Computer Communications 102 (2017) 150–164

Contents lists available at ScienceDirect

Computer Communications

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

A framework for SDN integrated management based on a CIM model and a vertical management plane

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*

ARTICLE INFO

Article history:
Received 12 February 2016
Revised 12 August 2016
Accepted 13 August 2016
Available online 16 August 2016

Keywords:
Common information model
Software-defined networking
Heterogeneous environments
Resource characterization
Configuration management

ABSTRACT

The Software-Defined Networking (SDN) paradigm establishes a typical three-plane architecture (*i.e.*, Data, Control, and Application planes) that facilitates the deployment of network functions and simplifies traditional network management tasks. However, SDN lacks an integrated or standardized framework for managing its architecture. 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 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.

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 [4–7]: *(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 [8], efficient energy usage [9], and access control [10,11]. Furthermore, some studies have addressed the need to manage the SDN architecture, including, for example, frameworks to configure the Data Plane [12,13], to deploy [14,15] and monitor [16,17] the Control Plane, to virtualize SDNs [18,19], and to develop the Application Plane [20,21]. 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

* Corresponding author.
E-mail addresses: festradasolano@unicauca.edu.co (F. Estrada-Solano), jaordonez@fup.edu.co (A. Ordonez), granville@inf.ufrgs.br (L.Z. Granville), omcaicedo@unicauca.edu.co (O.M. Caicedo Rendon).

http://dx.doi.org/10.1016/j.comcom.2016.08.006
0140-3664/© 2016 Elsevier B.V. All rights reserved.

solutions. In our previous work, we evaluated the feasibility of using mashups to control and monitor SDN in heterogeneous environments by composing management applications at the top of the SDN architecture [22–25]. 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 [26–29], nonetheless, they are focused exclusively on specific SDN technology instances or fall short in modeling the SDN architecture. In a later section, we define how these approaches provide an insufficient solution for representing the SDN architecture.

Recent proposals have considered a Management Plane in the SDN architecture for carrying out Operation, Administration, and Maintenance (OAM) functions [4–7]. 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 [30] (*i.e.*, Information, Organizational, Communication, and Functional). Then, we focus on introducing a novel information model that represents the SDN architecture from a management perspective as a Common Information Model (CIM) conceptual framework [31]. We preferred CIM over other information definition languages (*e.g.*, Structure of Management Information [SMI] [32] and Guidelines for the Definition of Managed Objects [GDMO] [33]) because its high expressiveness affords future robust semantic integration [34]. 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]) [26].

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 2, it is discussed both the background and related work. In Section 3, we overview the proposed Management Plane. In Section 4, we introduce our CIM-based information model. In Section 5, we present a case study used to evaluate the proposed approach. The paper concludes in Section 6.

2. Background

In this section, first, we present the SDN architecture. Second, we discuss the related work about the SDN management.

[1] ONF-CIM is not based on the CIM specification defined by the Distributed Management Task Force (DMTF).

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 [4–7] 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 [35] 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 [36] 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 [36] and Protocol-Oblivious Forwarding (POF) [37].

The Control Plane comprises Network Slicers (NetSlicer) and Network Operating 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 [38] 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 [35] and ForCES Control Elements (CE) [36] are NOS instances. Although NetSlicers can be considered as a specific NOS, it is important to describe them as separate components in order to demarcate their functionality: network virtualization versus decision making. In addition, some approaches may provide network virtualization as an NOS service for multiple tenant NetApps [39,40].

As aforementioned, the Control Plane provides NBIs to the Application Plane. An NBI encompasses common Application Programming Interfaces (API) based on NOS native bundles (*e.g.*, Floodlight Java API [41] and Ryu application API [42]), programming languages (*e.g.*, Pyretic [20] and Procera [21]), protocols (*e.g.*, Floodlight REST API [43]), file systems (*e.g.*, YANC [44]), among others. The Control Plane also defines EWBIs to enable information exchange between NOSs distributed in different domains. For example, the SDN Inter-networking (SDNI) Negotiation Interface, the West-East Bridge (WE-Bridge) [45], and the ForCES CE-CE interface [36].

