circuit Management,” in *NOMS*, 2012, pp. 385–392.

[12] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, “Composing Software-defined Networks,” in *NSDI*. Berkeley, CA, USA: USENIX Association, 2013, pp. 1–14.

[13] H. Kim and N. Feamster, “Improving Network Management with Software Defined Networking,” *Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[14] D. E. Simmen, M. Altinel, V. Markl, S. Padmanabhan, and A. Singh, “Damia: Data Mashups for Intranet Applications,” in *ACM SIGMOD*. New York, NY, USA: ACM, 2008, pp. 1171–1182.

[15] N. Laga, E. Bertin, R. Glitho, and N. Crespi, “Widgets and Composition Mechanism for Service Creation by Ordinary Users,” *Communications Magazine*, vol. 50, no. 3, pp. 52–60, 2012.

[16] N. Ozkan and W. Abidin, “Investigation of Mashups for Managers,” in *ISCIS*, september 2009, pp. 622 –627.

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

[18] V. Stirbu, Y. You, K. Roimela, and V. Mattila, “A Lightweight Platform for Web Mashups in Immersive Mirror Worlds,” *Pervasive Computing*, vol. 12, no. 1, pp. 34–41, 2013.

[19] Y. Han, G. Wang, G. Ji, and P. Zhang, “Situational Data Integration with Data Services and Nested Table,” *Service Oriented Computing and Applications*, vol. 7, no. 2, pp. 129–150, 2013.

[20] C. dos Santos, R. Bezerra, J. Ceron, L. Granville, and L. Rockenbach Tarouco, “On Using Mashups for Composing Network Management Applications,” *Communications Magazine*, vol. 48, no. 12, pp. 112–122, december 2010.

[21] O. Caicedo, F. Estrada, and Granville., “A Mashup-based Approach for Virtual SDN Management,” in *COMPSAC*, july 2013, pp. 143–152.

[22] J. Yu, B. Benatallah, F. Casati, and F. Daniel, “Understanding Mashup Development,” *Internet Computing*, vol. 12, no. 5, pp. 44 –52, sept.-oct. 2008.

[23] C. dos Santos, R. Bezerra, J. Ceron, L. Granville, and L. Tarouco, “Botnet Master Detection Using a Mashup-based Approach,” in *CNSM*, october 2010, pp. 390 –393.

[24] B. R.S., C. dos Santos, L. Bertholdo, L. Granville, and L. Tarouco, “On the Feasibility of Web 2.0 Technologies for Network Management: A Mashup-based Approach,” in *NOMS*, april 2010, pp. 487 –494.

[25] Y. Diao and A. Keller, “Quantifying the Complexity of IT Service Management Processes,” ser. DSOM’06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 61–73.

A Mashup Ecosystem for Network Management Situations

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*—Current network management approaches and their implementations are not intended to address dynamic situations that need rapid delivery of good-enough and comprehensive solutions. In this paper, we introduce a novel mashup ecosystem, called Mashment Ecosystem, that allows Network Administrators to conduct on a Mashment Maker the activities and interactions necessary to provide Mashments. Mashments are mashups aimed to tackle network management situations. We evaluate the Mashment Ecoystem by estimating with the Keystroke-Level Model and measuring in a test scenario the time that Network Administrators take to perform the activities of creating, launching, and publishing Mashments. Similarly, we evaluate the time for retrieving information about a network management situation by using or not Mashments. The evaluation results corroborated that Network Administrators, in our ecosystem, need short-time to deal with network management situations.**

I. Introduction

Nowadays, the use of computer networks has become vital to most enterprises. Up to now, in these networks, many management tasks require manual intervention by Network Administrators, mainly, to manage dynamic *Situations* that need rapid delivery of good-enough and comprehensive solutions [1]. In general, a *Situation* is a collection of entities (*i.e.*, things in a domain), their attributes, and relations in a time interval [2]. Hereinafter, we call a network management *Situation* as NMSit. Information technology departments do not provide situational solutions for NMSits because the requirements of these types of *Situations* are usually located in the long-tail of enterprise needs [3] [4]. As result, Network Administrators must create by themselves solutions for NMSits.

The Situation Management (SM) is an approach to provide solutions that enable to analyze, correlate, and coordinate the interaction between people, information, technologies, and actions intended to overcome *Situations* [2][5]. SM includes three aspects always linked to the time axis [2]: (*i*) the investigative aspect is related to retrospective cause analysis of *Situations* (*e.g.*, finding the root of a packet transmission failure in an OpenFlow-based network), (*ii*) the control aspect is directed to change or preserve *Situations* (*e.g.*, migrating a virtual switch from an overloaded server); and (*iii*) the predictive aspect is aimed to presage *Situations* (*e.g.*, projecting the appropriate time to migrate a critical router before a service disruption happen). In addition, as *Situations* are dynamic, SM emphasizes an adaptive management style.

Current network management solutions, such as Ganglia[6] and Nagios [7], are not intended to address the NMSits. Regarding these solutions, it is important to point out that they are not compatible, difficulting the information collection and information fusion that are necessaries to deal with a NMSit. Furthermore, the referred solutions are created without taking into account their rapid integration, extension, and improvement by Network Administrators, making hard the coping of NMSits. Thus, during a NMSit, the job of Network Administrators is hindered, since they are not able to enhance their workspace and must use numerous mismatched solutions from the Web and the network management.

Different research works have been developed about network management in traditional [8], virtual [9], Software Defined (SDN) [10], and cloud [11] networks. Such researches do not focus on face the NMSits and are developer centric. In the last years, mashups [12], that are end-user centric solutions formed by combining resources from different providers, have been applied in several domains as situational projects [13] and natural disasters [14] [15]. Nevertheless, mashups have not been used for tackling the NMSits. In this way, we raise the following question: how to tackle the NMSits by focusing on the Network Administrator?. In order to answer this question, we introduce a mashup ecosystem, named Mashment Ecosystem, which allows to carry out SM in network management. To the best of our knowledge, this work is the first to use an approach centric in the Network Administrator, SM, and mashup-based to deal with NMSits.

The Mashment Ecosystem, first, helps and encourages to Network Administrators to build up Mashments (*i.e.*, mashup used to deal with a NMSit) by themselves. Second, it allows Network Administrators to collect, correlate, and fuse information from heterogeneous resources offered by diverse providers. Third, it promotes the sharing and reuse of Mashments to avoid their wasting and to push the rise of innovative ones. Summarizing, the key contributions presented in this paper are to: (*i*) propose a novel Mashment Ecosystem for rapidly tackling NMSits by focusing in the Network Administrator; and (*ii*) demonstrate the short time that the Network Administrator needs to address a NMSit by building up and using a Mashment in our ecosystem.

The remainder of this paper is organized as follows. In Section II, we present SM and the mashup technology. In Section III, we introduce the Mashment Ecosystem. In Section IV, we expose and analyze the case study developed to evaluate our proposal. In Section V, we provide some conclusions and implications for future work.

II. Background

In this section, we present a SM background. Also, we describe mashups and research works about mashups–SM.

A. Situation Management

SM is an emerging approach to provide solutions that need the planning and implementing of actions aimed to overcome a determined Situation [2]. SM requires the use of novel techniques [5], first, to collect time/state information about *Situations*. Second, to correlate and fuse multi-source information for timely and correct decision making in *Situations*. Third, to analyze past and predict future *Situations*. Fourth, to present information aiming at the human comprehension maximization.

Solutions based on the SM concepts are found in several domains. For instance, in the domain of polyester film base manufacturing, a situational solution, based on an expert system, has been proposed for monitoring the non-steady state events and assisting human operators with the event tasks [16]. In the aviation domain, a scalable and distributed situational system has been introduced for the management of air security incidents such as terrorist attacks that need, to be overcomed, the coordination and sharing of information from different organizations [17]. An architecture, based on SM, the Service Oriented Architecture, and developer centric, has been outlined for supporting demand response aspects of the smart grid domain [18]. Althoug SM has been used in several domains, there is not a SM-based approach to deal with NMSits.

B. Mashups

Mashups are web applications centered in end-users and built up by combining several resources (*e.g.*, data, application logic, and user interfaces) from one or more providers [12]. Here, end-user centric means that mashups can be built by users without advanced programming skills. In addition, regarding the mashups is to noteworthy [19]. First, they encourage the sharing among end-users. Second, the providers that supply resources, the end-users/developers that create mashups, and the end-users that use mashups act as a single unit known as mashup ecosystem.

If a mashup is developed for rapidly coping an immediate need of one or a set of end-users, it can be considered as a situational solution [20]. Mashups have been useful to manage *Situations* in diverse domains. For instance, Mashups were used to help to overcome a fire emergency in San Diego (California, United States) by sharing weather and rescue information among civil organizations and the government [14]. In situational projects that involve a small number of users and have a short lifespan, a mashup environment has been introduced in order to support management tasks. In such environment, the project manager is able to quickly develop a mashup for visualizing and filtering the information of his/her project [13]. An architecture, based on the Web 2.0 and wireless sensor networks, has been proposed in order to estimate the speed and timing of possible floods. A mashup prototype that collects, correlates, and presents data from multiple wireless sensors was developed to test the architecture [15]. Despite the use of mashups in several domains, there is not a mashup-based approach for tackling the NMSits.

III. Mashment Ecosystem

In order to better explain our proposal, we present an overview of the Mashment Ecosystem. Subsequently, we describe its Resources, Stakeholders, Activities&Interactions, and Software Entities.

A. Overview

Current network management approaches do not focus on dealing with NMSits. In the same way, although the mashup technology provides good basis for developing composite situational solutions by end-users, it has not been used for tackling the NMSits. Therefore, there is a gap in the mashup and network management related research and, consequently, there is a chance for innovation. Hereinafter, we present how a Mashment Ecosystem, based on the abstraction of resources, the mashups composition model, and a Network Administrator centric approach, can be targeted to address the NMSits. In particular for coping the NMSits, the Mashment Ecosystem faces three issues: (*i*) the complexity and heterogeneity to collect, correlate, and fuse information from multiple resources of the Web and the network management, (*ii*) the demand by functionalities that allow Network Administrators to rapidly create adaptable solutions for NMSits; and (*iii*) the need by visualization functionalities that enable Network Administrators to get NMSit information, in a very understandable way.

Before detailing the Mashment Ecosystem, we introduce the Mashment concept and a motivating scenario. A Mashment is a tunable situational mashup that allows Network Administrators (their end-users) to tackle a NMSit by combining diverse types of resources from multiple providers. Tunable means that Mashments are adaptable and easily customizable. Since Mashments are a special type of mashup, they can be created by Network Administrators. Regarding a Mashment is also relevant to point out. First, it hides the heterogeneity, complexity, and stiffness of resources used to deal with a NMSit. Second, it bears the easy collection, correlation, and fusion of information about a NMSit. Third, it presents NMSit information, in a visual and clear way. Fourth, it can be rapidly created to cope a determined NMSit.

Motivating Scenario. Let's suppose the following NMSit: in a virtual network formed by OpenFlow-based heterogeneous Slices from different providers (NP_a, NP_b, and NP_c), a packet failure transmission occurs because of sudden and unidentified errors. To tackle this NMSit, the Network Administrator needs to found errors from Slices in NP_a, NP_b, and NP_c. As every NP uses a different OpenFlow Controller, aiming at overcoming this NMSit, the Network Administrator has three options. The first one is to collect, correlate, and visualize network monitoring information by using disparate solutions, such as command line interfaces to execute specific commands on each Controller, distinct web user interfaces to monitor virtual switches, and external web tools to display non-integrated information about packet traffic. A drawback of first option is that the use of several mismatched solutions consumes more time than use an integrated solution. The second option is to develop a low-level script to integrate the aforementioned commands, user interfaces, and web tools. This option also consume a lot of time because, the Network Administrator usually does not have advanced knowledge in programming. The third option is to participate in the Mashment Ecosystem. A Network Administrator in our ecosystem is able to quickly build up, in a high-level abstraction, by him/herself a Mashment to face the described NMSit. This Mashment hides the resources heterogeneity from NP_a, NP_b, and NP_c. Furthermore, the Mashment presents the network management information of virtual network in an integrated and intelligible way.

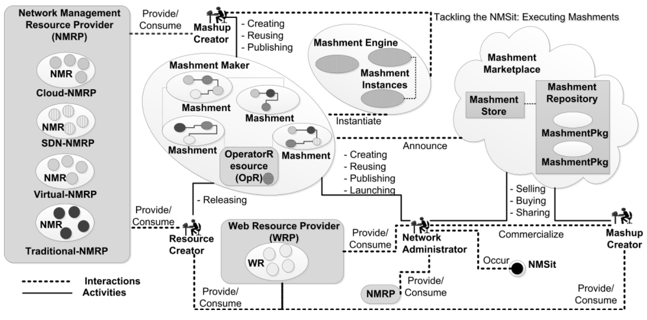


Fig. 1: Mashment Ecosystem

The Mashment Ecosystem (see Figure 1) is formed by: resources (Network Management Resources, Web Resources, and Operator Resources), Mashments, stakeholders (Network Administrator, Mashup Creator, Resource Creator, Web Resource Providers, Network Management Resource Providers, and Software Entity Providers), software entities (Mashment Maker, Mashment Engine, Mashment Repository, and Mashment Store), activities performed by stakeholders, interactions between stakeholders, and interactions between software entities. In this ecosystem, Network Administrators and Mashup Creators build up Mashments by using the Mashment Maker. Mashments are made up of resources from different providers and are executed in the Mashment Engine. The resources are released by the Resource Creator. In the Martketplace, the Mashments are shared, sold, and purchased by Mashup Creators and Network Administrators.

If a NMSit occurs, a Network Administrator can address it as follows: (*i*) buying or getting free of charge a Mashment(s), rapidly creating one or more Mashments, or quickly reusing a Mashment(s) previously built; and (*ii*) executing the purchased or created Mashment(s). It is important to highlight that the Mashment Ecosystem evolves over time because of the emerging and perishing of resources, the sharing and commercialization of Mashments, and the dynamic interactions between stakeholders.

The Mashment Ecosystem can be expressed as $MEco = \{M, St, A, I, Se\}$. Where: St, A, I, and Se are sets of stakeholders, activities, interactions, and software entities, respectively. In turn, the set of Mashments $M = \{m_i | m_i = (R_{used}, r_{root}, \delta, nmsit) : R_{used} \subset R, r_{root} \in R, nmsit \in NMSit\}$. Here, R_{used} is the set of resources on m_i, r_{root} is the root resource that starts the m_i execution, δ is the execution flow of resources on m_i, and $nmsit$ is the specific NMSit tackled by m_i. The other sets forming our ecosystem are described in next subsections.

B. Resources

A resource is a clearly identifiable entity in a time interval, which is conceived or can be adapted to tackle a NMSit. The set of resources is $R = \{r_i | r_i \in NMR \cup WR \cup OpR\}$. Where, Network Management Resources (NMR) are entities intended for the network management. NMR examples are Ganglia to manage traditional networks, Citrix Center for monitoring virtual resources, NetOpen to control OpenFlow-based networks, network monitoring systems based on the Simple Network Management Protocol, and all Application Programming Interfaces (API) that provide interaction with network elements.

Web Resources (WR) are Internet entities conceived or useful (via adaptation) for the network management. WR examples are the Google Maps API to show the geographic location of several network devices, the Multi Router Traffic Grapher (MRTG) to generate web pages with images presenting the traffic of network links, and the RRDTool to display over time the performance data of routers.

Operator Resources (OpR) are entities for combining resources (*i.e.*, NMR, WR, and even Mashments). There are two classes of OpR. Configuration_OpR to set up parameters for both access and communication to resources. An example of Configuration_OpR is a service to configure the security credentials required to monitor a virtual router. Control_OpR are composition patterns, such as $Split$, $Merge$, $Aggregate$, $Invoke$, $Trigger$, and $Receive$, useful, for instance, to collect, correlate, and fuse resources.

C. Stakeholders

A stakeholder affects and is affected by the activities and interactions performed by other one. The set of stakeholders is $St = \{st_i | st_i \in NMRP \cup WRP \cup SEP \cup ResourceCreators \cup MashupCreators \cup NetworkAdministrators\}$. Where, The Network Management Resource Provider (NMRP) is in charge of supplying NMR. Citrix Systems and Cisco Systems, providing solutions and programming interfaces to manage virtual servers and network devices, are examples of NMRP. The Web Resource Provider (WRP) is responsible for supplying WR. An example of WRP is a big player as Yahoo Inc. that provides visualization libraries and map services useful to present network management information. Another example is Oetieker&Partners Inc. that supplies web solutions intended for network monitoring, such as RDDTool and SmokePing.

The Software Entity Provider (SEP) is in charge of offering one or more software entities. In general, the Mashment Maker (containing visual representations of NMR, WR, OpR, and Mashments) and the Mashment Engine are provided, in an unified way, by the same SEP. In turn, the Mashment Repository and Mashment Store are usually offered, in a distributed way, by different SEP. In the Mashment Ecoystem can exist several Makers, Engines, Repositories, and Stores.

Before participating in the building of a Mashment, many WR and NMR need of adaption, in data format and/or communication protocol. The Resource Creator is responsible for this adaptation that we called releasing. Since such a releasing requires strong programming skills, Resource Creators are usually software companies and professional developers. The Open Software community, that provides APIs to interact via standardized protocols with network devices and servers containing virtual routers, is an example of Resource Creator.

The Mashup Creator creates, publishes, and launches Mashments by means of the Mashment Maker and the Mashment Engine. Also, he/she is able to share, sell, and buy Mashments in the Marketplace. A Mashup Creator can get profits by commercializing Mashments. Software companies, professional developers, and end-users are examples of this class of stakeholder.

The Network Administrator is responsible for tackling the NMSits in traditional, virtual, SDN, and cloud networks. The Mashment Ecosystem allows Network Administrators to: (*i*) create and execute Mashments, (*ii*) reuse existing Mashments, NMR, OpR, and WR, (*iii*) improve their workspace as result of (*i*) and (*ii*); and (*iv*) get profits through publishing and selling Mashments.

D. Activities and Interactions

The activities are actions conducted by stakeholders in the software entities. The set of activities is $A = \{Releasing, Creating, Reusing, Publishing, Launching, Selling, Buying, Sharing\}$. Where, $Releasing$ is carried out by Resource Creators to enable the combination of heterogeneous resources through adapting NMR and WR. After $Releasing$, the adapted resources can be used to build up Mashments.

Mashup Creators and Network Administrators perform $Creating$, $Reusing$, $Publishing$, $Sharing$, $Selling$, and $Buying$. $Creating$ allows to build up a Mashment (*i.e.*, define δ or execution flow), which involves: (*i*) discover the available resources (NMR, WR, OpR, and Mashments), (*ii*) select the suitable resources to address a NMSit, (*iii*) orchestrate a static or dynamic plan for tackling a NMSit, by combining the previously selected resources, (*iv*) monitor if the WR, NMR, and Mashments used on the orchestrated plan are available and flawless; and (*v*) reconfigure the orchestrated plan if any resource is in flaw or unavailable.

$Reusing$ enables to take advantage of existing Mashments, aiming at the creation of more complex and innovative Mashments. $Publishing$ allows to package and put Mashments in the Mashment Repository, aiming at their sharing, selling, and buying. $Sharing$ enables to offer and get Mashments free of charge. $Selling$ and $Buying$ allow the commercialization of Mashments. $Sharing$, $Selling$, and $Buying$ promote the reuse of Mashments and encourage the evolution of the Mashment Ecosystem. Mashments (built and purchased) are sent to execute through performing $Launching$, which, in turn, is conducted by Network Administrators. Every Mashment launched is called Mashup Instance.

The interactions take place in the relationships: stakeholder/stakeholder and software entity/software entity. The set of interactions can be expressed as $I = \{Provides, Consume, Tackling, Commercialize, Occur, Instantiate, Announce\}$. Where, $Provide$ and $Consume$ occur from the need of supplying and consumption of NMR and WR, during: the building up of the Mashment Maker, the resources releasing, and the Mashments creation that enhances and improves the Mashment Maker. $Provide$ and $Consume$ take place among: Resource Providers, Resource Creator, Mashup Creators, and Network Administrators.

$Instantiate$ is conducted among the Mashment Maker and the Mashment Engine when a Network Administrator launches a Mashment. $Announce$ is performed among the Mashment Maker and the Mashment Marketplace, aiming at publishing of Mashments that can be later shared, purchased, and sold in $Commercialize$. Mashup Creators and Network Administrators carry out $Commercialize$. $Occur$ and $Tackling$ are special interactions used to represent the emerging of a NMSit and the corresponding responses offered by Mashment Instances. During $Tackling$, Mashment Instances interact with NMR, WR, and OpR in order to face a NMSit.

E. Software Entities

The software entities are responsible for supporting and automating the activities and interactions aforedescribed. The set of software entities is $Se = \{Maker, Engine, Marketplace\}$. Where, the Mashment Maker allows Network Administrators and Mashment Creators to build up Mashments, in an easy and rapid way. The Maker is a development environment that provides a composition approach, high-level programming tools, and a lightweight development process. The composition approach consists of four phases related to $Creating$ and $Reusing$ activities: (*i*) discover and select, (*ii*) orchestrate, (*iii*) monitor and reconfigure; and (*iv*) reuse. The above phases enable, first, to use the last information retrieved from available resources. Second, to avoid the use of redundant, incomplete, or irrelevant information provided by resources in failure. Third, to avoid the lack of information because of unavailable resources.

The high-level programming tools are visual facilities, based on drag-and-drop and wire mechanisms, that allow to perform the composition approach, $Publishing$, and $Launching$. Such tools are responsible for hiding the data mapping among WR, NMR, and OpR. The data mapping is a problem particularly daunting for Network Administrators, because, generally, they are not expert developers. As the Mashment Maker is devoted to Network Administrators, we define a simple process to tackle whatever NMSit: (*i*) *conduct the approach of composition* above described and so build up Mashments for the NMSit or *buy the Mashment* for the NMSit, (*ii*) *use the Mashments* to deal the NMSit; and (*iii*) *maintenance of Mashments* to avoid their malfunctioning.

The Mashment Marketplace allows to establish a new value chain in which revenues are shared not only by WRP and NMRP but all stakeholders. Marketplaces that involve end-users (as Network Administrators) and professional developers (as Mashup Creators) have proved valuable to promote the evolution over time of Service Ecosystems (as the Mashment Ecosystem). The Android Market and the Apple Store are succesful examples of solutions marketplaces. The Mashment Marketplace is make up of the Mashment Store(s) and the Mashment Repository(s). In the Mashment Store are performed

$Selling$, $Sharing$, and $Buying$. As result of $Announce$, in the Mashment Respositoy are stored MashmentPkgs (a packaged Mashment) to be sold or shared. The reuse of existing Mashments, by $Selling$ and $Sharing$, is a key aspect for the evolution of proposed ecosystem.

The Mashment Engine is responsible for the lifecycle (*i.e.*, $Instantiate$) of Mashment Instances that are Mashments on run time. A Mashment Instance allows to tackle (*i.e.*, $Tackling$) a NMSit. As the Mashments are make up of WR, NMR, OpR, and even Mashments, that function in back-end or front-end, the Mashment Engine is an enabler to create, destroy, and cache such resources into both web servers and web/mobile clients.

IV. Case Study

To assess our proposal, we perform a test environment for the Mashment Ecosystem. This section describes the test conditions and analyzes the obtained results.

A. Test Environment

The Figure 2 depicts the test environment for the raised case study. To set up such environment, first, we created a heterogeneous virtual network. Second, we developed the Mashment Maker, the Mashment Engine, and the Marketplace Repository (as a Database). Third, we released into the Mashment Maker the resources to create, launch, and publish a Mashment, hereinafter called MVN. When suddenly and unidentified transmission errors occur (*i.e.*, the NMSit) in the virtual network built, MVN allows the Network Administrator to visually look for them (*i.e.*, tackling the NMSit) by presenting, in a visual and integrated way, the collected, correlated, and fused information about packet traffic from switches.

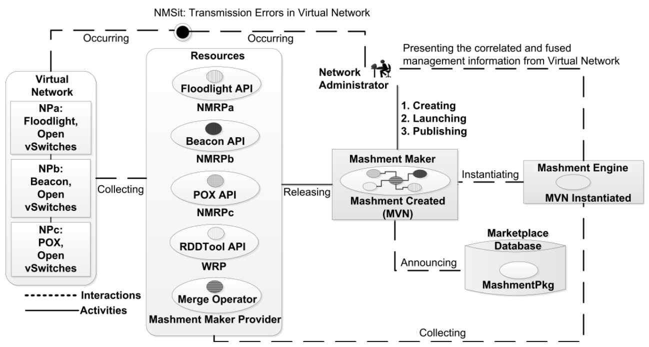


Fig. 2: Test Environment

The virtual network was created by using Open vSwitch and one different OpenFlow Controller in each network provider (NP_a, NP_b, and NP_c), namely, Floodlight, Beacon, and POX. Beacon and Floodlight are controllers based on the Java programming language. In turn, POX is based on Python. The controllers and switches were deployed on the Mininet that is a software for emulating OpenFlow networks. The Mininet was executed on Oracle VM VirtualBox. The Mashment Maker was developed by using Asynchronous Javascript and XML (AJAX), web services based on the Representational State Transfer (REST), and APIs for Floodlight, Beacon, POX, and RRDTool. The Mashment Maker was deployed on the Apache Tomcat Server. This Apache Tomcat was used as the Mashment Engine in the server-side. In the client-side, the Mashment Engine used was Firefox. The Marketplace Database was implemented on a MySQL Server. Network Administrators created the MVN by using the Mashment Maker.

B. Evaluation and Analysis

To evaluate the Mashment Ecosystem, we estimated and experimentally measured the time that Network Administrators take to perform $Creating$, $Launching$, and $Publishing$ MVN. MVN (see Figure 3) is formed by five visual components (*i.e.*, R_{used}): BeaconController, FloodlightController, and POXController representing the OpenFlow controllers (*i.e.*, NMR) used in the virtual network, RRDTool (*i.e.*, WR) representing the web tool used to generate images that present packet traffic information, and Monitoring Panel that is a merge operator (*i.e.*, OpR) and the root resource (*i.e.*, r_{root}). Subsequently, we also estimated and experimentally measured the time that Network Administrators take to retrieve, by using MVN, information about the NMSit above presented.

The time estimation was made by using the Keystroke-Level Model (KLM). In KLM, each activity is modeled as a sequence of actions. The time average for KLM actions is [21]: (*i*) Press and release a key $\rightarrow K = 0.2s$, (*ii*) Type a string $\rightarrow T_n = n * K$, (*iii*) Hold or release the mouse $\rightarrow B = 0.1s$, (*iv*) Point the mouse $\rightarrow P = 1.1s$, (*v*) Move the hand from mouse to keyboard or viceversa $\rightarrow H = 0.4s$; and (*vi*) Mental preparation $\rightarrow M = 1.35s$. In addition to these actions, we used, drag-and-drop a visual element $\rightarrow T_{dnd}$ and wire two visual elements $\rightarrow T_{wire}$. T_{dnd} and T_{wire} are given by [4]: $T_{dnd} = P + 2B = 1.3s$ and $T_{wire} = 3P + 8B = 4.1s$.

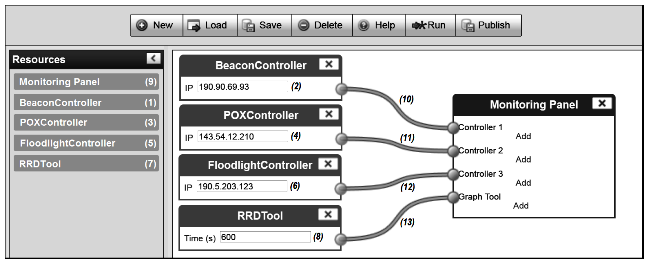


Fig. 3: Mashment Maker - Creating MVN

To tackle the NMSit, the Network Administrator creates MVN, the corresponding actions sequence (*i.e.*, δ) is as follows (see Figure 3): ***(1)*** Drag-and-drop Beacon $\rightarrow T_{dnd}$, ***(2)*** Configure Beacon (IP Address=190.90.69.93) $\rightarrow T_{con12}$, ***(3)*** Drag-and-drop POX $\rightarrow T_{dnd}$, ***(4)*** Configure POX (IP Address=143.54.12.210) $\rightarrow T_{con13}$, ***(5)*** Drag-and-drop Floodlight $\rightarrow T_{dnd}$, ***(6)*** Configure Floodlight (IP Address=190.5.203.123) $\rightarrow T_{con13}$, ***(7)*** Drag-and-drop RRDTool $\rightarrow T_{dnd}$, ***(8)*** Configure RRDTool (Time in seconds = 600) $\rightarrow T_{rrd}$, ***(9)*** Drag-and-drop Monitoring Panel $\rightarrow T_{dnd}$, ***(10)*** Wire Beacon to Monitoring Panel $\rightarrow T_{wire}$, ***(11)*** Wire POX to Monitoring Panel $\rightarrow T_{wire}$, ***(12)*** Wire Floodlight to Monitoring Panel $\rightarrow T_{wire}$; and ***(13)*** Wire RRDTool to Monitoring Panel $\rightarrow T_{wire}$. Where, $T_{con38} = P + 2H + T_{n=38} = 8.8s$ and $T_{rrd} = P + 2H + T_{n=3} = 2.5s$.

According the previous sequence, the estimated time for $Creating$ MVN is $\mathbf{C_{est} = 13M + T_{con38} + T_{rrd} + 5T_{dnd} + 4T_{wire}}$. Then, it is expected that, by using the Mashment Maker, Network Administrators take $56.25s$ to build up MVN. We consider this $\mathbf{C_{est}}$ is good because, for instance, just typing the example script (10 lines with 40 characters each one) to generate a single RRD image takes $T_{script} = 10M + T_{n=400} = 93.5s$. In this way, we can state that Network Administrators, participating in the proposed ecosystem, can rapidly create Mashments aimed to tackle a NMSit. Generalizing, the estimated time to create any Mashment can be expressed as: $\delta(t) = T_{sel} + T_{conn} + T_{conf} + T_{ment}$. Where, $T_{sel} = \sum_1^i T_{dnd}$, $T_{conn} = \sum_1^j T_{wire}$, $T_{conf} = P + H + (nt * K)$, and $T_{ment} = M * (i + j + o)$. Here, i is the number of elements in the set R_{used} (drag-and-dropped), j is the number total of wires linking R_{used}, nt is the number total of characters to configure R_{used}, and o is the number of R_{used} to be configurated. In $\delta(t)$ a dynamic composition model could be used to decrease both $Tsel$ and T_{conn}. This dynamic composition for Mashments is a future work.

The Network Administrator performs the following actions sequence for $Launching$ MVN. This sequence is the same for every Mashment: ***(1)*** Drag-and-drop MVN (when a Mashment is created, it is represented as a resource in the Mashment Maker) $\rightarrow T_{dnd}$, ***(2)*** Point the mouse to Run button $\rightarrow P$; and ***(3)*** Press and release Run button $\rightarrow 2B$. Therefore, the estimated time for $Launching$ is given by $\mathbf{L_{est} = 3M + P + T_{dnd} + 2B}$. As result, it is expected that Network Administrators take $6.65s$ to start MVN by means of the Mashment Maker.

The Network Administrator performs the following actions sequence for $Publishing$ MVN. This sequence is the same for every Mashment. ***(1)*** Point the mouse to Save button $\rightarrow P$, ***(2)*** Point the mouse to dialog that asks the Mashment name $\rightarrow P$, ***(3)*** Type the string MVN $\rightarrow T_{n=3}$, ***(4)*** Mouse press and release to store MVN in the Mashment Maker $\rightarrow 2B$, ***(5)*** Point the mouse to Publish button $\rightarrow P$, ***(6)*** Point the mouse to dialog that asks the Marketplace location $\rightarrow P$, ***(7)*** Type a repository string, for instance, http://www.mashments.mplace.com/repos $\rightarrow T_{n=37}$; and ***(8)*** Mouse press and release to store MVN in the Marketplace Database $\rightarrow 2B$. Therefore, the time estimated for the $Publishing$ activity is given by $\mathbf{P_{est} = 8M + 4(P + B) + T_{n=3} + T_{n=37}}$. As result, it is expected that the Network Administrator takes $23.60s$ to publish any Mashment. Afterwards, MVN and, in general, any Mashments can be shared, sold, and purchased in the Marketplace Store that will be presented in a future work.

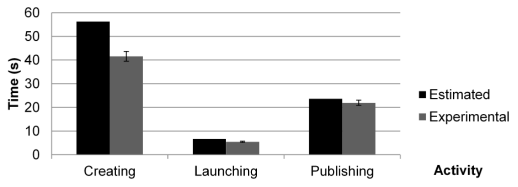
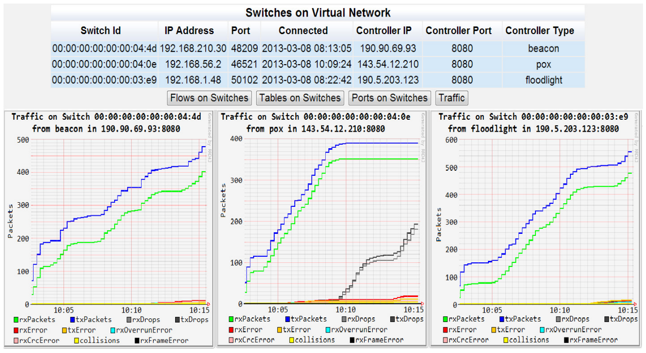


Fig. 4: Activities Time: Estimated vs Experimental

We also conducted an experimental study to measure the time that Network Administrators take to perform $Creating$, $Launching$, and $Publishing$. In the study participated 30 Network Administrators whose age ranged from 22 to 35. Although all participants frequently had used web tools none of them had used a mashup maker before. Thus, each participant was trained to use the Mashment Maker by 45 minutes. We took the experimental average time in seconds with a 95% confidence level. The results of estimated and experimental times (see Figure 4) corroborate the short time that Network Administrators need to tackle a NMSit by performing activities in the proposed ecosystem. Furthermore, as the experimental times of $Creating$ ($41.55s$), $Launching$ ($5.46s$), and $Publishing$ ($21.92s$) were always less than the corresponding estimated times, we can state that the implementation of proposed ecosystem had a good behavior in front of KLM estimations.


Switches on Virtual Network

Switch Id	IP Address	Port	Connected	Controller IP	Controller Port	Controller Type
00:00:00:00:00:00:04:4d	192.168.210.30	48209	2013-03-08 08:13:05	190.90.69.93	8080	beacon
00:00:00:00:00:00:04:0e	192.168.56.2	46521	2013-03-08 10:09:24	143.54.12.210	8080	pox
00:00:00:00:00:00:03:e9	192.168.1.48	50102	2013-03-08 08:22:42	190.5.203.123	8080	floodlight

Flows on Switches
Tables on Switches
Ports on Switches
Traffic
Traffic on Switch 00:00:00:00:00:00:04:4d from beacon in 190.90.69.93:8080
Traffic on Switch 00:00:00:00:00:00:04:0e from pox in 143.54.12.210:8080
Traffic on Switch 00:00:00:00:00:00:03:e9 from floodlight in 190.5.203.123:8080
Packets
10:05
10:10
10:15
rxPackets
txPackets
rxDrops
txDrops
rxError
txError
rxOverrunError
rxCrcError
collisions
rxFrameError

Fig. 5: MVN on runtime

On runtime, MVN (see Figure 5) allows Network Administrators to tackle the raised NMSit. MVN during its execution presents, simultaneously in an integrated user interface, the information of flows, links, and packect traffic of Open vSwitches, regardless of controllers from network providers. For instance, into MVN, the actions sequence to retrieve packet traffic information of three switches, each one in a different controller, is as follows: ***(1)*** Point the mouse to controllers list $\rightarrow P$, ***(2)*** Mouse press and release to select three controllers $\rightarrow 6B$; ***(3)*** Point the mouse to Switches button $\rightarrow P$, ***(4)*** Mouse press and release the Switches button $\rightarrow 2B$, ***(5)*** Mouse press and release to select three switches $\rightarrow 6B$, ***(6)*** Point the mouse to Traffic button $\rightarrow P$, ***(7)*** Mouse press and release the Traffic button to open the RRDTool images that contain the packet traffic information $\rightarrow 2B$. The estimated time to the above sequence is given by $\mathbf{R_{est} = 7M + 3P + 16B}$. Thus, it is expected that, by using MVN, Network Admstrators take $14.35s$ to deal the raised NMSit, by analyzing, in an integrated user interface, three RRDTool images that present information about packets received, transmitted, dropped, and with error. This result was corroborated by the experimental study in which $\mathbf{R_{exp} = 9.01s < R_{est}}$.

If the Network Administrator does not participate in the Mashment Ecosystem, he/she performs the following actions sequence to retrieve the information about the packet traffic on one switch from a specific controller web tool: ***(1)*** Point the mouse to Switches tab $\rightarrow P$, ***(2)*** Mouse press and release to select the Switches tab $\rightarrow 2B$, ***(3)*** Point the mouse to select a switch $\rightarrow P$, ***(4)*** Mouse press and release to select a switch $\rightarrow 2B$, ***(5)*** Point the mouse to Ports button $\rightarrow P$; and ***(6)***

Mouse press and release to select ports of switch $\rightarrow 2B$. These actions must be repeated three times, one by each controller web tool. Therefore, without MVN the estimated time to retrieve non-integrated information about the packet traffic on three switches is: $\mathbf{R_{3s} = 3(6M + 6B + 3P) = 36s}$. Therefore, $\mathbf{R_{exp}} < \mathbf{R_{est}} < \mathbf{R_{3s}}$. In this sense, it is important to highlight that the retrieving time for MVN is significantly smaller (a 60% taking into account the estimated time) than for the non-MVN case. According this result, we can state that a Network Administrator in the Mashment Ecosystem can tackle a NMSit faster than one out of it.

V. Conclusions and future work

In this paper, we introduced a mashup ecosystem (Mashment Ecosystem) that allows to tackle network management situations (NMSit). The Mashment Ecosystem and its implementation are based on the high-level abstraction of NMR, WR, and OpR, the composition model of mashups, and an approach centered in the Network Administrator for building up of composite solutions. Our ecosystem empowers the Network Administrator with the important ability to rapidly create, launch, and publish Mashments that are mashups devised to collect, correlate, fuse, and present integrated information about a NMSit. We also presented experimental measured and KLM estimation of time that the Network Administrator take to: (*i*) create, launch, and publish MVN that is a Mashment aimed to address a specific NMSit: transmission errors in a heterogeneous virtual network; and (*ii*) retrieve, by using MVN, the integrated information about the referred NMSit.