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 [8] to better redistribute traffic in SDN networks, ElasticTree [9] to efficiently provide energy for SDN components, and Resonance [10] and OpenRoads [11] 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) [12] and Open vSwitch Database (OVSDB) [13] that define protocols to configure NetDevs, *(ii)* Kandoo [14] and HyperFlow [15] to scale and distribute NOSs, *(iii)* OpenFlow Management Infrastructure (OMNI) [16] and ROVIZ [17] that provide graphic interfaces to monitor NOSs, *(iv)* VeRTIGO [18] and ADVisor [19] to configure NetSlicers; and *(v)* Pyretic [20] and Procera [21] 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 [22–25]. 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 architecture from a management perspective: *(i)* ONF-CIM [26] defines a Core Information Model (CoreModel) [27] that uses the Unified Modeling Language (UML) to structure the forwarding functions of the Data Plane, *(ii)* Network Abstraction Model (NAM) [28] employs a building block approach to represent all resources of NetDevs; and *(iii)* CIM-SDN [29] proposes a CIM extension schema to model SDN. It is worth noting that these *information models* fall short in representing the SDN architecture or are tied to specific SDN technology instances. ONF CoreModel describes only the Data Plane and was designed for OpenFlow. NAM focuses on NetDevs and is extensible enough to represent different functionalities of the SDN architecture, however, it is deeply tied to the ForCES FE Model [46]. 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 complete, technology-agnostic reference model of the SDN architecture in order to achieve integrated management in heterogeneous environments.

3. A Management Plane for SDN

To define our approach, we extend the Management Plane concept considered by recent SDN proposals for covering OAM functions omitted and restricted in the traditional SDN architecture [4–7]. 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 NOSs 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 of heterogeneous SDN resources. To better explain our approach, we present a high-level overview of the Management Plane using the four OSI submodels [30]: Information, Organizational, Communication, and Functional.

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.

Table 1
Proposals on SDN management.

References	Approach	Requirements for SDN management			
		Complete architecture	Integrated management	Heterogeneous environment	Information model
[8]–[11]	Functions in NetApps				
[12]–[21]	Isolated tools	✓			
[22]–[25]	Mashup based		✓	✓	
[26]–[29]	Information models		✓		✓
–	This approach	✓	✓	✓	✓

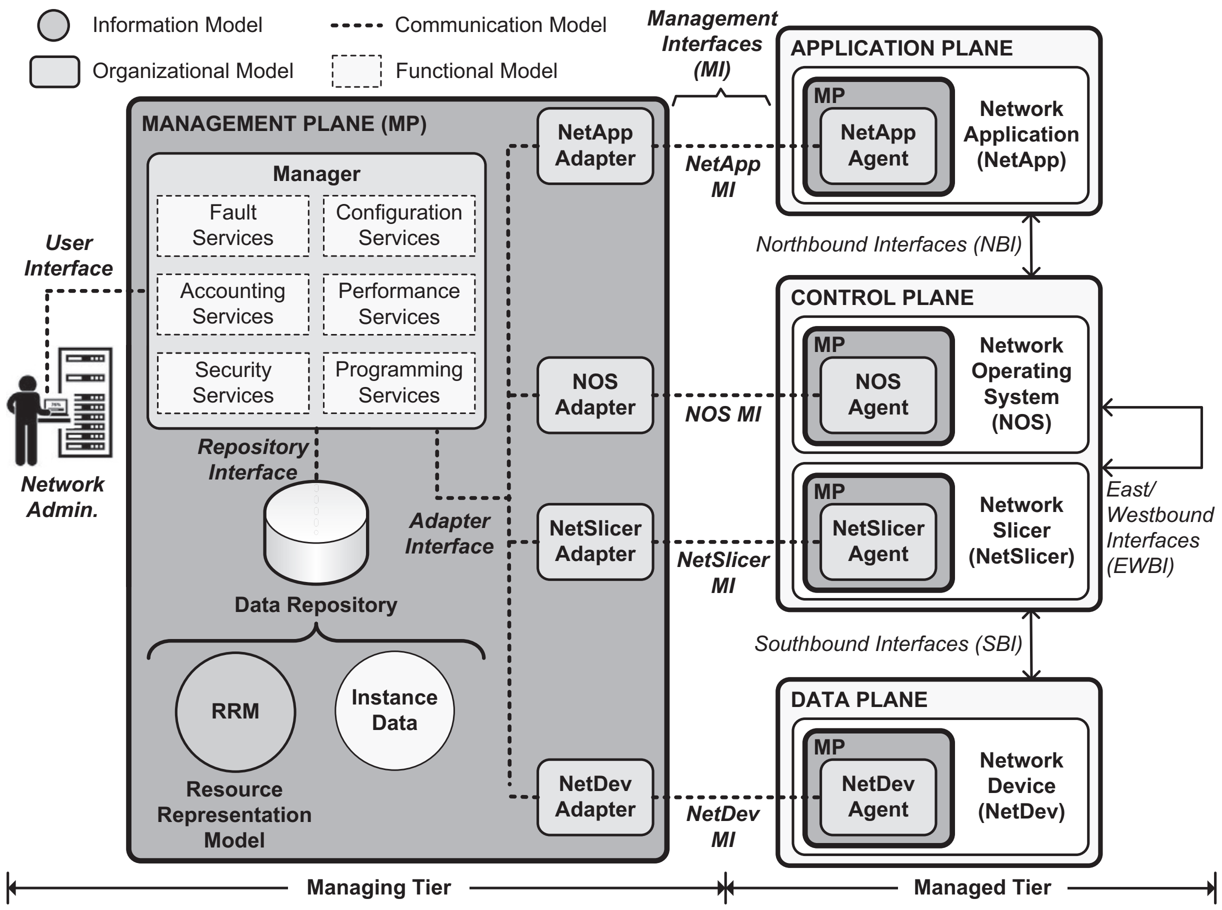

Fig. 1. High-level SDN architecture with Management Plane.

It is important to highlight that we define the Management Plane by referencing the four OSI network management submodels [30]: *(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.

3.2. 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 architecture from a management perspective 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. Both of these 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] [47] and eXtensible Markup Language [XML] [48]) and a standardized protocol (*e.g.*, HyperText Transfer Protocol [HTTP] [49] and Simple Object Access Protocol [SOAP] [50]) 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 [13], Network Configuration Protocol [NETCONF] [51], and Simple Network Management Protocol [SNMP] [52]).

3.2.4. Functional model

As considered by recent Management Plane approaches for SDN [5,7], this proposal also references the five OSI management functional areas [53] 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 by Wickboldt et al. [4].

4. Information model

As aforementioned, our Management Plane requires an information model that provides a technology-agnostic and consistent abstraction of the SDN architecture to enable integrated management. Few approaches provide models that attempt to characterize the SDN architecture from a management perspective [12,28,29], 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 [32] and GDMO [33]), affording future robust semantic integration [34]. CIM supplies several classes, associations, properties, and methods to describe network resources at a conceptual level, such as Ethernet ports, LAN endpoints, and VLANs [54,55]. However, CIM lacks elements that represent specific SDN features for management [29]. Thus, our information model introduces new elements that extend the actual CIM Schema to characterize the SDN architecture from a management perspective. 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. Although CIM schemas can be considered as a Data Model, their graphical notation support (*i.e.*, UML) provides a nearby approach to an information model [56]. In fact, the extended CIM schema described in this paper includes important aspects of information models: the independence of particular implementations or protocols and the relationships between managed objects.

In next paragraphs and figures (Figs. 2–5) 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. It is noteworthy that this class schema represents the main contribution of this paper, particularly the new classes and the new 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