The aforementioned NMSit has a particular challenge: it needs the fast development of a solution (MVN) able to retrieve, merge, and rapidly present, in an integrated way, network management information from different OpenFlow controllers and their underlying virtual network elements. The Mashment Ecosystem allowed the Network Administrator to overcome such a challenge, corroborating its significance and the relevance of Mashment concept. Through an experimental and KLM evaluation, we have confirmed, first, the short time that a Network Administrator takes to MVN: create (estimated=$56.25s$, experimental=67.24), launch (estimated=$6.65s$, experimental=$5.46s$), and publish (estimated=$23.60s$, experimental=21.92). Second, the short time that a Network Administrator, that is using MVN, takes to retrieve (estimated=$14.35s$, experimental=$9.01s$) the integrated information about the raised NMSit. The experimental evaluation confirmed KLM predictions and, consequently, the feasibility of using our ecosystem to tackle any NMSit. Furthermore, it is important to highlight that the time to retrieve non-integrated information about this NMSit, by using mistmached solutions, is $36s$. Thus, it is expected that a Network Administrator in the Mashment Ecosystem can tackle a NMSit 60% faster than one out of it.

As future work, we plan to propose and implement a Mashment dynamic composition model in order to tackle the NMSits more rapidly. Furthermore, we are interested in evaluating the productivity of Network Administrators participating in the Mashment Ecosystem. We also plan to implement the Mashment Marketplace to evaluate its feasibility. The acceptance of Mashments by Network Administrators is a topic to explore too.

Acknowledgment

The research of PhD(c) Caicedo is funded by two scholarships one PECPG of the CAPES (Brazil) and another one of the University of Cauca (Colombia).

References

[1] Z. Zhao, S. Bhattarai, J. Liu, and N. Crespi, "Mashup services to daily activities: end-user perspective in designing a consumer mashups," in *iiWAS '11*. New York, NY, USA: ACM, 2011, pp. 222–229.

[2] G. Jakobson, J. Buford, and L. Lewis, "Situation Management: Basic Concepts and Approaches," in *Information Fusion and Geographic Information Systems*, ser. Lecture Notes in Geoinformation and Cartography, W. Cartwright, G. Gartner, L. Meng, M. . Peterson, V. . Popovich, M. Schrenk, and K. . Korolenko, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, ch. 2, pp. 18–33.

[3] G. Bader, W. He, A. Anjomshoaa, and A. Tjoa, "Proposing a context-aware enterprise mashup readiness assessment framework," *Information Technology and Management*, vol. 13, pp. 377–387, 2012.

[4] S. Tian, G. Weber, and C. Lutteroth, "A tuplespace event model for mashups," in *OzCHI '11*. New York, NY, USA: ACM, 2011, pp. 281–290.

[5] G. Jakobson, L. Lewis, C. Matheus, M. Kokar, and J. Buford, "Overview of situation management at sima 2005," in *MILCOM '05. IEEE*, 2005, pp. 1630 –1636 Vol. 3.

[6] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: Design, implementation and experience," *Parallel Computing*, vol. 30, p. 2004, 2003.

[7] W. Barth, *Nagios: System and Network Monitoring*, 2nd ed. San Francisco, CA, USA: No Starch Press, 2008.

[8] G. Pavlou, "On the evolution of management approaches, frameworks and protocols: A historical perspective," *J. Netw. Syst. Manage.*, vol. 15, no. 4, pp. 425–445, Dec. 2007.

[9] M. Bari, R. Boutaba, R. Esteves, L. Granville, M. Podlesny, M. Rabbani, Q. Zhang, and M. Zhani, "Data center network virtualization: A survey," *Communications Surveys Tutorials, IEEE*, vol. PP, no. 99, pp. 1–20.

[10] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[11] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, apr 2010.

[12] E. M. Maximilien, A. Ranabahu, and S. Tai, "Swashup: situational web applications mashups," in *OOPSLA '07*. New York, NY, USA: ACM, 2007, pp. 797–798.

[13] N. Ozkan and W. Abidin, "Investigation of mashups for managers," in *ISCIS 2009*, sept. 2009, pp. 622 –627.

[14] A. Majchrzak and P. H. B. More, "Emergency! web 2.0 to the rescue!" *Commun. ACM*, vol. 54, pp. 125–132, April 2011.

[15] E. Tosti and W. Smari, "Sensors integration in a grid-based architecture for emergency management systems," in *DEST '10*, April, pp. 435–442.

[16] J. Adams and C. Reynolds, "A complex situational management application employing expert systems," in *Systems, Man, and Cybernetics, 2000. IEEE*, vol. 3, 2000, pp. 1959 –1964 vol.3.

[17] R. Koelle and A. Tarter, "Towards a distributed situation management capability for sesar and nextgen," in *ICNS '12*, april 2012, pp. O6–1 –O6–12.

[18] B. Magoutas, G. Mentzas, and D. Apostolou, "Proactive situation management in the future internet: The case of the smart power grid," in *DEXA '11*, 29 2011-sept. 2 2011, pp. 267 –271.

[19] K. Huang, Y. Fan, and W. Tan, "An empirical study of programmable web: A network analysis on a service-mashup system," in *ICWS '12*, june 2012, pp. 552 –559.

[20] R. Latih, A. Patel, A. Zin, T. Yiqi, and S. Muhammad, "Whip: A framework for mashup development with block-based development approach," in *ICEEI '11*, july 2011, pp. 1 –6.

[21] D. Kieras, "Using the keystroke-level model to estimate execution times," in *University of Michigan*, 2001.

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

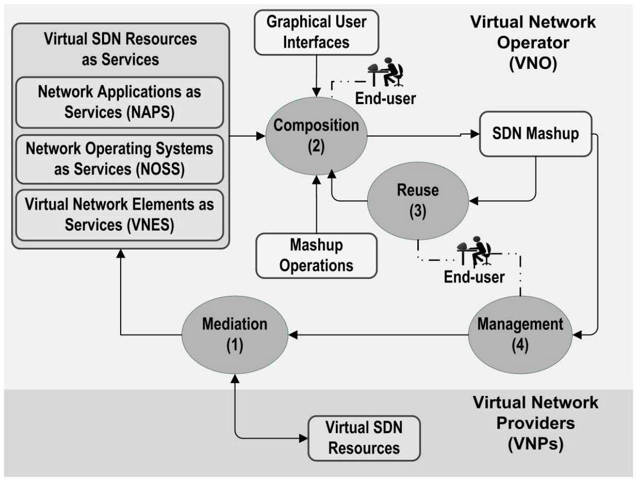

Virtual SDN Resources as Services
Network Applications as Services (NAPS)
Network Operating Systems as Services (NOSS)
Virtual Network Elements as Services (VNES)
Graphical User Interfaces
End-user
Virtual Network Operator (VNO)
Composition (2)
SDN Mashup
Reuse (3)
Mashup Operations
End-user
Mediation (1)
Management (4)
Virtual SDN Resources
Virtual Network Providers (VNPs)

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.

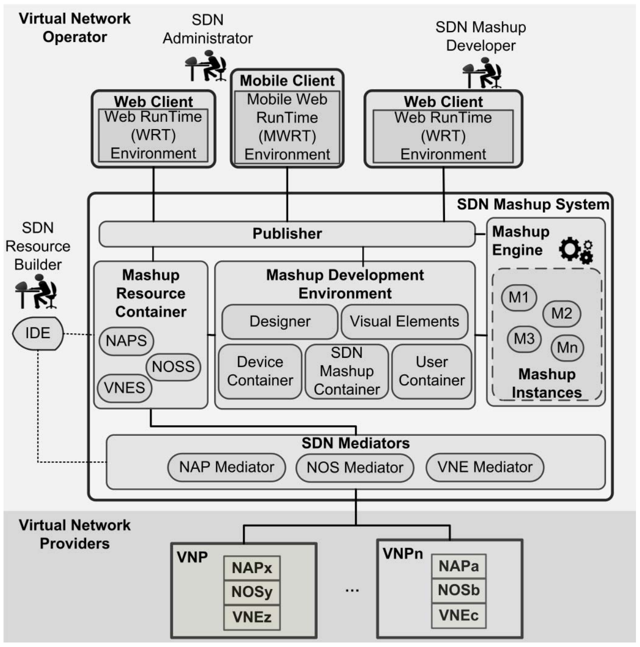


Figure 2. SDN Mashup System

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].

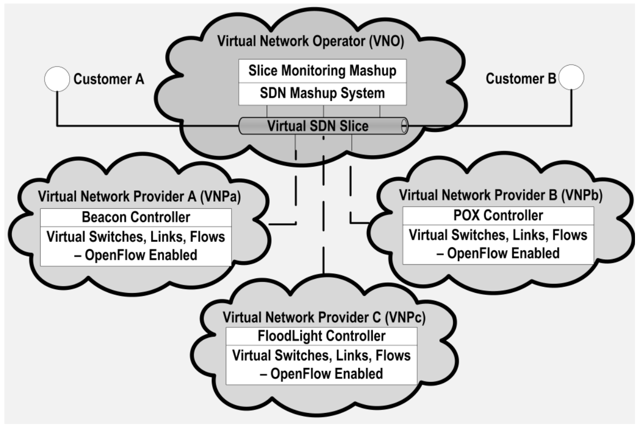

Virtual Network Operator (VNO)
Slice Monitoring Mashup
SDN Mashup System
Customer A
Customer B
Virtual SDN Slice
Virtual Network Provider A (VNPa)
Beacon Controller
Virtual Switches, Links, Flows – OpenFlow Enabled
Virtual Network Provider B (VNPb)
POX Controller
Virtual Switches, Links, Flows – OpenFlow Enabled
Virtual Network Provider C (VNPc)
FloodLight Controller
Virtual Switches, Links, Flows – OpenFlow Enabled

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.

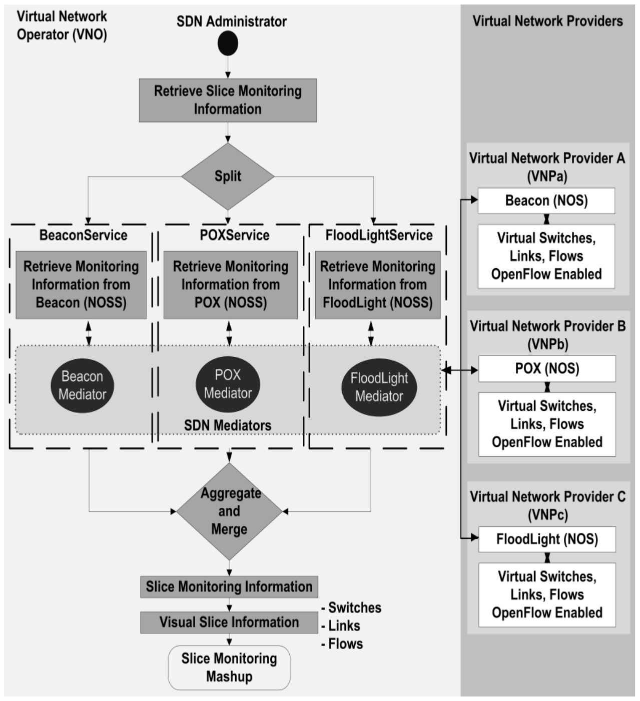
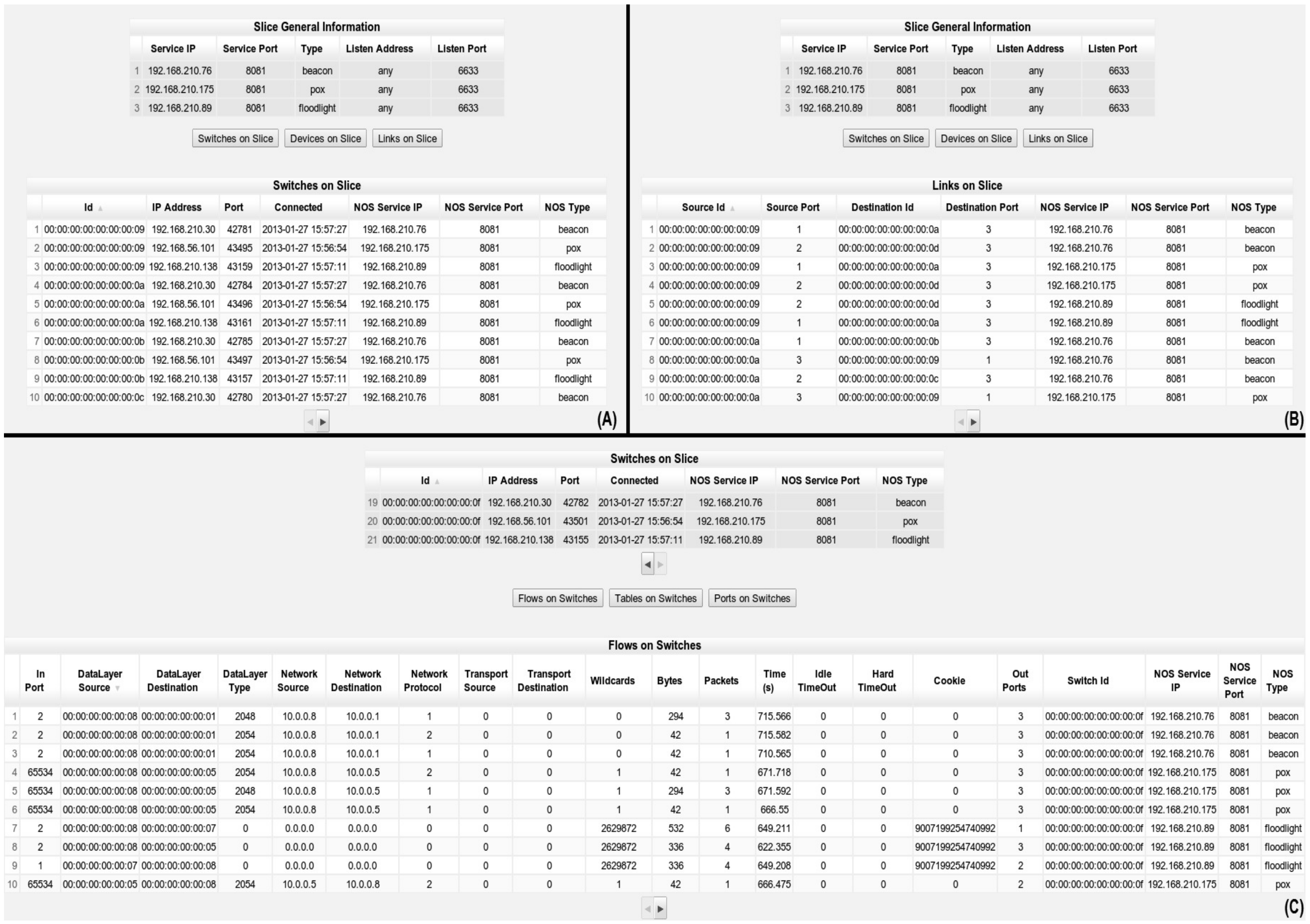

Virtual Network Operator (VNO)
SDN Administrator
Virtual Network Providers
Retrieve Slice Monitoring Information
Split
BeaconService
POXService
FloodLightService
Retrieve Monitoring Information from Beacon (NOSS)
Retrieve Monitoring Information from POX (NOSS)
Retrieve Monitoring Information from FloodLight (NOSS)
Beacon Mediator
POX Mediator
FloodLight Mediator
SDN Mediators
Aggregate and Merge
Slice Monitoring Information
Visual Slice Information
- Switches
- Links
- Flows
Slice Monitoring Mashup
Virtual Network Provider A (VNPa)
Beacon (NOS)
Virtual Switches, Links, Flows OpenFlow Enabled
Virtual Network Provider B (VNPb)
POX (NOS)
Virtual Switches, Links, Flows OpenFlow Enabled
Virtual Network Provider C (VNPc)
FloodLight (NOS)
Virtual Switches, Links, Flows OpenFlow Enabled

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



Slice General Information

	Service IP	Service Port	Type	Listen Address	Listen Port
1	192.168.210.76	8081	beacon	any	6633
2	192.168.210.175	8081	pox	any	6633
3	192.168.210.89	8081	floodlight	any	6633

Switches on Slice

	Id	IP Address	Port	Connected	NOS Service IP	NOS Service Port	NOS Type
1	00:00:00:00:00:00:00:09	192.168.210.30	42781	2013-01-27 15:57:27	192.168.210.76	8081	beacon
2	00:00:00:00:00:00:00:09	192.168.56.101	43495	2013-01-27 15:56:54	192.168.210.175	8081	pox
3	00:00:00:00:00:00:00:09	192.168.210.138	43159	2013-01-27 15:57:11	192.168.210.89	8081	floodlight
4	00:00:00:00:00:00:00:0a	192.168.210.30	42784	2013-01-27 15:57:27	192.168.210.76	8081	beacon
5	00:00:00:00:00:00:00:0a	192.168.56.101	43496	2013-01-27 15:56:54	192.168.210.175	8081	pox
6	00:00:00:00:00:00:00:0a	192.168.210.138	43161	2013-01-27 15:57:11	192.168.210.89	8081	floodlight
7	00:00:00:00:00:00:00:0b	192.168.210.30	42785	2013-01-27 15:57:27	192.168.210.76	8081	beacon
8	00:00:00:00:00:00:00:0b	192.168.56.101	43497	2013-01-27 15:56:54	192.168.210.175	8081	pox
9	00:00:00:00:00:00:00:0b	192.168.210.138	43157	2013-01-27 15:57:11	192.168.210.89	8081	floodlight
10	00:00:00:00:00:00:00:0c	192.168.210.30	42780	2013-01-27 15:57:27	192.168.210.76	8081	beacon

Slice General Information

	Service IP	Service Port	Type	Listen Address	Listen Port
1	192.168.210.76	8081	beacon	any	6633
2	192.168.210.175	8081	pox	any	6633
3	192.168.210.89	8081	floodlight	any	6633

Links on Slice

	Source Id	Source Port	Destination Id	Destination Port	NOS Service IP	NOS Service Port	NOS Type
1	00:00:00:00:00:00:00:09	1	00:00:00:00:00:00:00:0a	3	192.168.210.76	8081	beacon
2	00:00:00:00:00:00:00:09	2	00:00:00:00:00:00:00:0d	3	192.168.210.76	8081	beacon
3	00:00:00:00:00:00:00:09	1	00:00:00:00:00:00:00:0a	3	192.168.210.175	8081	pox
4	00:00:00:00:00:00:00:09	2	00:00:00:00:00:00:00:0d	3	192.168.210.175	8081	pox
5	00:00:00:00:00:00:00:09	2	00:00:00:00:00:00:00:0d	3	192.168.210.89	8081	floodlight
6	00:00:00:00:00:00:00:09	1	00:00:00:00:00:00:00:0a	3	192.168.210.89	8081	floodlight
7	00:00:00:00:00:00:00:0a	1	00:00:00:00:00:00:00:0b	3	192.168.210.76	8081	beacon
8	00:00:00:00:00:00:00:0a	3	00:00:00:00:00:00:00:09	1	192.168.210.76	8081	beacon
9	00:00:00:00:00:00:00:0a	2	00:00:00:00:00:00:00:0c	3	192.168.210.76	8081	beacon
10	00:00:00:00:00:00:00:0a	3	00:00:00:00:00:00:00:09	1	192.168.210.175	8081	pox