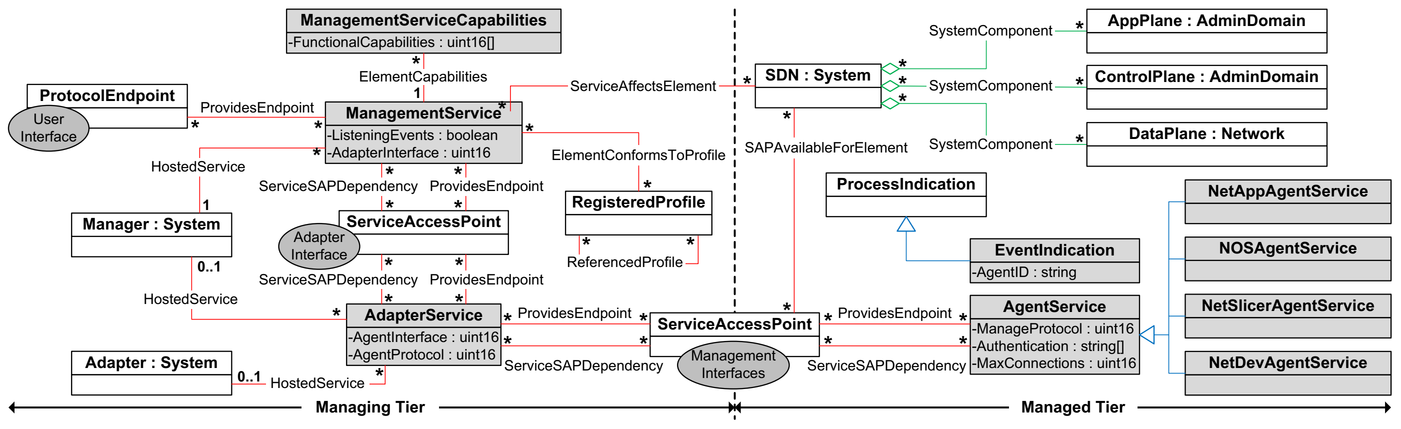

Fig. 2. Class schema to model SDN Management Plane.

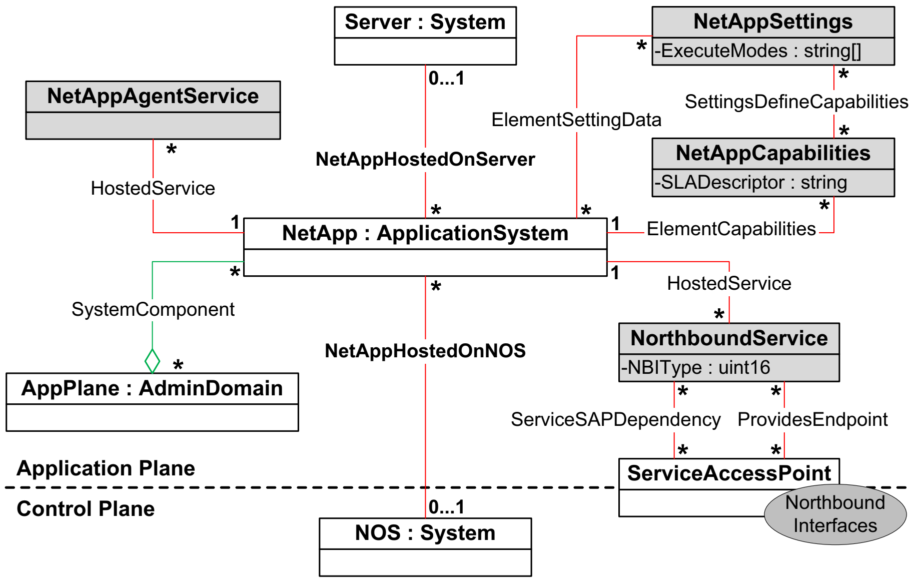

Fig. 3. Class schema to model SDN Application Plane.

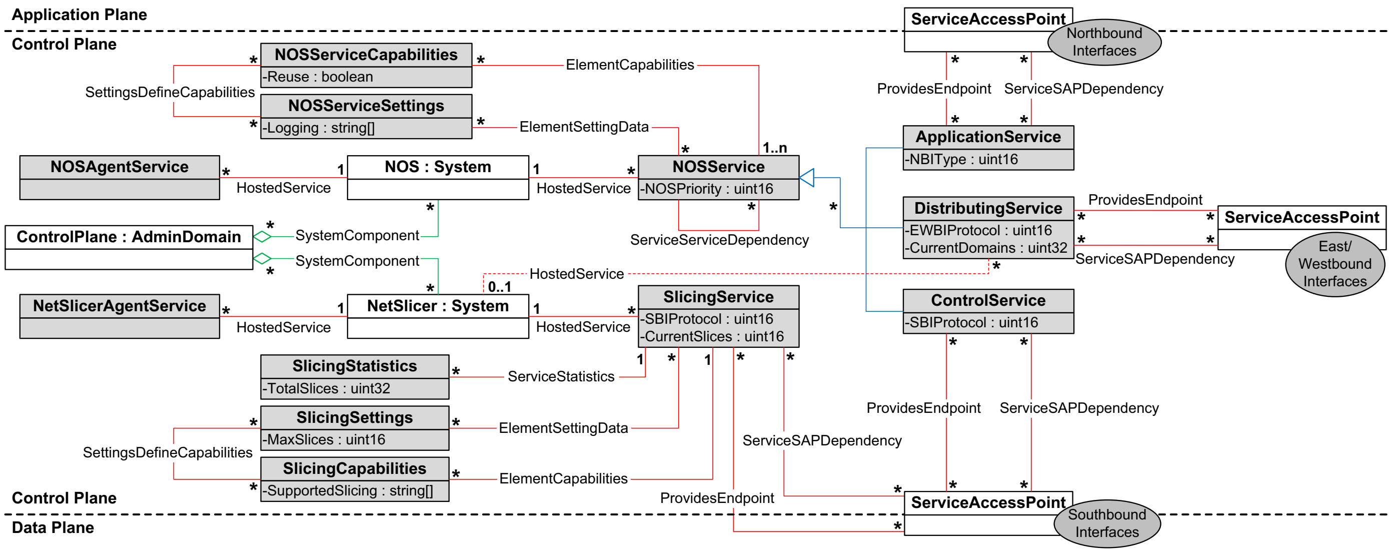

Fig. 4. Class schema to model SDN Control Plane.

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 *ElementCapabilities* association, the *ManagementServiceCapabilities* class describes both supported and excluded abilities for Management Services. For example, the property *FunctionalCapabilities* establishes a numeric value map based on the Functional Model for classifying SDN Management Services as Fault, Configuration, Accounting, Performance, Security, or Programming services (see Section 3.2.4). Thus, a Management Service that modifies the SBI communication of NetDevs may declare 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 con-

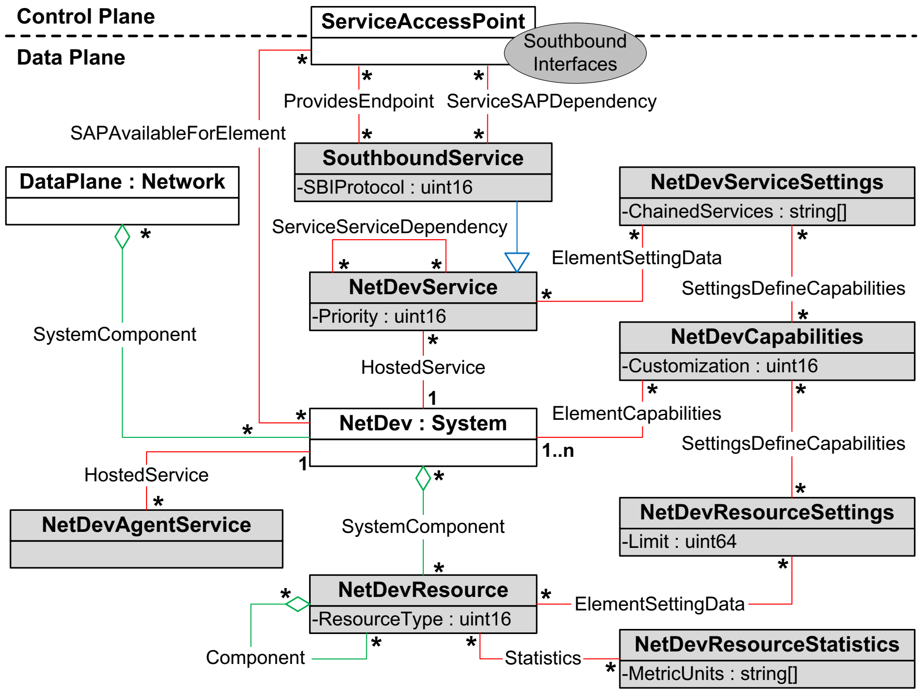

Fig. 5. Class schema to model SDN Data Plane.

nected to the adapter *ServiceAccessPoint* indicate that both the *ManagementService* and the *AdapterService* establish their own communication points to allow access from the other. The *Service-SAPDependency* 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*, such as the property *ListeningEvents* that specifies if the *ManagementService* handles event notifications and the properties *AdapterInterface* and *ManageInterface* that identify the access APIs provided by the *ManagementService* and the *AdapterService*, respectively. 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. Besides the aforementioned property *ManageInterface*, the *AdapterService* also provides the property *AgentProtocol* that identifies the communication protocol used to interact with a specific *AgentService*. 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*. Both the property *AgentProtocol* from the *AgentService* and the property *ManageProtocol* from the *AdapterService* define the communication protocol. 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*. Besides the above described property *ManageProtocol*, the *AgentService* also provides the property *Authentication* that declares the access parameters (*e.g.*, login and password) and the property *MaxConnections* that defines the maximum number of concurrent connections supported. 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. The property *AgentID* of the *EventIndication* class identifies the instance of *AgentService* that generates the notification. 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 SDN architecture, 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 SDN architecture.