Switches on Slice

	Id	IP Address	Port	Connected	NOS Service IP	NOS Service Port	NOS Type
19	00:00:00:00:00:00:00:0f	192.168.210.30	42782	2013-01-27 15:57:27	192.168.210.76	8081	beacon
20	00:00:00:00:00:00:00:0f	192.168.56.101	43501	2013-01-27 15:56:54	192.168.210.175	8081	pox
21	00:00:00:00:00:00:00:0f	192.168.210.138	43155	2013-01-27 15:57:11	192.168.210.89	8081	floodlight

Flows on Switches

	In Port	DataLayer Source	DataLayer Destination	DataLayer Type	Network Source	Network Destination	Network Protocol	Transport Source	Transport Destination	Wildcards
1	2	00:00:00:00:00:08	00:00:00:00:00:01	2048	10.0.0.8	10.0.0.1	1	0	0	0
2	2	00:00:00:00:00:08	00:00:00:00:00:01	2054	10.0.0.8	10.0.0.1	2	0	0	0
3	2	00:00:00:00:00:08	00:00:00:00:00:01	2054	10.0.0.8	10.0.0.1	1	0	0	0
4	65534	00:00:00:00:00:08	00:00:00:00:00:05	2054	10.0.0.8	10.0.0.5	2	0	0	1
5	65534	00:00:00:00:00:08	00:00:00:00:00:05	2048	10.0.0.8	10.0.0.5	1	0	0	1
6	65534	00:00:00:00:00:08	00:00:00:00:00:05	2054	10.0.0.8	10.0.0.5	1	0	0	1
7	2	00:00:00:00:00:08	00:00:00:00:00:07	0	0.0.0.0	0.0.0.0	0	0	0	2629872
8	2	00:00:00:00:00:08	00:00:00:00:00:05	0	0.0.0.0	0.0.0.0	0	0	0	2629872
9	1	00:00:00:00:00:07	00:00:00:00:00:08	0	0.0.0.0	0.0.0.0	0	0	0	2629872
10	65534	00:00:00:00:00:05	00:00:00:00:00:08	2054	10.0.0.5	10.0.0.8	2	0	0	1

	Bytes	Packets	Time (s)	Idle TimeOut	Hard TimeOut	Cookie	Out Ports	Switch Id	NOS Service IP	NOS Service Port	NOS Type
1	294	3	715.566	0	0	0	3	00:00:00:00:00:00:00:0f	192.168.210.76	8081	beacon
2	42	1	715.582	0	0	0	3	00:00:00:00:00:00:00:0f	192.168.210.76	8081	beacon
3	42	1	710.565	0	0	0	3	00:00:00:00:00:00:00:0f	192.168.210.76	8081	beacon
4	42	1	671.718	0	0	0	3	00:00:00:00:00:00:00:0f	192.168.210.175	8081	pox
5	294	3	671.592	0	0	0	3	00:00:00:00:00:00:00:0f	192.168.210.175	8081	pox
6	42	1	666.55	0	0	0	3	00:00:00:00:00:00:00:0f	192.168.210.175	8081	pox
7	532	6	649.211	0	0	9007199254740992	1	00:00:00:00:00:00:00:0f	192.168.210.89	8081	floodlight
8	336	4	622.355	0	0	9007199254740992	3	00:00:00:00:00:00:00:0f	192.168.210.89	8081	floodlight
9	336	4	649.208	0	0	9007199254740992	2	00:00:00:00:00:00:00:0f	192.168.210.89	8081	floodlight
10	42	1	666.475	0	0	0	2	00:00:00:00:00:00:00:0f	192.168.210.175	8081	pox

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 Beacon-Controller/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 FloodLight-Service are based on the REST architectural model [21]. We have used REST because it is becoming in the de-facto 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.

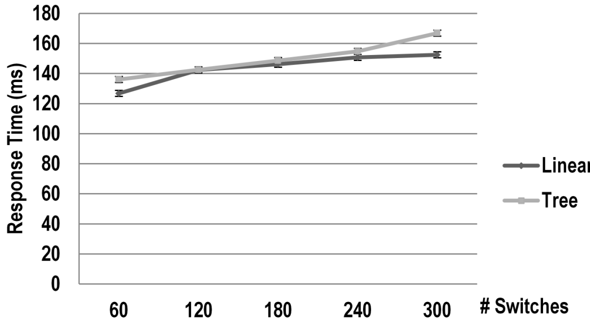


Figure 6. Slice Monitoring Mashup - SwitchesList

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.

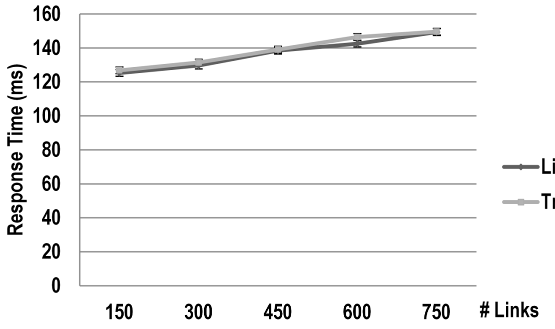


Figure 7. Slice Monitoring Mashup - LinksList

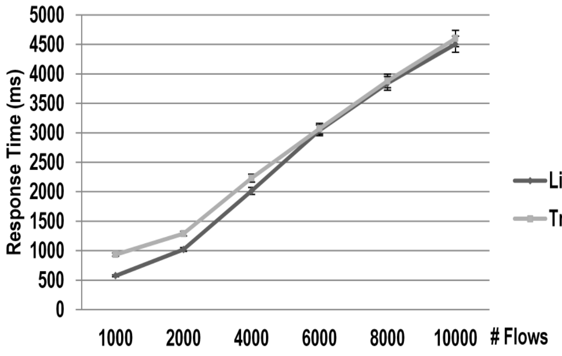


Figure 8. Slice Monitoring Mashup - FLowsList

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.

A CIM-based Information Model for Heterogeneous SDN Management

Felipe Estrada-Solano[a,*], Armando Ordonez[b],
Lisandro Zambenedetti Granville[c], Oscar Mauricio Caicedo Rendon[a]

[a]*Telematics Engineering Group, Telematics Department, University of Cauca, Calle 5 No. 4-70, Popayan, CA, Colombia*
[b]*Intelligent Management Systems Group, Foundation University of Popayan, Calle 5 No. 8-58, Popayan, CA, Colombia*
[c]*Computer Networks Group, Institute of Informatics, Federal University of Rio Grande do Sul, Av. Bento Gonçalves, 9500 Porto Alegre, RS, Brazil*

Abstract

The Software-Defined Networking paradigm establishes a three-plane architecture that facilitates the deployment of network functions and simplifies traditional network management tasks. However, this architecture lacks an integrated or standardized framework for managing the SDN itself. Some investigations have addressed such shortage by proposing different solutions that tackle specific management requirements for particular SDN technology instances. This isolated approach forces network administrators to use multiple frameworks to achieve a complete SDN management that is complex and time-consuming in heterogeneous environments. In this paper, we introduce an Information Model based on the Common Information Model that establishes a technology-agnostic and consistent characterization of the whole SDN architecture. Such Information Model represents the core towards building a Management Plane aimed to facilitate the integrated SDN management in heterogeneous environments. To test our Information Model, we developed a prototype and conducted a performance evaluation in an SDN configuration scenario that deploys different managing technologies. The obtained results

[*]Corresponding author. Tel.: +57 3185274311
Email addresses: `festradasolano@unicauca.edu.co` (Felipe Estrada-Solano), `jaordonez@fup.edu.co` (Armando Ordonez), `granville@inf.ufrgs.br` (Lisandro Zambenedetti Granville), `omcaicedo@unicauca.edu.co` (Oscar Mauricio Caicedo Rendon)

provide directions that corroborate the feasibility of our approach (in terms of time-response and network traffic) for configuring heterogeneous SDNs.

Keywords: Common Information Model, Software-Defined Networking, heterogeneous environments, resource characterization, configuration management.

1. Introduction

Over the past 20 years, programmable networks have evolved as a key driver to innovate and to cope with complexity and management in computer networks. Nowadays, Software-Defined Networking (SDN) paradigm is an attractive trend to program networks in both research and industry [1, 2]. From a high-level point of view, SDN separates control and forwarding planes, allowing to operate networks in a simpler way from a logically centralized software program often referred to as controller [3].

SDN standardization bodies (*e.g.*, Open Network Foundation [ONF] and Linux Foundation) and networking vendors (*e.g.*, Cisco and Juniper) describe a typical SDN architecture as three horizontal planes [4, 5]: *(i)* a lower Data Plane to forward packets, *(ii)* a middle Control Plane to compile decision policies and to enforce them on the Data Plane through Southbound Interfaces (SBI); and *(iii)* an upper Application Plane to orchestrate business functions and high-level services that manage network behavior using Northbound Interfaces (NBI) provided by the Control Plane. Additionally, the SDN architecture includes East/Westbound Interfaces (EWBI) to enable deploying a distributed Control Plane.

At the top of the SDN architecture, a lot of research has proposed services and applications to simplify traditional network management tasks, such as load-balancing [6], efficient energy usage [7], and access control [8, 9]. Furthermore, some studies have addressed the need to manage SDN itself, including, for example, frameworks to configure the Data Plane [10, 11], to deploy [12, 13] and monitor [14, 15] the Control Plane, to virtualize SDNs [16, 17], and to develop the Application Plane [18, 19]. However, to the best of our knowledge, no integrated solution exists to manage SDN as a whole by employing well-defined interfaces and supporting different deployment technologies.

The lack of frameworks for integrated network management forces network administrators to handle several isolated solutions to manage resources from distinct planes of the SDN architecture as well as various technology instances. Thus, SDN management remains complex and time-consuming because of the diversity of solutions. In our previous work, we evaluated the feasibility of using mashups to control and monitor heterogeneous SDN environments by composing management applications at the top of the SDN architecture [20, 21, 22, 23]. Still, there is the need to characterize SDN as a whole from the management perspective to accomplish a joint understanding in heterogeneous and distributed environments. Some approaches have addressed the formal representation of SDN [24, 25, 26, 27], nonetheless, they are focused exclusively on specific technologies or fall short in modeling the SDN environment.

Recent proposals have considered a Management Plane in the SDN architecture for carrying out Operation, Administration, and Maintenance (OAM) functions [4] [5]. These proposals expose a very high-level view of their management component. We argue that is needed to extend and detail such Management Plane aiming to facilitate integrated control and monitoring of heterogeneous SDNs. As a first critical step, this Management Plane requires an Information Model that homogenizes management data to achieve consistency among all OAM tasks. In this paper, we specify the SDN Management Plane using the four Open System Interconnection (OSI) submodels [28] (*i.e.*, Information, Organizational, Communication, and Functional). Then, we focus on introducing a novel Information Model that represents the SDN management environment as a Common Information Model (CIM) conceptual framework [29]. We preferred CIM over other information definition languages (*e.g.*, Structure of Management Information [SMI] [30] and Guidelines for the Definition of Managed Objects [GDMO] [31]) because its high expressiveness affords future robust semantic integration [32]. Leveraging CIM features, we provide a technology-independent and consistent model across distinct vendors and SDN instances. Furthermore, our Information Model establishes a shared abstraction of managed and managing SDN resources from Data, Control, and Application planes to achieve a complete SDN management. It is noteworthy that the proposed model provides concepts and artifacts that may complement or enhance the Information Model structured by ONF (*i.e.*, ONF-CIM[1]) [24].

[1]ONF-CIM is not based on the CIM specification defined by the Distributed Management Task Force (DMTF).

The main contributions presented in this paper are:

- An Information Model based on CIM that describes managed and managing SDN resources regardless of deploying technologies;
- An SDN management system prototype that is based on the above Information Model;
- The demonstration that our proposal is effectively feasible (in terms of time-response and network traffic) when managing an SDN deployed with heterogeneous technologies.

The remainder of this paper is organized as follows. In Section II, it is discussed both the background and related work. In Section III, we overview the proposed Management Plane. In Section IV, we introduce our CIM-based Information Model. In Section V, we present a case study used to evaluate the proposed approach. The paper concludes in Section VI.

2. Background

In this section, first, we present the SDN architecture. Second, we discuss the related work about the SDN management.

2.1. Software-Defined Networking architecture

Multiple standardization bodies, such as ONF and Linux Foundation, focus on encouraging and normalizing open SDN frameworks. Also, various private networking vendors, such as Cisco and Juniper, offer proprietary SDN deployments. In turn, several research efforts work on improving architectural aspects of SDN. These open, proprietary, and research SDN solutions establish a typical SDN architecture [4, 5] composed of three horizontal planes (*i.e.*, Data Plane, Control Plane, and Application Plane) and three interfaces (*i.e.*, SBI, NBI, and EWBI).

The Data Plane deploys the network infrastructure composed of interconnected hardware and software-based Network Devices (NetDev) that perform forwarding operations. A NetDev ranges from dumb switches to custom switches. A dumb switch merely carries out simple forwarding functions, such as Longest Prefix Match (LPM). For example, OpenFlow-Only switches [33] just forward packets regarding their flow tables that are updated by the Control Plane. A custom switch relies on programmable platforms (*e.g.*, OpenWrt and NetFPGA) to integrate more complex forwarding functions, such as Network Address Translation (NAT) and firewall. For example, Forwarding Elements (FE) in ForCES [34] include multiple associated Logical Functional Blocks (LFB) to carry out such forwarding functions. An LFB defines either a punctual action for handling packets or a configuration entity operated by the Control Plane.

The Control Plane enforces decision policies on the Data Plane through SBIs. Each SBI defines the set of instructions and the communication protocols to allow the interaction between components in the Control Plane and in the Data Plane. The OpenFlow protocol is the most well-known open standard SBI because its widespread usage by vendors and research [1]. Other SBI proposals are ForCES [34] and Protocol-Oblivious Forwarding (POF) [35].

The Control Plane comprises Network Slicers (NetSlicer) and Network Operative Systems (NOS). A NetSlicer divides the underlying network infrastructure into several isolated logical network instances (*a.k.a.* slices), assigning their control to multiple tenant NOSs. For example, FlowVisor [36] acts as an OpenFlow proxy between switches and controllers, redirecting messages according to specific slicing dimensions, such as bandwidth and forwarding tables. An NOS compiles the network logic for instructing the underlying Data Plane and provides generic services (*e.g.*, topology discovering and host tracking) and NBIs to the Application Plane, facilitating to add custom Network Applications (NetApp). OpenFlow Controllers [33] and ForCES Control Elements (CE) [34] are NOS instances. This architecture avoids considering the NetSlicer as a specific NOS in order to demarcate their functionality (*i.e.*, network virtualization versus decision making). In addition, some approaches may provide network virtualization as an NOS service for multiple tenant NetApps [37] [38].

As aforementioned, the Control Plane provides NBI to the Application Plane. An NBI encompasses common Application Programming Interfaces (API) based on protocols (*e.g.*, Floodlight REST API [39]), programming languages (*e.g.*, Pyretic [18] and Procera [19]), file systems (*e.g.*, YANC [40]), among others. The Control Plane also defines EWBIs to deploy distributed NOSs. For example, SDNi [41] and ForCES CE-CE interface [34].

The Application Plane consists of NetApps that deploy and orchestrate business logic and high-level network functions, such as routing policies and access control. NetApps run either locally or remotely regarding NOSs. Local NetApps prefer NBIs based on programming languages. Remote NetApps usually employ protocol-based APIs.

2.2. Software-Defined Networking management

Most SDN proposals have tackled traditional network management tasks by carrying out managing *functions in NetApps* at the Application Plane. For example, wildcard-based algorithms [6] to better redistribute traffic in SDN networks, ElasticTree [7] to efficiently provide energy for SDN components, and Resonance [8] and OpenRoads [9] to control access to SDN resources. However, *functions in NetApps* lack of mechanisms to deal with several management requirements from distinct SDN architectural planes, such as: *(i)* in the Data Plane, configure certain NetDevs to communicate with a preferred NOS, *(ii)* in the Control Plane, set up a NetSlicer to link NOSs to their corresponding virtual network instances; and *(iii)* in the Application Plane, modify business parameters to customize NetApps logic.

Some investigations have tackled the above gap by providing *isolated tools* that address specific management requirements for particular SDN technology instances. For example: *(i)* OpenFlow Management and Configuration Protocol (OF-CONFIG) [10] and Open vSwitch Database (OVSDB) [11] that define protocols to configure NetDevs, *(ii)* Kandoo [12] and HyperFlow [13] to scale and distribute NOSs, *(iii)* OpenFlow Management Infrastructure (OMNI) [14] and ROVIZ [15] that provide graphic interfaces to monitor NOSs, *(iv)* VeRTIGO [16] and ADVisor [17] to configure NetSlicers; and *(v)* Pyretic [18] and Procera [19] that supply development tools to build NetApps. Considering that heterogeneous SDNs deploy a variety of resources from multiple vendors and distinct technologies, it is to highlight that a classic solution based on using *isolated tools* to accomplish a complete SDN management is complex and time-consuming.

In our previous work, we assessed the feasibility of a *mashup-based* approach to allow network administrators to compose NetApps that control and monitor heterogeneous SDNs [20, 21, 22, 23]. In such *mashup-based* approach, management applications relied on services and middlewares developed and extended by builder actors. These builder actors realized their job according to deployed SDN technology and their own managing data models, constraining such management applications to operate only on particular SDN domains. Therefore, as an essential step, an SDN management solution requires an Information Model that establishes a shared characterization of the entire SDN to enable integrated management in heterogeneous environments.

Few approaches have defined *Information Models* to characterize the SDN managed environment: *(i)* ONF-CIM [24] defines a Core Information Model (CoreModel) [25] that uses the Unified Modeling Language (UML) to structure the forwarding functions of Data Plane resources, *(ii)* ForCES Network Abstraction Model [26] employs a building block approach to represent ForCES FEs; and *(iii)* CIM-SDN [27] proposes a CIM extension schema to model SDN resources. It is worth noting that these *Information Models* fall short in representing the whole SDN managed environment and are tied to specific SDN technology instances. ONF CoreModel and ForCES Abstraction Model describe only the Data Plane. The former was designed for OpenFlow and the latter for ForCES. CIM-SDN merely includes the main elements from the Data and the Control Planes. Although CIM-SDN is based on a technology-neutral model (*i.e.*, CIM), the extended schema is highly attached to the OpenFlow architecture. Finally, ONF-CIM simply includes CoreModel so far but provides a flexible environment that allows to expand and refine its structure as new insights emerge, such as the approach described in this paper.

Table 1 summarizes the targets and gaps in SDN management of the above-reviewed proposals. Unlike these proposals, we consider an SDN management approach based on a reference, technology-agnostic model of the whole SDN in order to achieve integrated management in heterogeneous SDN environments.

3. A Management Plane for SDN

To define our approach, we extend the Management Plane concept considered by early SDN proposals for covering OAM functions omitted and restricted in the traditional SDN architecture [4, 5]. For example, assigning the Data Plane resources to the corresponding control components and configuring the policies and Service Level Aggreements (SLA) of the Control and Application planes. Although the NOS may implement many of these OAM functions, flooding the Control Plane with a lot of management tasks may cause low network performance. Unlike the above proposals, our Management Plane aggregates components that facilitate the integrated management in heterogeneous SDN environments. To better explain our approach, we present a high-level overview of the Management Plane using the four OSI submodels: Information, Organizational, Communication, and Functional.

Table 1: Proposals on SDN management

		Requirements for SDN management			
References	**Approach**	**Managing the whole SDN**	**Integrated management**	**Heterogeneous environment**	**Information Model**
[6]–[9]	Functions in NetApps				
[10]–[19]	Isolated tools	✓			
[20]–[23]	Mashup based		✓	✓	
[24]–[27]	Information Models		✓		✓
-	This approach	✓	✓	✓	✓

3.1. Overview

Fig. 1 depicts our Management Plane. This plane is formed by the Data Repository, the Manager, Adapters, Management Interfaces, and Agents. The Data Repository holds the Resource Representation Model (RRM) and serves the Manager to store management instance data. RRM handles metadata to provide an abstract, technology-neutral characterization of SDN resources. The Manager orchestrates and deploys Management Services to carry out different SDN management functions. These Management Services expose user interfaces to allow interaction of Network Administrators. Adapters afford a protocol-agnostic communication between the Manager and Agents through well-defined Management Interfaces. Each Management Interface connects both corresponding Adapter and Agent. Agents situate on SDN resources to act on behalf of the Manager. The whole operation of the Management Plane is based on RRM to achieve an integrated and technology-independent SDN management.

It is important to highlight that we define the Management Plane by referencing the four OSI network management submodels [28]: *(i)* an Information Model to establish a shared abstraction of SDN resources, *(ii)* an Organizational Model to specify roles and collaboration forms, *(iii)* a Communication Model to delineate the exchange of management data; and *(iv)* a Functional Model to structure management requirements.

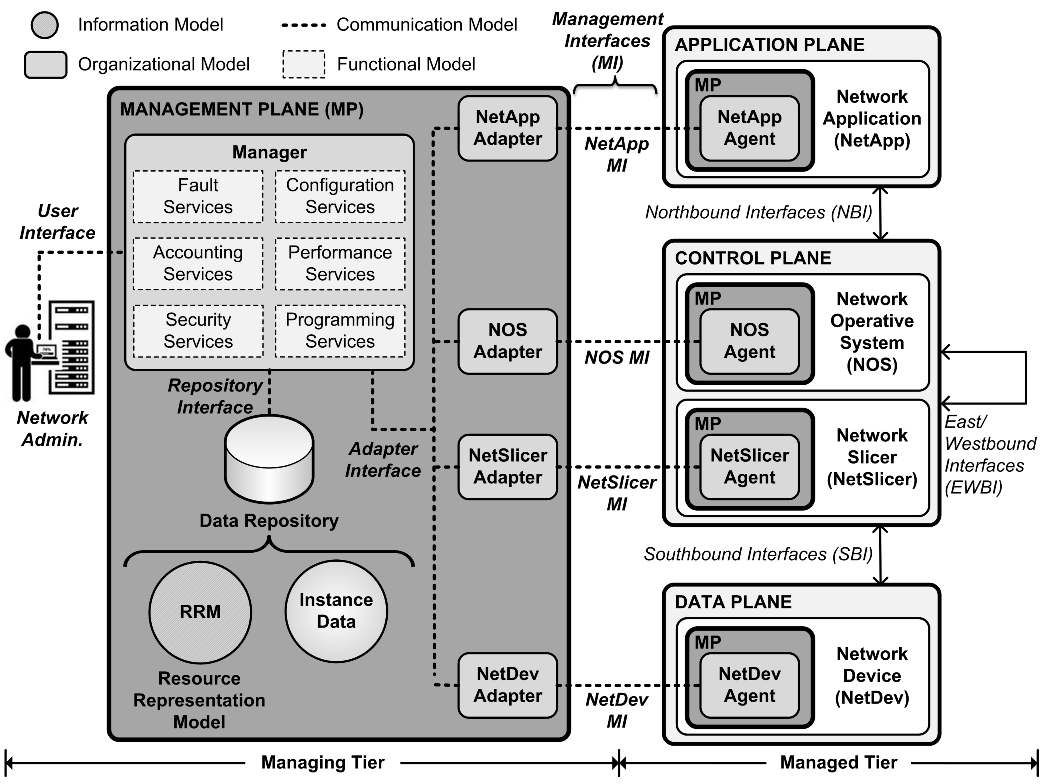


Figure 1: High-level SDN architecture with Management Plane

3.2. Submodels

Following we overview the four submodels of the proposed Management Plane.

3.2.1. Information Model

Our approach introduces a CIM model to describe the SDN management environment at a conceptual level regardless of deploying technologies. We use UML to graphically represent SDN resources and their relationships as CIM classes and associations, respectively. This object-oriented, well-understood abstract framework standardizes SDN management information across disparate vendors and SDN instances. Thus, enabling to carry out integrated management in heterogeneous SDNs. Further network designers may extend the proposed CIM model to include customized resource behavior. In our approach, the Information Model is realized by RRM in the Data

Repository. We focus on the details of the proposed Information Model in Section 4.

3.2.2. Organizational Model

We depict a two-tier like network management model that incorporates three kinds of entities (*a.k.a.* roles). A Managing Tier that encloses *manager* and *adapter* entities, and a Managed Tier that contains *agent* entities. A *manager* entity is responsible for: *(i)* housing and coordinating logic of management functions, *(ii)* providing user interaction with deployed management functions through tailored user interfaces (*e.g.*, command-line, graphical, and Web-based); and *(iii)* sending requests to and receiving replies and events from *agents* by means of *adapters*. An *adapter* entity allows a *manager* to interact with any specific *agent* by parsing data formats and protocols handled by their communication interfaces (*i.e.*, Adapter Interface and Management Interfaces). An *agent* entity resides on managed resources to carry out management requests delegated by a *manager*, such as performing an operation or responding to a query. In addition, an *agent* entity may dispatch unsolicited events to a *manager*. Each organizational component in our Management Plane gets the same name as its corresponding role. The Manager acts as a *manager* entity. NetApp Adapter, NOS Adapter, NetSlicer Adapter, and NetDev Adapter play an *adapter* role. NetApp Agent, NOS Agent, NetSlicer Agent, and NetDev Agent perform *agent* tasks. We differentiate Adapters and Agents regarding of SDN managed resources (*i.e.*, NetApp, NOS, NetSlicer, and NetDev) to demarcate the communication between such kind of entities located at different architectural planes.

3.2.3. Communication Model

Our Management Plane defines the User Interface, the Repository Interface, the Adapter Interface, and the set of Management Interfaces. The User Interface enables Network Administrators to interact with Management Services exposed by the Manager. The Repository Interface connects the Manager with the Data Repository. The Adapter Interface and Management Interfaces transport request messages (*i.e.*, operations and queries) from the Manager to a particular Agent, passing through the related Adapter. These both kind of interfaces also transmit reply messages and unsolicited events sent by an Agent towards the Manager. In order to match each Agent with its respective Adapter, we establish a Management Interface (MI) per SDN managed resource: NetApp MI, NOS MI, NetSlicer MI, and NetDev MI. Regarding communication support, the User Interface and the Adapter Interface must employ a consistent data format (*e.g.*, JavaScript Object Notation [JSON] [42] and eXtensible Markup Language [XML] [43]) and a standardized protocol (*e.g.*, HyperText Transfer Protocol [HTTP] [44] and Simple Object Access Protocol [SOAP] [45]) to exchange management data. The Repository Interface relies on technologies deployed by the Data Repository (*e.g.*, XML over HTTP). Finally, Management Interfaces handle data formats and protocols implemented by Agents (*e.g.*, OVSDB [11], Network Configuration Protocol [NETCONF] [46], and Simple Network Management Protocol [SNMP] [47]).

3.2.4. Functional Model

This proposal references the five OSI management functional areas to classify Management Services: Fault Services, Configuration Services, Accounting Services, Performance Services, and Security Services. Fault Services detect, separate, and fix failures in physical and logical SDN resources. Configuration Services modify and update behavior of SDN resources. Accounting Services tracks and allocate usage of SDN resources. Performance Services monitor, collect, and report information about the operation of SDN resources. Security Services control and analyze access to SDN resources. In addition, we include Programming Services to coordinate programmable software of SDN resources, such as checking and deploying versions of a particular NetApp running on a specific NOS. By using or combining the aforementioned Management Services, our Management Plane allows network administrators to carry out different SDN management requirements, as those described in [4].

4. Information Model

As aforementioned, our Management Plane requires an Information Model that provides a technology-agnostic and consistent abstraction of the whole SDN environment to enable integrated management. Few approaches provide models that attempt to characterize the SDN management environment [10, 26, 27], but they are tied to specific SDN instances and expose incomplete SDN representations.

In this paper, we introduce a CIM-based Information Model that provides a technology-independent and consistent abstraction of SDN managed and managing resources. This Information Model represents every plane in the

SDN architecture to encourage a complete SDN management regardless of deploying technologies. We adopted CIM because it offers higher expressiveness than other information definition languages (*e.g.*, SMI [30] and GDMO [31]), affording future robust semantic integration [32]. CIM supplies several classes, associations, properties, and methods to describe network resources at a conceptual level, such as Ethernet ports, LAN endpoints, and VLANs [48]. However, CIM lacks elements that represent specific SDN features for management [27]. Thus, our Information Model introduces new elements that extend the actual CIM Schema to characterize the whole SDN management environment. We present this extended schema as a graphical visualization composed of UML classes and associations that represent SDN managed and managing resources and their relationships.

In next paragraphs and figures (Figs. 2, 5, 4, and 3) we describe a simple version of the proposed Information Model. Specific properties and methods from each class are out of scope. We exclude the *CIM_* prefix from the current CIM elements and the *SDN_* prefix from the new elements. For example, *CIM_System* appears as *System* and *SDN_AgentService* as *AgentService*. To provide a better visualization, the proposed class schema displays gray background for the new classes, white background for the current CIM classes, bold characters for the new associations, and thin characters for the current CIM associations. For the sake of simplicity, we omit inheritance associations between the new classes and the current CIM classes. Unless otherwise stated, general inheritance associations satisfy the following: *(i)* the new classes with suffix *Capabilities* represent subclasses from the *EnabledLogicalElementCapabilities* CIM class, *(ii)* with suffix *Service* from the *Service* CIM class; and *(iii)* with suffix *Settings* from the *SettingData* CIM class. In addition, we skip the *BindsTo* CIM associations for the CIM classes *ServiceAccessPoint* and *ProtocolEndpoint*. The *BindsTo* association connects the class itself to define a protocol layering dependency between an upper and a lower protocol. For example, the OpenFlow protocol binds the TCP protocol to set the port and address enabled for OpenFlow communication.

Fig. 2 illustrates the extended class schema for the proposed Management Plane. We introduced five new classes to characterize the novel components defined in this approach: in the Managing Tier, the *ManagementService*, the *ManagementServiceCapabilities*, and the *AdapterService*; and in the Managed Tier, the *AgentService* and the *EventIndication*.

The *ManagementService* class represents Management Services that allow carrying out different SDN management functions. Through the *Element-*

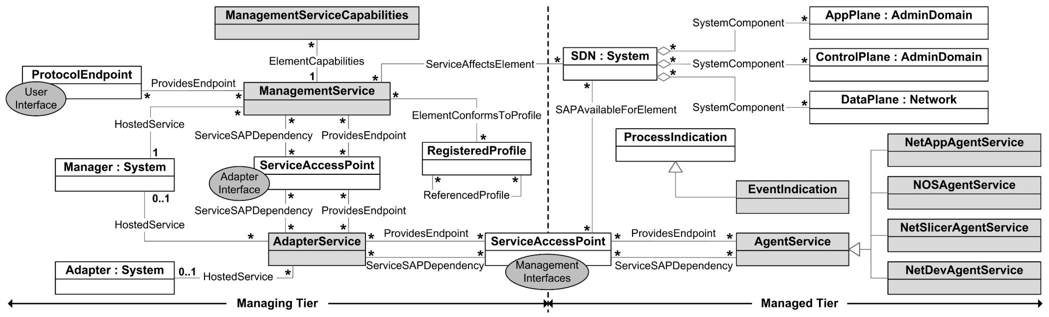


Figure 2: Class schema to model SDN Management Plane

Capabilities association, the *ManagementServiceCapabilities* class describes both supported and excluded abilities for Management Services. The *ManagementServiceCapabilities* relies on the Functional Model to classify SDN Management Services as Fault, Configuration, Accounting, Performance, Security, or Programming services (see Section 3.2.4). For example, a Management Service that modifies the SBI communication of NetDevs declares capabilities of Configuration Services.

The *RegisteredProfile* class models a CIM profile specification defined by any standard organization for managing SDNs. Each profile specification includes a small subset of the proposed class schema and delineates corresponding behavior as management requirements. The *ReferencedProfile* association indicates that a profile specification may reference others. In addition, the *ElementConformsToProfile* association describes which CIM profile specifications a Management Service apply. For example, a Configuration Service fulfills with a profile specification of DMTF that standardizes how to achieve seamless migration in NetDevs.

The *Manager* represents the system hosting the SDN Management Services. The *HostedService* association realizes this relationship between the *ManagementService* and the *Manager*. Although this model presents the *Manager* as an instance of the *System* class, it also may implement an instance of a subclass from *System*, such as *ComputerSystem*, *J2eeServer*, or a new class. For example, a Configuration Service may be carried out as a Web application running on either an Apache Tomcat Server or a GlassFish Server.

The *ProtocolEndpoint* class tagged as *User Interface* models the communication point that enables access of Network Administrators. The corresponding *ProvidesEndpoint* association implies that the *ManagementService* supplies such user *ProtocolEndpoint*. For example, a Configuration Service provides an HTTP interface to allow Network Administrators to set SBI parameters of NetDevs through a Web browser.

The *ServiceAccessPoint* class tagged as *Adapter Interface* represents the communication point between the *ManagementService* and the *AdapterService*. The *ProvidesEndpoint* associations connected to the adapter *ServiceAccessPoint* indicate that both the *ManagementService* and the *AdapterService* establish their own communication points to allow access from the other. The *ServiceSAPDependency* associations imply that both the *ManagementService* and the *AdapterService* utilize the adapter *ServiceAccessPoint* to access the other. The *ManagementService* and the *AdapterService* support properties and methods for sending and receiving requests, responses, and events through the adapter *ServiceAccessPoint*. For example, a Configuration Service and a NetDev Adapter establish a mutual communication using JSON over HTTP. Using this channel, the NetDev Adapter forwards to the Configuration Service an event from a NetDev Agent that notifies about failures with misconfiguration. Similarly, the Configuration Service uses the same channel to fix this failure by sending a configuration request to the NetDev Adapter. The NetDev Adapter forwards this request to the corresponding NetDev Agent.

The *AdapterService* class models an Adapter in charge of parsing and forwarding requests, responses, and events between the *ManagementService* and the *AgentService*. The *AdapterService* is a superclass that holds properties and methods for handling the communication through the adapter and management interfaces. Four subclasses inherit from the *AdapterService*: the *NetDevAdapterService*, the *NetSlicerAdapterService*, the *NOSAdapterService*, and the *NetAppAdapterService*. For the sake of brevity and because the behavior of these subclasses is very similar, we decide to exclude them in Fig. 2. Each subclass from the *AdapterService* adds properties and methods to support functionality provided by the subclass from the *AgentService* that uses the same managed resource name (*e.g.*, the *NetDevAdapterService* matches the *NetDevAgentService*). In addition, regarding this correlation, every subclass deriving from the *AdapterService* instruments specific aspects from the proposed class schema. For example, a NetDev Adapter, which matches a NetDev Agent, only concerns about functionality for managing NetDevs (see Fig. 5).

The *AdapterService* may be hosted by either the *Manager* or the *Adapter*. Both *HostedService* associations linked to the *AdapterService* indicate this relationship. As well as the *Manager*, the *Adapter* may be an instance of either the *System* class or one of its subclasses. For example, a NetDev Adapter may be executed on either the same server running Management Services or a different server.

The *ServiceAccessPoint* class tagged as *Management Interfaces* represents the communication point between the *AdapterService* and the *AgentService*. The *ProvidesEndpoint* and the *ServiceSAPDependency* associations related to the management *ServiceAccessPoint* indicate that both the *AdapterService* and the *AgentService* provide and utilize management interfaces to perform their functionality. Subclasses from the *AdapterService* and from the *AgentService* inherits these associations. Each instance of the subclasses from the *AdapterService* handles the protocol used by the corresponding instance of the subclasses from the *AgentService*, affording a protocol-agnostic communication for the *ManagementService*. For example, a NetDev Adapter uses the OF-CONFIG protocol to access a NetDev Agent for OpenFlow switches. A second NetDev Adapter utilizes the SNMP protocol to contact a second NetDev Agent for ForCES FEs. A Configuration Service communicates with both NetDev Adapters using a standardized data format and protocol (*e.g.*, JSON over HTTP). The NetDev Adapters forward to the NetDev Agents the management requests received from the Configuration Service. Similarly, the NetDev Adapters forward to the Configuration Service responses and events received from the NetDev Agents. Thus, the Configuration Service carry out a protocol-agnostic management on different NetDev technology instances.

The *AgentService* class represents an Agent running on SDN managed resources, such as NetDev, NetSlicer, NOS, and NetApp. This is a superclass that defines properties and methods for supporting the management *ServiceAccessPoint* and for handling the *EventIndication*. The *EventIndication* is a subclass from the *ProcessIndication* class. The *EventIndication* maps an unsolicited event sent by the *AgentService* towards the *ManagementService* to notify about changes and alerts in SDN managed resources. For example, a NetDev Agent dispatches an event that notifies a detected misconfiguration on its hosting NetDev. The corresponding NetDev Adapter receives this unsolicited event and forwards it to a Configuration Service.

Four subclasses derive from the *AgentService*: the *NetDevAgentService*, the *NetSlicerAgentService*, the *NOSAgentService*, and the *NetAppAgentService*. Each subclass supports methods to carry out management tasks in its hosting SDN managed resource, such as retrieving statistical information, modifying configuration parameters, discovering capabilities, and changing communication attributes.

We use the *System* class to model the *SDN* as an entity composed of the *DataPlane*, the *ControlPlane*, and the *AppPlane*. The *Network* class represents the *DataPlane* as a logical, virtual, or physical network that groups interconnected NetDevs capable of exchanging information. The *AdminDomain* class indicates that the *ControlPlane* and the *AppPlane* gather similarly managed components, such as NetSlicers and NOSs for the Control Plane, and NetApps for the Application Plane.

The *ServiceAffectsElement* association between the *SDN* and the *ManagementService* reflects that Management Services have an effect in the entire SDN, such as changing resource behavior, monitoring failures, and analyzing performance. Besides, the *SAPAvailableForElement* association between the *SDN* and the management *ServiceAccessPoint* implies that management interfaces provide managing access for the whole SDN.

Fig. 3 shows the extended class schema for the Application Plane. We introduced three new classes and two novel associations to describe specific management features of NetApps. The new classes are the *NetAppCapabilities*, the *NetAppSettings*, and the *NorthboundService*. The new associations are the *NetAppHostedOnNOS* and the *NetAppHostedOnServer*.