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 NOS native bundles (*e.g.*, Floodlight and Ryu APIs), programming languages (*e.g.*, Pyretic and Procera), and protocols (*e.g.*, REST). The property *NBIType* from the *NorthboundService* identifies the API type used for the communication through the northbound *ServiceAccessPoint*.

The *NetAppHostedOnNOS* association between the *NetApp* and the *NOS* represents local NetApps running on NOSs. Usually, these local NetApps utilize NBIs based on NOS native bundles and 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. For instance, the property *SLADescriptor* allows to describe the service level policies of a specific *NetApp*. The *NetAppSettings* class establishes configuration parameters for the *NetApp*, such as the property *ExecuteModes* that defines the different execution modes supported by a particular *NetApp*. The *ElementSettingData* association depicts the relationship 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 *NOSService*, the *NOSServiceCapabilities*, and the *NOSServiceSettings*.

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. The property *NOSPriority* from the *NOSService* designates the order at which the *NOS* processes the instances of the different hosted services. 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*. Its property *NBIType* specifies the provided API for the northbound communication. The *DistributingService* defines services that enable to deploy a distributed Control Plane across distinct domains through the east/westbound *ServiceAccessPoint*. This class includes the property *EWBIProtocol* that declares the protocol for the east/westbound communication and the property *CurrentDomains* that indicates the number of different domains currently handled by the *DistributingService*. The *ControlService* describes services that handle the communication with NetDevs and NetSlicers through the southbound *ServiceAccessPoint*. Its property *SBIProtocol* specifies the protocol for the southbound communication.

The *ServiceServiceDependency* association reflects 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 Negotiation Interface and the WE-Bridge mechanism. 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, like the property *Reuse* that specifies the reusability of the *NOSService*. The *ElementCapabilities* association between the *NOSServiceCapabilities* and the *NOSService* reflects this relationship. The *NOSServiceSettings* class defines the configuration parameters for NOSs services, such as the property *Logging* that allows to set the method for storing data generated by the *NOSService*. 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 for sending discovery messages updates the behavior of a tracking service.

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 *SlicingService* includes the property *SBIProtocol* that describes the protocol for the southbound communication and the property *CurrentSlices* that reports the number of slices currently operated by a particular NetSlicer. 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 property *TotalSlices* from the *SlicingStatistics* reports 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*, like the property *SupportedSlicing* that indicates the different slicing methods defined by a NetSlicer. 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 property *MaxSlices* declares 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*.

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 the *NetDevCapabilities*. The *ElementCapabilities* association between the *NetDev* and the *NetDevCapabilities* indicates this relationship. For example, the property *Customization* defines the degree of personalization for an instance of *NetDev*. An OpenFlow switch declares low customization capabilities because it provides simple forwarding functions based on match/action flow tables. A NetFPGA programmable hardware exposes higher customization capabilities because it offers complex plugging modules that enable to build specific functions. In addition, 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, like ports, queues, and tables. The *SystemComponent* association between the *NetDev* and the *NetDevResource* implies this aggregation. The latter includes the property *ResourceType* that allows to identify the type of network element composing the *NetDev*. Our schema presents the *NetDevResource* as a superclass from which individual subclasses inherit to represent NetDev components. For example, a subclass called *FlowTable* for characterizing 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 *NetDevResourceStatistics* defines arbitrary collections of statistical information applicable to instances of the *NetDevResource*. The *NetDevResourceStatistics* is a subclass that derives from the *StatisticalData*. The *Statistics* association attaches the *NetDevResourceStatistics* with the *NetDevResource*. For example, ports in switches delineate transmission metrics, such as received and transmitted bytes, packets, and errors. The property *MetricUnits* from the *NetDevResourceStatistics* declares the units of measurement used by a network element for statistical data.