The *AppPlane*, modeled with the *AdminDomain* class, uses the *SystemComponent* association to aggregate instances of the *NetApp*. Leveraging the *ApplicationSystem* class, the *NetApp* represents NetApps holding business logic on top of the SDN architecture. For example, NetApps that carries out load-balancing and access control tasks.

We use the *HostedService* association to indicate that the *NetApp* hosts the *NetAppAgentService* and the *NorthboundService*. The *NorthboundService* class models modules that communicate with services exposed by NOSs. The *ProvidesEndpoint* and the *ServiceSAPDependency* associations reflect that the *NorthboundService* uses and provides functions through the northbound *ServiceAccessPoint*. For example, a load-balancing and access-control NetApps retrieve and supply data from and to tracking and firewall services deployed by an NOS.

The *ServiceAccessPoint* tagged as *Northbound Interfaces* models the communication between the Application Plane and the Control Plane. This northbound *ServiceAccessPoint* encompasses different NBIs, such as APIs based on protocols (*e.g.*, REST) and on programming languages (*e.g.*, Pyretic and Procera).

The *NetAppHostedOnNOS* association between the *NetApp* and the *NOS*

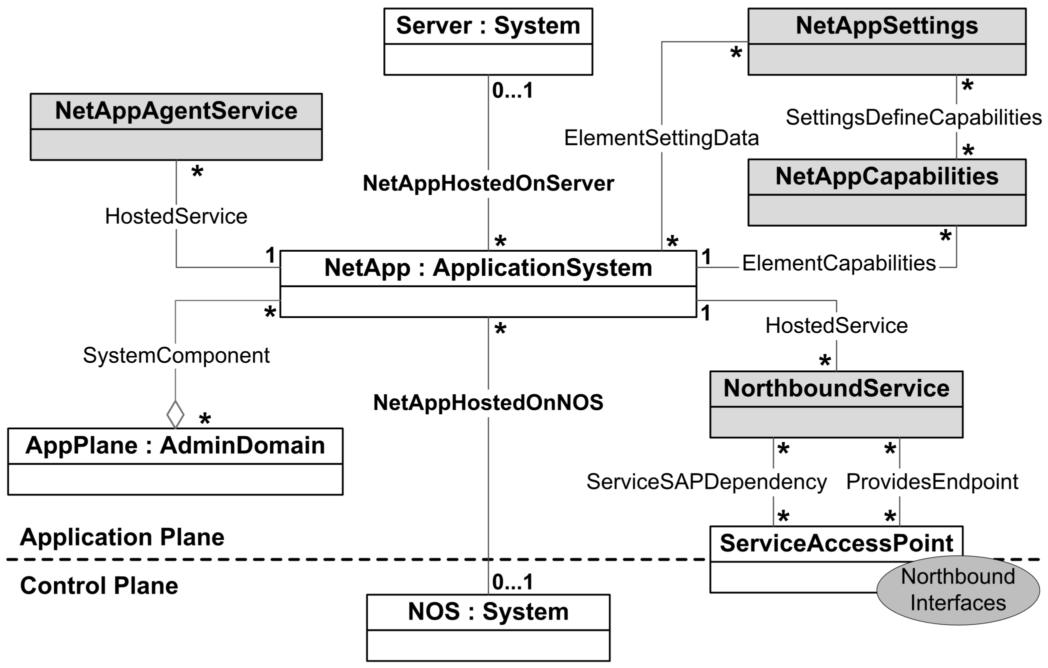


Figure 3: Class schema to model SDN Application Plane

represents local NetApps running on NOSs. Usually, these local NetApps utilize NBIs based on programming languages to access and supply functionality from and to NOS services. The *NetAppHostedOnServer* between the *NetApp* and the *Server* system models NetApps running on remote servers. These remote NetApps prefer NBIs based on protocols for communicating with the Control Plane.

Using the *ElementCapabilities* association, the *NetAppCapabilities* class describes the supported and excluded abilities of NetApps. The *NetAppSettings* class establishes configuration parameters for the *NetApp*. This relationship is depicted through the *ElementSettingData* association between the *NetAppSettings* and the *NetApp*. The *SettingsDefineCapabilities* association between the *NetAppSettings* and the *NetAppCapabilities* reflects that the setting data affect some NetApps capabilities. For example, configuring a different load-balancing algorithm modifies the behavior of the corresponding NetApp.

Fig. 4 describes the extended class schema for the Control Plane. Considering that CIM lacks elements that characterize the management information of NetSlicers and NOSs, we introduce seven new classes: the *SlicingService*, the *SlicingStatistics*, the *SlicingCapabilities*, the *SlicingSettings*, the *NOS-Service*, the *NOSServiceCapabilities*, and the *NOSServiceSettings*.

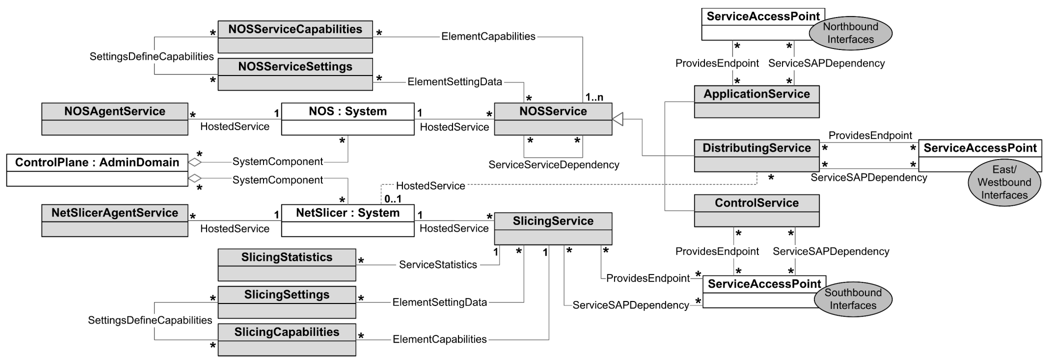


Figure 4: Class schema to model SDN Control Plane

The *AdminDomain* class uses the *SystemComponent* association to describe the *ControlPlane* as an entity composed of NOSs and NetSlicers. The *NOS* models an NOS, such as OpenFlow controllers and ForCES CEs. The *NetSlicer* represents a NetSlicer system, such as FlowVisor for OpenFlow-based networks. The *NOS* is the hosting system for the *NOSAgentService* and the *NetSlicer* for the *NetSlicerAgentService*.

The *NOSService* is a superclass that models network services hosted in NOSs. The *HostedService* association between the *NOSService* and the *NOS* indicates this relationship. Subclasses must inherit from the *NOSService* in order to define specific NOS services, such as tracking, route calculation, and firewall. We present three subclasses for NOS services: the *ApplicationService*, the *DistributingService*, and the *ControlService*. The *ApplicationService* depicts services that expose functionality to the Application Plane through the northbound *ServiceAccessPoint*. The *DistributingService* defines services that enable to deploy distributed Control Plane through the east/westbound *ServiceAccessPoint*. The *ControlService* describes services that handle the communication with NetDevs and NetSlicers through the southbound *ServiceAccessPoint*.

The *ServiceServiceDependency* association indicates that NOS services collaborate with or are necessary for other NOS services to perform their operation. For example, a topology service requires a tracking service to recognize hosts connected to specific switches.

We use the *ProvidesEndpoint* and the *ServiceSAPDependency* associations to correlate the *ApplicationService* with the northbound *ServiceAccessPoint*, the *DistributingService* with the east/westbound *ServiceAccessPoint*, and the *ControlService* with the southbound *ServiceAccessPoint*.

The *ServiceAccessPoint* tagged as *East/Westbound Interfaces* represents the communication point between distinct Control Plane domains. For example, the SDNi protocol from IETF and the CE-CE interface from ForCES. Although the exchange of information is usually carried out by NOSs, NetSlicers may also need to deploy a distributed architecture. This is reflected by the *HostedService* association between the *DistributingService* and the *NetSlicer*.

The *ServiceAccessPoint* tagged as *Southbound Interfaces* models the communication point between the Control Plane and the Data Plane. The *ServiceSAPDependency* and the *ProvidesEndpoint* associations reflect that the *ControlService* uses and supplies the southbound *ServiceAccessPoint* to send and receive messages to and from the Data Plane. This southbound *ServiceAccessPoint* encompasses different SBI protocols, such as OpenFlow, ForCES, and POF.

The *NOSServiceCapabilities* class declares the supported abilities of NOSs services. The *ElementCapabilities* association between the *NOSServiceCapabilities* and the *NOSService* reflects this relationship. The *NOSServiceSettings* class defines the configuration parameters for NOSs services. The *ElementSettingData* association between the *NOSServiceCapabilities* and the *NOSService* indicates this relationship. The *SettingsDefineCapabilities* association between the *NOSServiceSettings* and the *NOSServiceCapabilities* implies that configuring NOS services establishes some capabilities. For example, the time interval of a tracking service to send discovery messages.

The *SlicingService* class represents the functionality of a NetSlicer: divide the Data Plane into several isolated logical network instances (*a.k.a.* slices) and assign them to different NOSs. The *NetSlicer* hosts the *SlicingService* using the *HostedService* association. The *ProvidesEndpoint* and the *ServiceSAPDependency* associations between the *SlicingService* and the southbound *ServiceAccessPoint* indicate that NetSlicers provide and use SBIs to communicate with NetDevs and NOSs. For example, FlowVisor uses the OpenFlow protocol to communicate with both OpenFlow switches and OpenFlow controllers.

The *SlicingStatistics* class defines collections of metrics suitable to instances of the *SlicingService*. The *SlicingStatistics* is a subclass that derives from the *StatisticalData*. The *ServiceStatistics* association relates the *SlicingStatistics* with the *SlicingService*. For example, the total number of slices handled by a NetSlicer.

Through the *ElementCapabilities* association, the *SlicingCapabilities* class describes the supported and excluded capabilities of the *SlicingService*. Some of these capabilities are specified in the *SlicingSettings* class by means of the *SettingsDefineCapabilities* association. The *SlicingSettings* delineates the configuration parameters for the *SlicingService*. The *ElementSettingData* association between the *SlicingSettings* and the *SlicingService* reflects this relationship. For example, the maximum number of concurrent slices supported by a NetSlicer.

Fig. 5 depicts the extended class schema for the Data Plane. In order to describe specific management features of NetApps, we introduce five new classes: the *NetDevCapabilities*, the *NetDevResource*, the *NetDevResourceSettings*, the *NetDevService*, and the *NetDevServiceSettings*.

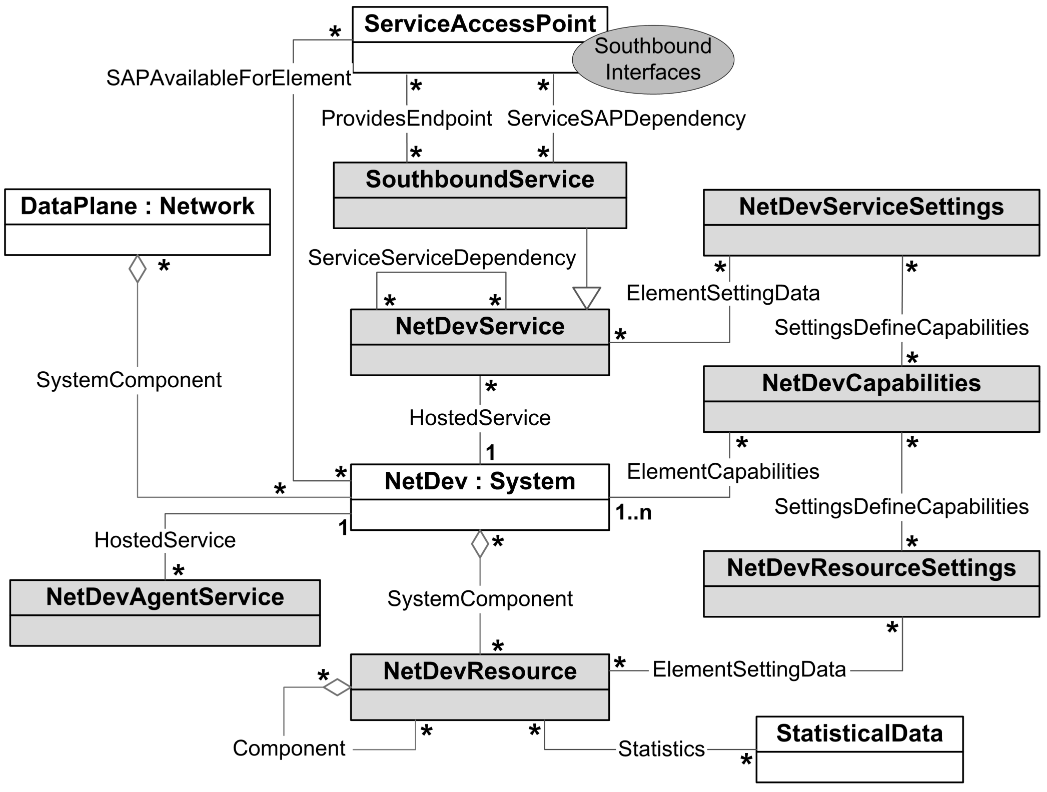


Figure 5: Class schema to model SDN Data Plane

As aforementioned, the *Network* class indicates that the *DataPlane* models a network composed of interconnected NetDevs. The *NetDev* represents a NetDev system within a network, such as OpenFlow switches and custom forwarding hardware (*e.g.*, OpenWrt and NetFPGA). The *DataPlane* aggregates the *NetDev* using the *SystemComponent* association. The *NetDev* hosts the *NetDevAgentService*.

The supported and excluded abilities of a NetDev are described by instances of the *NetDevCapabilities*. The *ElementCapabilities* association between the *NetDev* and the *NetDevCapabilities* indicates this relationship. For example, an OpenFlow switch declares simple capabilities such as forwarding functions based on match/action flow tables; a NetFPGA programmable hardware exposes more complex capabilities such as plugging modules to enable customizable functions. Both kinds of NetDevs also reveal network capacity enabled by its components, such as speed of ports and size of queues.

The *NetDevResource* class inherits from the *EnabledLogicalElement* to model network elements composing a NetDev, such as ports, queues, and tables. The *SystemComponent* association between the *NetDev* and the *NetDevResource* implies this aggregation. The *NetDevResource* is a superclass from which individual subclasses inherit to represent NetDev components. For example, a subclass called *FlowTable* to characterize flow tables that compose OpenFlow switches. In addition, the *Component* association connected to the *NetDevResource* models a network element composed of others, such as ports including various queues.

The *StatisticalData* is a superclass to define arbitrary collections of statistical information applicable to instances of the *NetDevResource*. The *Statistics* association attaches the *StatisticalData* with the *NetDevResource*. For example, ports in switches delineate transmission metrics, such as received and transmitted bytes, packets, and errors.

The *NetDevResourceSettings* class describes the configuration of network elements that compose NetDevs. The *ElementSettingData* association between the *NetDevResourceSettings* and the *NetDevResource* reflects this relationship. For example, the speed operation of ports and the buffer size of queues. The *SettingsDefineCapabilities* association between the *NetDevResourceSettings* and the *NetDevCapabilities* indicates that the configuration parameters of NetDev components specify some capabilities of NetDevs.

The *NetDevService* is a superclass that represents network services hosted in NetDevs. This hosting relationship is depicted with the *HostedService* association between the *NetDevService* and the *NetDev*. Subclasses must derive from the *NetDevService* in order to model particular NetDev services, such as forwarding, route calculation, and firewall. This is the case of the *SouthboundService*, which defines services that query, receive, and execute instructions to and from the Control Plane. The *SouthboundService* inherits from the *NetDevService* and includes properties and methods to handle the communication through the SBIs. For example, the Secure Channel component in OpenFlow switches manipulates the OpenFlow protocol to communicate with external controllers and update internal flow tables.

The *ServiceServiceDependency* association indicates that NetDev services cooperate with or are required for other NetDev services to perform their functions. For example, an inspection service requires a forwarding service to redirect malicious packets to a specific destination.

The *ServiceSAPDependency* and the *ProvidesEndpoint* associations reflect that the *SouthboundService* uses and supplies the southbound *ServiceAccessPoint* to send and receive messages to and from the Control Plane. The *SAPAvailableForElement* association between this *ServiceAccessPoint* and the *NetDev* implies that the SBIs allow access from the Control Plane for managing NetDevs components and hosted services. For example, the OpenFlow protocol enables to manipulate flow tables within OpenFlow switches and the ForCES protocol facilitates to configure logical functions residing in ForCES FEs.

Through the *ElementSettingData* association, the *NetDevServiceSettings* class describes the configuration parameters of services hosted in NetDevs. For example, the routing algorithm of a route calculation service and the list of OpenFlow controllers of a southbound service. The *SettingsDefineCapabilities* association between the *NetDevServiceSettings* and the *NetDevCapabilities* implies that the configuration of NetDev services characterizes some capabilities of NetDevs.

5. Case Study

To assess our approach, first, we establish a network management scenario that deploys different SDN management technologies. Second, we implement the system prototype that relies on the present approach. Third, we build up a test environment based on the described scenario. Fourth, we conduct a performance evaluation to determine the feasibility of using the proposed approach in terms of time-response and network traffic. These metrics are related to the behavior on runtime of solutions used to manage heterogeneous SDNs. Finally, we expose the qualitative features of this approach.

5.1. Scenario: configuring SDN-based networks by using heterogeneous management interfaces

Let us suppose that a Cloud Service Provider (CSP) enables access to its cloud resources by deploying a basic SDN data center network: three tiers of NetDevs (*i.e.*, core, aggregation, and edge) handled by a *Current NOS* and arranged in a simple tree topology (*i.e.*, each NetDev has a single parent). Usually, the CSP Network Administrator purchase SDN forwarding resources from *Vendor A*. However, at some point in time, the Network Administrator decided to buy NetDevs from *Vendor B* because of economic profits. As the network became bigger and since the SDN technology updates constantly, the CSP decided to install a *New NOS* that offers better performance, reliability, and security features. Now, the Network Administrator faces the challenge of configuring heterogeneous NetDevs for being controlled by the *New NOS*.

Considering that *Vendor A* provides a different NetDev management interface than *Vendor B*, the Network Administrator typically would use an *Isolated Solution* (*i.e.*, *Vendor A* Tool and *Vendor B* Tool) to execute specific configuration commands on NetDevs from distinct vendors. This solution hinders and retards managing tasks of Network Administrator. Instead, our approach hides network heterogeneity by establishing a common NetDev configuration model and by adapting to each vendor management interface. Thus, we afford an *Integrated Solution* that allows the Network Administrator to seamlessly configure every NetDev to be controlled by the new NOS, mitigating the complexity and time consumption of managing heterogeneous SDN resources.