The *NetDevResourceSettings* class describes the configuration of network elements that compose NetDevs. The *ElementSettingData* association between the *NetDevResourceSettings* and the *NetDevResource* reflects this relationship. The *SettingsDefineCapabilities* association between the *NetDevResourceSettings* and the *NetDevCapabilities* indicates that the configuration parameters of NetDev components specify some capabilities of NetDevs. For example, the property *Limit* establishes boundaries for NetDev components, such as the highest speed operation of ports and the maximum buffer size of queues.

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*. The property *Priority* from the *NetDevService* indicates the preference for processing a group of services hosted by a 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 property *SBIProtocol* may specify that the southbound communication uses the OpenFlow protocol, which defines a secure channel in switches for communicating with external controllers and updating 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 property *ChainedServices* allows to delineate the sequence of services that forms a specific *NetDevService*. In particular, a subclass from the *NetDevServiceSettings* may include properties for establishing the routing algorithm of a route calculation service or 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.

The presented information model leverages the extensibility of CIM schemas to allow vendors and providers to represent specific implementation features of SDN resources. As above described, a specific implementation model must inherit from the classes defined in the proposed class schema. Furthermore, network designers may use or extend our information model to represent management requirements of new network paradigms seamlessly together with the functionality described in the proposed class schema. For example, network functions from Network Function Virtualization (NFV) can be successfully adapted to the conceptual model of NetDevs from SDN [28]. Finally, since the proposed class schema follow CIM conventions, our information model may include elements from the existing CIM Schema to represent other managed resources, such as legacy networks [54] and virtual networking [55].

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 purchases SDN forwarding resources from *Vendor A*. However, at some point in time, the Network Administrator decided to buy NetDevs from *Vendor B* because it offered a better benefit-cost ratio than *Vendor A*. As the network became bigger and since the implementations of SDN resources are constantly updated, 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.

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 architecture from a management perspective 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.

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

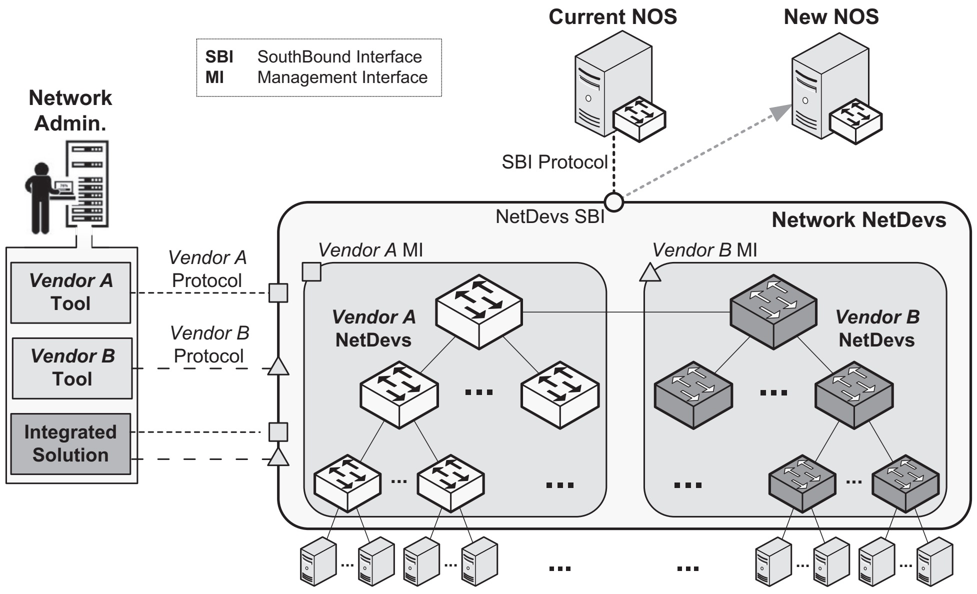

Fig. 6. Scenario: configuring NOS of heterogeneous NetDevs.