Fig. 6 illustrates the above-described scenario. NetDevs provided by *Vendor A* are OpenFlow switches that enable the OVSDB management interface to accept configuration requests (*i.e.*, OVSDB switches). NetDevs from *Vendor B* are OpenFlow switches that run the OF-CONFIG server to support configuration through the OF-CONFIG protocol (*i.e.*, OF-CONFIG switches). The *Current NOS* that initially handles the OpenFlow switches is a Floodlight controller. The *New NOS* that has to be set in the OpenFlow switches for controlling them is an Opendaylight controller. In addition, we defined a managing operation called *SetController*. This operation describes the process of configuring a number of NetDevs that provide distinct management interfaces (*i.e.*, OVSDB switches and OF-CONFIG switches) in order to establish the *New NOS* (*i.e.*, Opendaylight) as their controller. *SetController* represents a common management task that Network Administrators must perform when conducting updating, maintenance, and recovery functions in heterogeneous SDN-based networks.

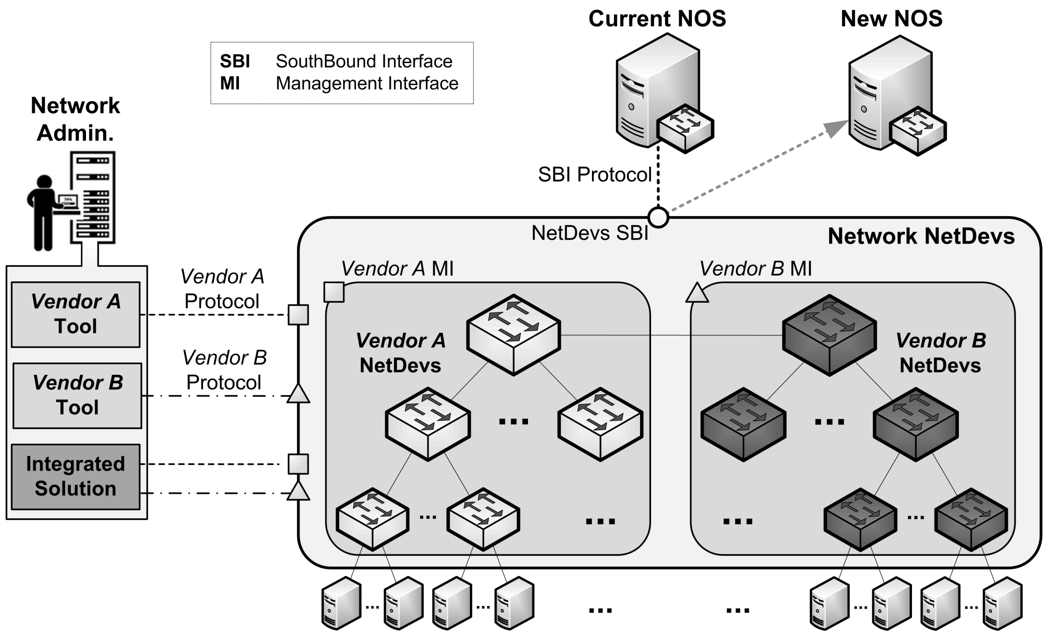


Figure 6: Scenario

5.2. Implementation

To evaluate our approach, we developed two prototypes to conduct the *SetController* operation: *Integrated Solution* and *Isolated Solution*.

5.2.1. Integrated Solution

We built this prototype upon the proposed approach for performing *SetController* regardless the different configuration interfaces. Fig. 7 depicts the implemented *Integrated Solution*. The Data Repository is a CIM Object Manager (CIMOM) that provides RRM as CIM schemas and stores instance data as CIM instances. CIMOM is the main component of a Web-Based Enterprise Management (WBEM) framework. CIM schemas characterize the SDN managed environment while CIM instances represent the SDN managed resources. To build RRM, we compiled both the CIM Schema 2.18.1

and the SDN Extension Schema. The SDN Extension Schema implements the proposed Information Model.

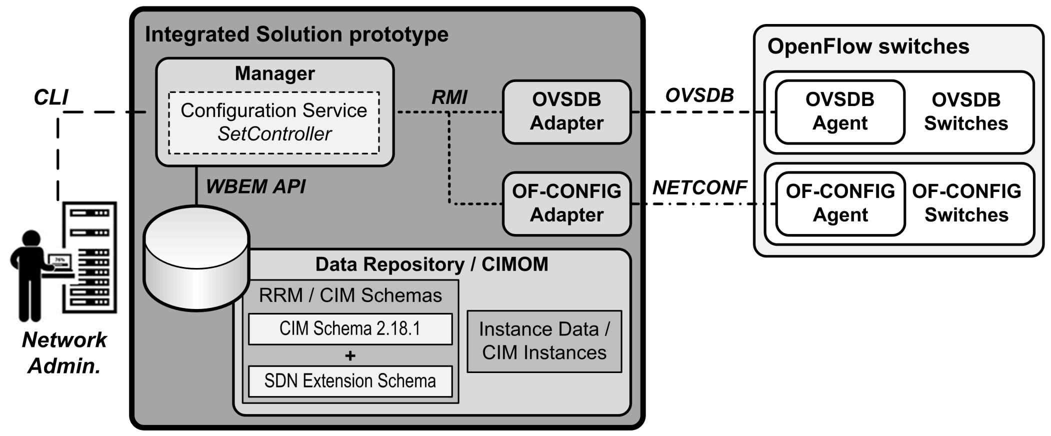


Figure 7: Integrated Solution prototype

Since this prototype focuses on the *SetController* operation, the compiled SDN Extension Schema is limited to the following new classes from the proposed Information Model: *AgentService*, *NetDevAgentService*, *AdapterService*, *NetDevAdapterService*, *NetDevService*, and *SouthboundService*. In addition, CIMOM stores CIM instances of the above-mentioned classes and of classes included in the CIM Schema 2.18.1: *ComputerSystem*, *HostedService*, *RemotePort*, *ServiceSAPDependency*, *TCPProtocolEndpoint*, *ProvidesEndpoint*, *IPProtocolEndpoint*, and *BindsTo*. We used the Managed Object Format (MOF) to formally express the SDN Extension Schema and the CIM instances.

The Manager is carried out in a Java application. This Manager deploys the *SetController* operation as a Configuration Service. The User Interface of the Manager is a simple Command Line Interface (CLI) that allows executing the configuration commands of *SetController*. The Repository Interface uses the WBEM API to read and write instances stored in CIMOM. The Adapter Interface relies on the Remote Method Invocation (RMI) to communicate the Manager and the Adapters.

The Java application also deploys the NetDev Adapters: the OVSDB Adapter and the OF-CONFIG Adapter. The OVSDB Adapter employs the Opendaylight OVSDB API to communicate with the OVSDB Agent that maintains the configuration database of OVSDB switches. The OF-CONFIG

Adapter relies on the NETCONF4J API to connect with the OF-CONFIG Agent deployed by the OF-CONFIG switches for accepting configuration requests. OF-CONFIG utilizes NETCONF as the associated protocol.

Based on the information retrieved from CIMOM, the Manager invokes the appropriate Adapter. Once achieved the configuration in each requested OpenFlow switch, the Manager updates the instance data stored in CIMOM.

5.2.2. Isolated Solution

This prototype describes a classic solution of using a configuration tool for each management technology (*i.e.*, OVSDB and OF-CONFIG) to perform operations as *SetController*. We built both the OVSDB Tool and the OF-CONFIG Tool as bash scripts that automatize the usage of their underlying software. The OVSDB Tool uses the *ovs-vsctl* program to configure OVSDB switches. The OF-CONFIG Tool employs the NETCONF client *netopeer-cli* to communicate with OF-CONFIG switches. Both tools provide a simple CLI to specify the configuration parameters.

5.3. Test environment

To evaluate the proposed approach, we conducted a case study in a test environment that allowed deploying the described scenario. Fig. 8 depicts this environment formed by two OpenFlow networks, two OpenFlow controllers, the Manager Client, and CIMOM. Each OpenFlow network ran on an Ubuntu Server 14.04 Virtual Machine (VM) with 1 virtual processor and 1.5 GB RAM assigned, both hosted by an Ubuntu 14.04 machine with 2.53 GHz Intel Core i5 processor and 4 GB RAM. Each VM executed Mininet 2.2.1, a software for emulating OpenFlow-based networks, to deploy a simple tree topology with 111 Open vSwitches 2.3.1. A tunnel over an IP network interconnected the root switches from each tree topology.

The Open vSwitches used the OpenFlow protocol over a second IP network to communicate with a specific OpenFlow controller: Floodlight v1.0.1 or Opendaylight Helium. Later in each evaluation, we will define the exact quantity of switches per controller. Each OpenFlow controller operated on an Ubuntu 14.04 machine with 2.4 GHz Intel Core 2 duo processor and 2 GB RAM.

A third IP Network transmitted management data among the OpenFlow networks, the Manager Client, and CIMOM. The Manager Client was an Ubuntu 14.04 machine with 2.33 GHz Intel Core 2 duo processor and 2 GB RAM. The Manager Client hosted the Manager and Adapters components of

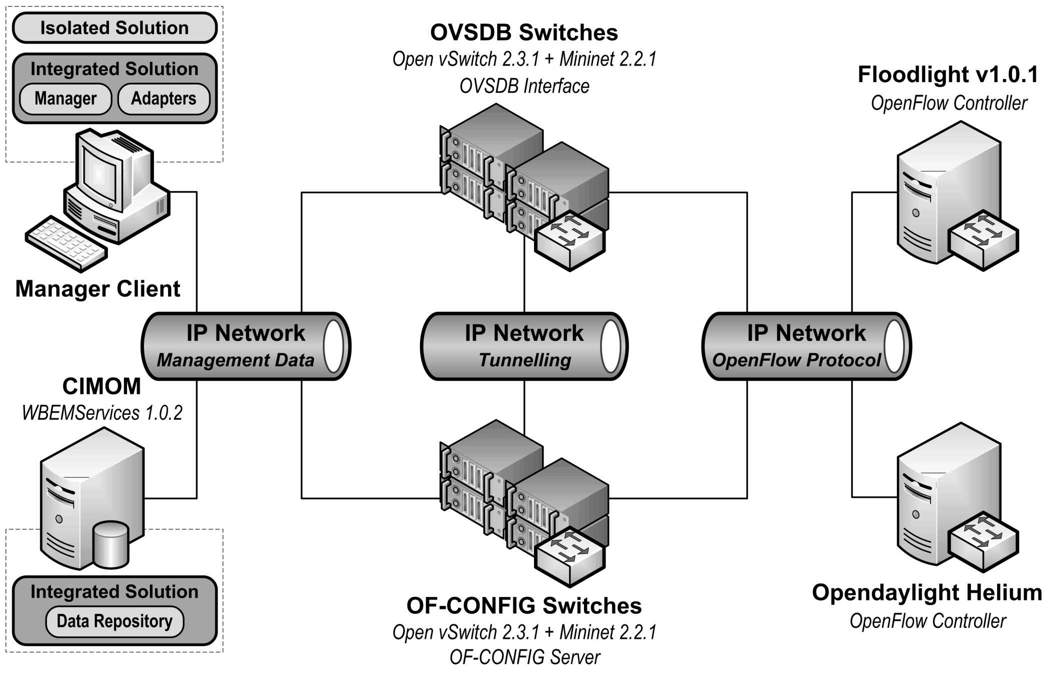


Figure 8: Test Environment

the *Integrated Solution*, and the configuration tools of the *Isolated Solution*. We executed CIMOM from the WBEM Services 1.0.2 on an Ubuntu 14.04 machine with 2.53 GHz Intel Core 2 Duo processor and 4 GB RAM. CIMOM realizes the Data Repository of the *Integrated Solution* prototype.

5.4. Evaluation and analysis

To evaluate the proposed approach, it was proceeded to measure the time-response and the network traffic of the *Integrated Solution* and the *Isolated Solution* when used in the test environment (see Fig. 8) to conduct the operation *SetController*. Although the test environment allows to perform such operation in parallel for reducing the overall time-response, we carried out a sequential configuration for assuming the worst scenario. Furthermore, since many Open vSwitches run on the same VM, executing the operation in sequence avoided readings distorted by the overuse of shared resources. The number of configured switches for each evaluation was 2, 20, 50, 100, 150, and 200. Half were OVSDB Switches, and the other half were OF-CONFIG Switches. It is worth to mention that the values of 2 and 20 configured switches allowed us to demarcate a boundary for the analysis in terms of time-response and network traffic. In all evaluation cases, we took the average values for 30 measurements with a 95% confidence level.

Fig. 9 depicts the time-response results. Time-response is the time in seconds (s) measured since the Network Administrator executes the operation *SetController* on the Manager Client until receiving the reply of the last configured switch. Nevertheless, since a configuration reply is received for each switch, we provide a per switch basis analysis. Considering that the time-response (r in s) of Web systems can be ranked as optimal ($r \leq 0.1$), good ($0.1 < r \leq 1$), admissible ($1 < r \leq 10$), and deficient ($r > 10$) [49], the time-response results reveal: *(i) SetController* of both the *Isolated Solution* and the *Integrated Solution* has an admissible r that grows moderately (less than $1.5s$ and $1.8s$ per switch, respectively) when the number of configured switches increases, *(ii)* the *Integrated Solution* takes longer than the *Isolated Solution*, as expected for using additional components (*e.g.*, CIMOM and Adapters) to cope with the heterogeneity; and *(iii)* the time-response overhead per switch of the *Integrated Solution* is $0.8s$ for 2 switches and less than $0.35s$ for 20 switches or more. Based on the above results, the time-response overhead of the *Integrated Solution* ($Tr_{oh:integrated}$) depends on the number of configured switches (N_{sw}): *(i)* if $N_{sw} < 20$, $Tr_{oh:integrated} \leq 0.8N_{sw}$ and *(ii)* if $N_{sw} \geq 20$, $Tr_{oh:integrated} \leq 0.35N_{sw}$.

Let us compare the time-response results with the time that the Network Administrator takes to build the configuration command on the CLIs (*i.e.*, time composing). In this case, the *Isolated Solution* aggregates an overhead to the time composing of the *Integrated Solution*. This is because the *Isolated Solution* forces the Network Administrator to decide which tool must use to configure each OpenFlow switch. Unlike this, the *Integrated Solution* abstracts the heterogeneity of the configuration technologies of the OpenFlow switches.

To compute the time composing overhead of the *Isolated Solution* ($Tc_{oh:isolated}$), we use the Keystroke-Level Model (KLM) [50] because it is useful to estimate the time that an expert user (*i.e.*, the Network Administrator) spends to accomplish a routine task supported on computer keyboard and mouse (*i.e.*, build the configuration command on the CLI). Previous researches demonstrate the feasibility of using KLM for conducting this kind of time evaluation [22, 51]. In KLM, each task is modeled as a sequence of actions. We take two KLM operators: *(i)* press and release a key, $K = 0.2s$, and *(ii)* mentally preparing for decision making, $M = 1.35s$. In the best case, the Network Administrator carries out on the *Isolated Solution* the follow-

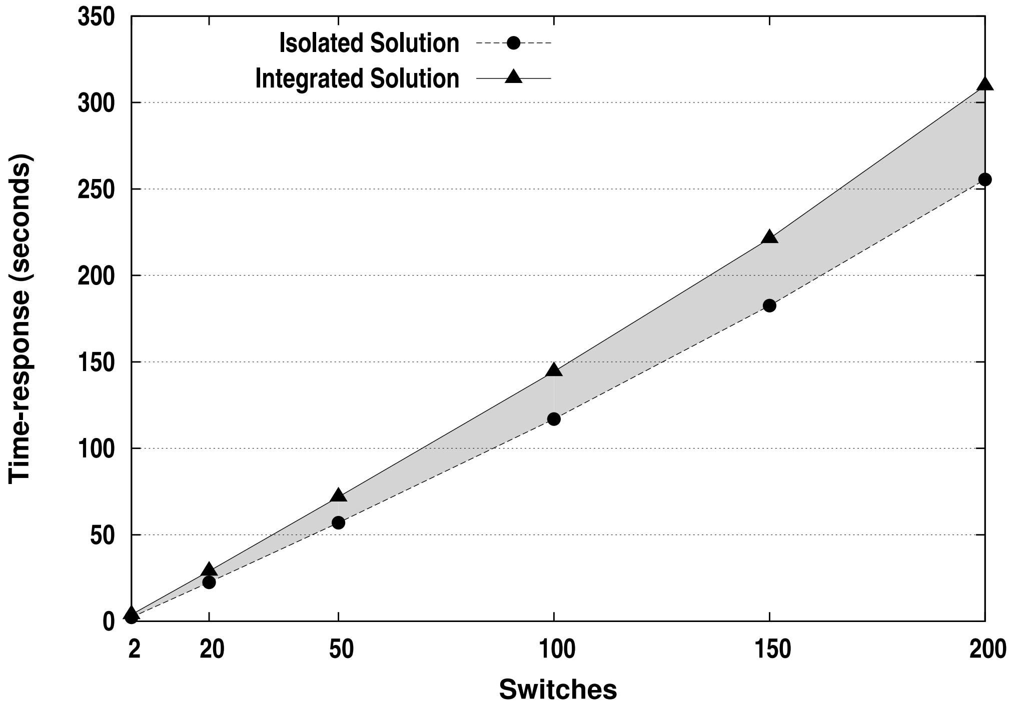


Figure 9: Configuring evaluation: time-response

ing additional actions: (i) change once from one tool to another by pressing ALT and TAB keys, and (ii) decide which tool must use for each switch. Based on these actions, $Tc_{oh:isolated}$ is also proportional to N_{sw}, therefore: $Tc_{oh:isolated} = K + MN_{sw} = 0.4 + 1.35N_{sw}$. The results obtained and estimated reveal that $Tr_{oh:integrated}$ is always less than $Tc_{oh:isolated}$.

Summing up, although the *Integrated Solution* includes additional modules (*e.g.*, CIMOM and adapters) to cope with the complexity of managing heterogeneous SDN environments, it introduces a time-response overhead shorter than the time composing overhead of the *Isolated Solution*. Certainly, the difference between the time overheads increases as more switches and distinct technologies incorporate the SDN managed environment. Additionally, considering that the time-consumption (Tt) is the sum of the time-response and the time composing, the difference between the time overheads demonstrates that the *Integrated Solution* reduces the time-consumption for carrying out the operation *SetController*, as can be seen in Eq. (1). Therefore, the time-response results corroborate that, in terms of such metric, it is feasible to use the proposed approach for executing management operations like the proved *SetController*.

$$\begin{aligned}
Tt_{integrated} &= Tr_{integrated} + Tc_{integrated} \\
Tt_{isolated} &= Tr_{isolated} + Tc_{isolated} \\
\\
Tr_{integrated} &= Tr_{oh:integrated} + Tr_{isolated} \\
Tc_{isolated} &= Tc_{oh:isolated} + Tc_{integrated} \\
\\
Tt_{integrated} &= Tr_{isolated} + Tc_{isolated} \\
&+ Tr_{oh:integrated} - Tc_{oh:isolated}
\end{aligned} \tag{1}$$

$$Tt_{integrated} = \begin{cases} Tt_{isolated} - 0.4 - 0.55N_{sw} & , N_{sw} < 20 \\ Tt_{isolated} - 0.4 - N_{sw} & , N_{sw} \geq 20 \end{cases}$$

Fig. 10 presents the network traffic results. Network traffic is the amount of data in kilobytes (KB) transmitted and received by the network interface of the Manager Client. These results reveal: *(i)* the traffic generated by *SetController* of both the *Isolated Solution* and the *Integrated Solution* grows moderately (approx 106 KB and 124 KB per switch, respectively) when the number of configured switches increases, *(ii)* the *Integrated Solution* generates more traffic than the *Isolated Solution*, as expected for handling management information of switches in CIMOM; and *(iii)* the additional traffic generated by the *Integrated Solution* is 32% for 2 switches and less than 17% for 20 switches or more. Considering that the *Integrated Solution*, unlike the *Isolated Solution*, copes with the heterogeneity of SDN environments by operating with standardized management data, the above facts corroborate that *SetController* of the *Integrated Solution* has a good behavior on network traffic.

Regarding the results obtained in the time-response and network traffic evaluation of the operation *SetController*, it is important to mention: *(i)* approx 94% of the time-response of *Isolated Solution* corresponds to the OF-CONFIG Tool, *(ii)* the OVSDB Tool generated approx 87% of the network traffic of *Isolated Solution*; and *(iii)* the time-response and network traffic overheads introduced by the *Integrated Solution* is smaller for many switches than for a few; this is because both the connection and the authentication

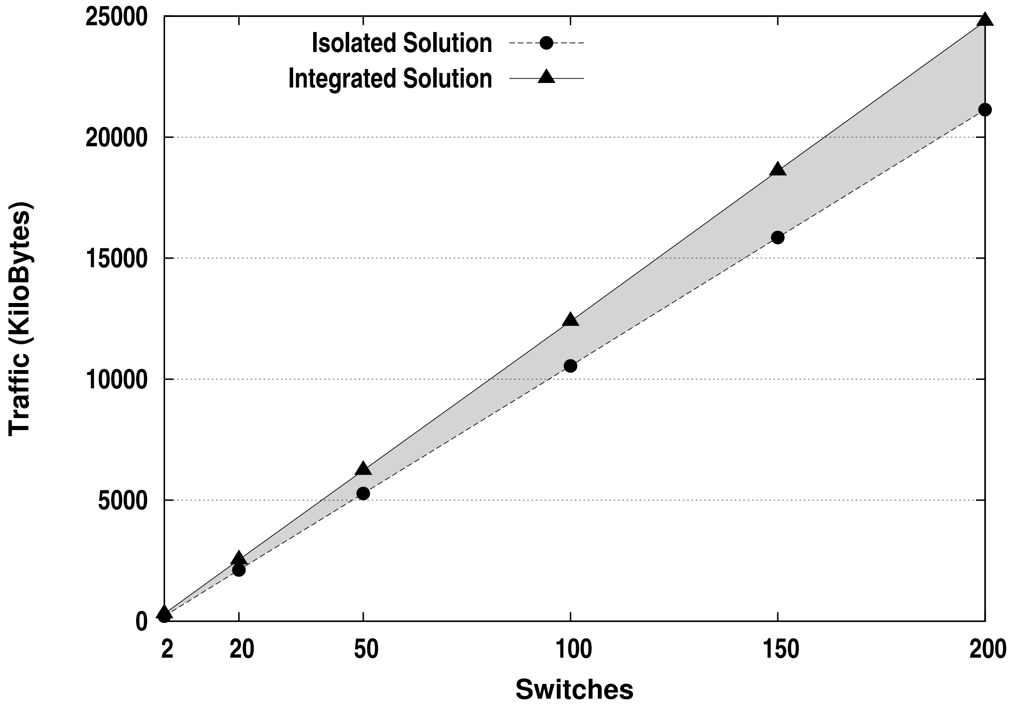


Figure 10: Configuring evaluation: network traffic

with CIMOM were realized just once for any number of configured switches. Summarizing, the time-response and network traffic results demonstrated that, in terms of such metrics, it is feasible to use the proposed approach in heterogeneous SDN environments to perform operations as the executed *SetController*.

From a qualitative point of view, our approach provides mainly simplicity and formalization. The simplicity refers to that the proposed Management Plane facilitates integrating the SDN management operations of network administrators. They do not require to employ multiple frameworks to completely manage SDNs deployed with various technologies because the proposed plane addresses the management requirements of the whole SDN environment and hides the heterogeneity of the deployed resources. Regarding the formalization, the Information Model introduced in this paper can be considered as a step forward in unifying a conceptual understanding of the SDN managed environment. It is possible because the Information Model relies on CIM to provide a technology-agnostic and consistent characterization of SDN. Moreover, future SDN proposals may extend this approach for tackling arising challenges in SDN management.

6. Conclusions and Future Work

In this paper, we introduced a Management Plane aimed to facilitate the integrated management of the whole SDN architecture in heterogeneous environments. We provided a description of this Management Plane by referencing the four OSI network management submodels: Information, Organization, Communication, and Function. Furthermore, we relied on CIM to define a consistent Information Model that characterizes the entire SDN management environment regardless of the deploying technologies. This Information Model extends the CIM Schema to accomplish a generic abstraction of the SDN managed and managing resources and their relationships. It is noteworthy that our proposal looks at the complete SDN management aspect instead of only modeling a certain part, empowering a fully integrated solution for managing heterogeneous SDNs.

We carried out and evaluated the proposed approach with a prototype in a realistic scenario based on SDN. This scenario established a particular challenge: configuring a heterogeneous SDN composed of NetDevs that deploy distinct management technologies. Our approach corroborated to be feasible when effectively (in terms of time-response and network traffic) overcoming such challenge. Through a quantitative perspective, the evaluation results revealed: *(i)* regarding the performance analysis for Java Websites [49], it has an admissible behavior in terms of time-response, similar than using isolated tools, *(ii)* it introduces a small time-response overhead ($< 0.8s$ per switch) compared with the minimal time required by network administrators to handle dispersed solutions ($> 1.35s$ per switch), and *(iii)* it has a good behavior on network traffic when managing several devices ($< 17\%$ for 20 switches or more) in relation to employing distinct tools. From a qualitative point of view, the proposed approach reduces the complexity of SDN management by including modules (*e.g.*, Data Repository and Adapters) that hide heterogeneity of the SDN environments.

As future research, we plan to evaluate the proposed Information Model with other SDN technology instances (*e.g.*, ForCES). We are also interested in focusing on assessing the remaining submodels from the Management Plane (*e.g.*, Communication and Functions) in order to afford a complete SDN management architecture.

Acknowledgments

This research was supported in part by the University of Cauca (Colombia), the Foundation University of Popayan (Colombia), the Federal University of Rio Grande do Sul (Brazil), and by Colciencias (Colombia) under Ph.D. scholarship 647-2014 granted to M.Sc.(c) Estrada-Solano.

References

[1] N. Feamster, J. Rexford, E. Zegura, The road to sdn, Queue 11 (12) (2013) 20:20–20:40. doi:10.1145/2559899.2560327.
URL `http://doi.acm.org/10.1145/2559899.2560327`

[2] P. Lin, J. Bi, H. Hu, T. Feng, X. Jiang, A quick survey on selected approaches for preparing programmable networks, in: Proceedings of the 7th Asian Internet Engineering Conference, AINTEC '11, ACM, New York, NY, USA, 2011, pp. 160–163. doi:10.1145/2089016.2089044.
URL `http://doi.acm.org/10.1145/2089016.2089044`

[3] H. Kim, N. Feamster, Improving network management with software defined networking, Communications Magazine, IEEE 51 (2) (2013) 114–119. doi:10.1109/MCOM.2013.6461195.

[4] J. Wickboldt, W. De Jesus, P. Isolani, C. Both, J. Rochol, L. Granville, Software-defined networking: management requirements and challenges, Communications Magazine, IEEE 53 (1) (2015) 278–285. doi:10.1109/MCOM.2015.7010546.

[5] ONF, SDN architecture v1.0, Technical Reference TR-502, Open Network Foundation (June 2014).
URL `https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf`

[6] R. Wang, D. Butnariu, J. Rexford, Openflow-based server load balancing gone wild, in: Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services, Hot-ICE'11, USENIX Association, Berkeley, CA, USA, 2011, pp. 12–12.
URL `http://dl.acm.org/citation.cfm?id=1972422.1972438`

[7] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, N. McKeown, Elastictree: saving energy in data center networks, in: Proceedings of the 7th USENIX conference on Networked systems design and implementation, NSDI'10, USENIX Association, Berkeley, CA, USA, 2010, pp. 17–17.
URL `http://dl.acm.org/citation.cfm?id=1855711.1855728`

[8] A. K. Nayak, A. Reimers, N. Feamster, R. Clark, Resonance: dynamic access control for enterprise networks, in: Proceedings of the 1st ACM workshop on Research on enterprise networking, WREN '09, ACM, New York, NY, USA, 2009, pp. 11–18. doi:10.1145/1592681.1592684.
URL `http://doi.acm.org/10.1145/1592681.1592684`

[9] K.-K. Yap, M. Kobayashi, D. Underhill, S. Seetharaman, P. Kazemian, N. McKeown, The stanford openroads deployment, in: Proceedings of the 4th ACM International Workshop on Experimental Evaluation and Characterization, WINTECH '09, ACM, New York, NY, USA, 2009, pp. 59–66. doi:10.1145/1614293.1614304.
URL `http://doi.acm.org/10.1145/1614293.1614304`

[10] ONF, Openflow management and configuration protocol (OF-CONFIG) v1.2, Technical Specification TS-016, Open Network Foundation (2014).
URL `https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1.2.pdf`

[11] B. Pfaff, B. Davie, The open vswitch database management protocol, RFC 7047 (December 2013).
URL `http://www.ietf.org/rfc/rfc7047.txt`

[12] S. Hassas Yeganeh, Y. Ganjali, Kandoo: a framework for efficient and scalable offloading of control applications, in: Proceedings of the first workshop on Hot topics in software defined networks, HotSDN '12, ACM, New York, NY, USA, 2012, pp. 19–24. doi:10.1145/2342441.2342446.
URL `http://doi.acm.org/10.1145/2342441.2342446`

[13] A. Tootoonchian, Y. Ganjali, Hyperflow: a distributed control plane for openflow, in: Proceedings of the 2010 internet network management conference on Research on enterprise networking, INM/WREN'10, USENIX

Association, Berkeley, CA, USA, 2010, pp. 3–3.
URL http://dl.acm.org/citation.cfm?id=1863133.1863136

[14] D. Mattos, N. Fernandes, V. da Costa, L. Cardoso, M. Campista, L. Costa, O. Duarte, Omni: Openflow management infrastructure, in: Network of the Future (NOF), 2011 International Conference on the, 2011, pp. 52–56. doi:10.1109/NOF.2011.6126682.

[15] S. Natarajan, X. Huang, An interactive visualization framework for next generation networks, in: Proceedings of the ACM CoNEXT Student Workshop, CoNEXT '10 Student Workshop, ACM, New York, NY, USA, 2010, pp. 14:1–14:2. doi:10.1145/1921206.1921222.
URL http://doi.acm.org/10.1145/1921206.1921222

[16] R. Corin, M. Gerola, R. Riggio, F. De Pellegrini, E. Salvadori, Vertigo: Network virtualization and beyond, in: Software Defined Networking (EWSDN), 2012 European Workshop on, 2012, pp. 24–29. doi:10.1109/EWSDN.2012.19.

[17] E. Salvadori, R. Doriguzzi Corin, M. Gerola, A. Broglio, F. De Pellegrini, Demonstrating generalized virtual topologies in an openflow network, SIGCOMM Comput. Commun. Rev. 41 (4) (2011) 458–459. doi:10.1145/2043164.2018520.
URL http://doi.acm.org/10.1145/2043164.2018520

[18] J. Reich, C. Monsanto, N. Foster, J. Rexford, D. Walker, Modular SDN programming with pyretic, USENIX 38 (5) (2013) 40–47.
URL https://www.usenix.org/system/files/login/articles/09_reich-online.pdf

[19] A. Voellmy, H. Kim, N. Feamster, Procera: A language for high-level reactive network control, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN '12, ACM, New York, NY, USA, 2012, pp. 43–48. doi:10.1145/2342441.2342451.
URL http://doi.acm.org/10.1145/2342441.2342451

[20] O. M. C. Rendon, C. R. P. dos Santos, A. S. Jacobs, L. Z. Granville, Monitoring virtual nodes using mashups, Computer Networks 64 (0) (2014) 55 – 70. doi:http://dx.doi.org/10.1016/j.comnet.2014.02.007.

URL http://www.sciencedirect.com/science/article/pii/S1389128614000498

[21] O. Caicedo Rendon, F. Estrada Solano, L. Zambenedetti Granville, An approach to overcome the complexity of network management situations by mashments, in: Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on, 2014, pp. 875–883. doi:10.1109/AINA.2014.142.

[22] O. Caicedo Rendon, F. Estrada-Solano, L. Zambenedetti Granville, A mashup ecosystem for network management situations, in: Global Communications Conference (GLOBECOM), 2013 IEEE, 2013, pp. 2249–2255. doi:10.1109/GLOCOM.2013.6831409.

[23] O. M. C. Rendon, F. Estrada-Solano, L. Z. Granville, A mashup-based approach for virtual SDN management, in: Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual, 2013, pp. 143–152. doi:10.1109/COMPSAC.2013.22.

[24] ONF, Common Information Model (CIM) overview, Technical Reference TR-513, Open Network Foundation (November 2015).
URL https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/Common_Information_Model_CIM_Overview_1.pdf

[25] ONF, Core information model (CoreModel)) v1.1, Technical Reference TR-512, Open Network Foundation (November 2015).
URL https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/ONF-CIM_Core_Model_base_document_1.1.pdf

[26] E. Haleplidis, J. Hadi Salim, S. Denazis, O. Koufopavlou, Towards a network abstraction model for sdn, Journal of Network and Systems Management 23 (2) (2015) 309–327. doi:10.1007/s10922-014-9319-3.
URL http://dx.doi.org/10.1007/s10922-014-9319-3

[27] B. Pinheiro, R. Chaves, E. Cerqueira, A. Abelem, Cim-sdn: A common information model extension for software-defined networking, in: Globecom Workshops (GC Wkshps), 2013 IEEE, 2013, pp. 836–841. doi:10.1109/GLOCOMW.2013.6825093.

[28] ISO, Information technology – Open Systems Interconnection – Systems management overview, ISO/IEC 10040:1998, International Organization for Standardization, Geneva, Switzerland (November 1998).

[29] DMTF, Common Information Model (CIM) Infrastructure v2.7.0, Specification DSP0004, Distributed Management Task Force (April 2012).
URL `http://www.dmtf.org/sites/default/files/standards/documents/DSP0004_2.7.0.pdf`

[30] K. McCloghrie, D. Perkins, J. Schoenwaelder, J. Case, M. Rose, S. Waldbusser, Structure of Management Information version 2 (SMIv2), Standards Track RFC 2578, IETF (April 1999).
URL `https://tools.ietf.org/html/rfc2578`

[31] ITU, Information technology Open Systems Interconnection Structure of management information: Guidelines for the Definition of Managed Objects, Recommendation X.722, International Telecommunication Union (January 1992).
URL `https://www.itu.int/rec/T-REC-X.722-199201-I`

[32] J. de Vergara, V. Villagra, J. Asensio, J. Berrocal, Ontologies: giving semantics to network management models, Network, IEEE 17 (3) (2003) 15–21. doi:10.1109/MNET.2003.1201472.

[33] ONF, Openflow switch specification v1.5.0, Technical Specification TS-020, Open Network Foundation (December 2014).
URL `https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf`

[34] L. Yang, R. Dantu, T. Anderson, R. Gopal, Forwarding and control element separation (ForCES) framework, Informational 3746, IETF (April 2004).
URL `http://datatracker.ietf.org/doc/rfc3746/`

[35] H. Song, Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined

Networking, HotSDN '13, ACM, New York, NY, USA, 2013, pp. 127–132. doi:10.1145/2491185.2491190.
URL `http://doi.acm.org/10.1145/2491185.2491190`

[36] R. Sherwood, G. Gibb, A. G. Yap, Kok-Kiong, M. Casado, N. McKeown, G. Paraulkar, Flowvisor: A network virtualization layer, Tech. Rep. OpenFlow-tr-2009-1, OpenFlow Team, Standford University (October 2009).
URL `http://www.openflow.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf`

[37] X. Jin, J. Rexford, D. Walker, Incremental update for a compositional sdn hypervisor, in: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, HotSDN '14, ACM, Chicago, Illinois, USA, 2014, pp. 187–192. doi:10.1145/2620728.2620731.
URL `http://doi.acm.org/10.1145/2620728.2620731`

[38] D. Drutskoy, E. Keller, J. Rexford, Scalable network virtualization in software-defined networks 17 (2) (2013) 20–27. doi:10.1109/MIC.2012.144.
URL `http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6362137`

[39] Project Floodlight, Floodlight REST API (2015).
URL `https://floodlight.atlassian.net/wiki/display/floodlightcontroller/Floodlight+REST+API`

[40] M. Monaco, O. Michel, E. Keller, Applying operating system principles to sdn controller design, in: Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks, HotNets-XII, ACM, New York, NY, USA, 2013, pp. 2:1–2:7. doi:10.1145/2535771.2535789.
URL `http://doi.acm.org/10.1145/2535771.2535789`

[41] H. Yin, H. Xie, T. Tsou, D. Lopez, P. Aranda, R. Sidi, SDNi: A message exchange protocol for software defined networks (SDNs) across multiple domains, Internet-Draft (December 2012).
URL `https://tools.ietf.org/html/draft-yin-sdn-sdni-00`

[42] T. Bray, The JavaScript Object Notation (JSON) data interchange format, Standards Track RFC 7159, IETF (March 2014).
URL `https://tools.ietf.org/html/rfc7159`

[43] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, J. Cowan, Extensible Markup Language (XML) 1.1 (second edition), Recommendation, W3C (August 2006).
URL https://www.w3.org/TR/xml11/

[44] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Hypertext Transfer Protocol – HTTP/1.1, Standards Track RFC 2616, The Internet Society (June 1999).
URL https://tools.ietf.org/html/rfc2616

[45] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar, Y. Lafon, Soap version 1.2 part 1: Messaging framework (second edition), Recommendation, W3C (April 2007).
URL https://www.w3.org/TR/2007/REC-soap12-part1-20070427/

[46] R. Enns, M. Bjorklund, J. Schoenwaelder, A. Bierman, Network Configuration Protocol (NETCONF), Standards Track RFC 6241, IETF (June 2011).
URL https://datatracker.ietf.org/doc/rfc6241/

[47] D. Harrington, R. Presuhn, B. Wijnen, An architecture for describing Simple Network Management Protocol (SNMP) management frameworks, Standards Track RFC 3411, The Internet Society (December 2002).
URL https://tools.ietf.org/html/rfc3411

[48] DMTF, Network management profile v1.0.0a, Specification DSP1046, Distributed Management Task Force (April 2014).
URL http://www.dmtf.org/sites/default/files/standards/documents/DSP1046_1.0.0a.pdf

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

[50] D. Kieras, Using the keystroke-level model to estimate execution times, Unpublished report, University of Michigan, [Online]. Available: http://www.ict.griffith.edu.au/marilyn/uidweek10/klm.pdf (2001).
URL http://www.ict.griffith.edu.au/marilyn/uidweek10/klm.pdf

[51] C. R. P. dos Santos, L. Z. Granville, W. Cheng, D. Loewenstern, L. Shwartz, N. Anerousis, Performance management and quantitative modeling of it service processes using mashup patterns, in: Network and Service Management (CNSM), 2011 7th International Conference on, 2011, pp. 1–9.
URL `http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6104022`