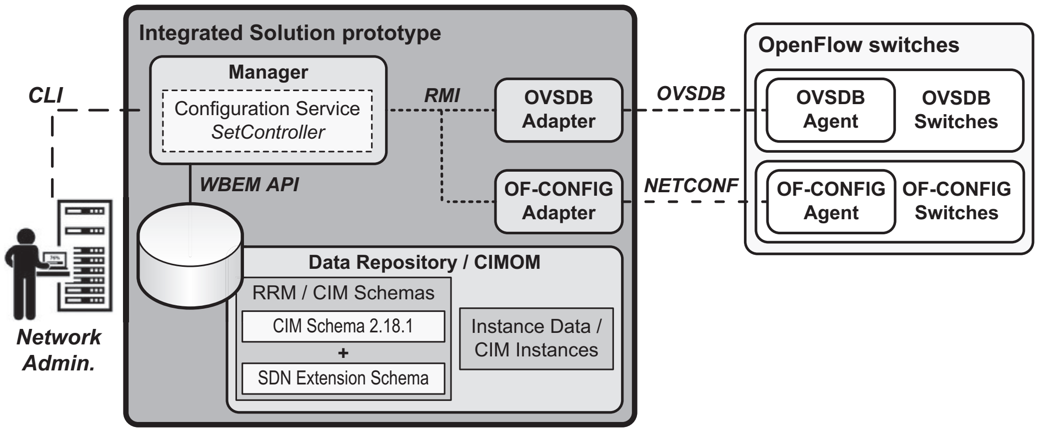

Fig. 7. Integrated solution prototype.

(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 one 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

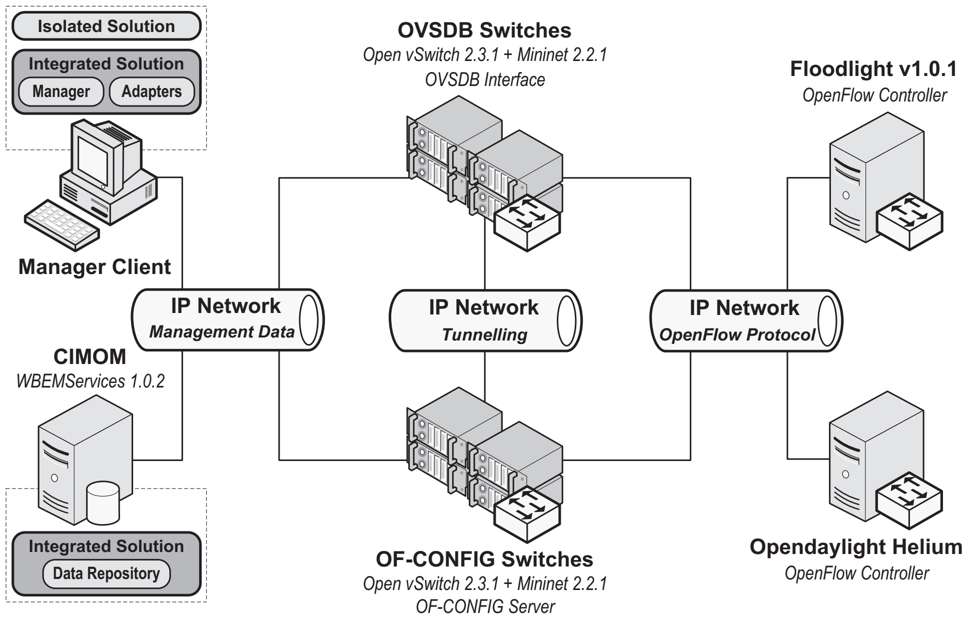

Fig. 8. Test environment.

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 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 ($r > 10$) [57], the time-response results reveal: *(i) SetController* of both the *Isolated Solution* and the *Integrated Solution* has an admissible r that grows

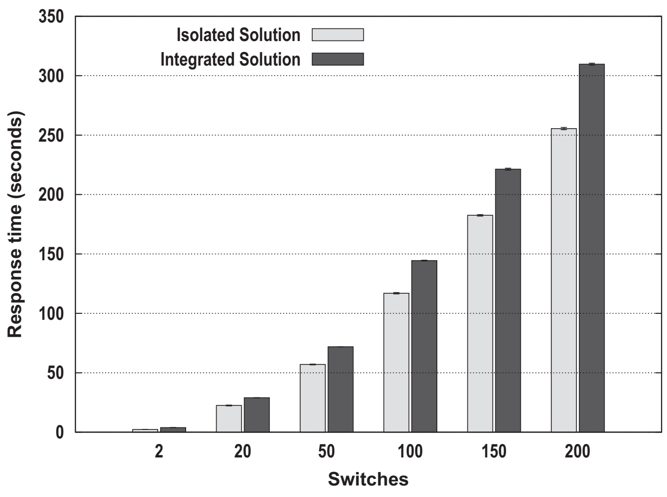

Fig. 9. Configuring evaluation: time-response.

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.

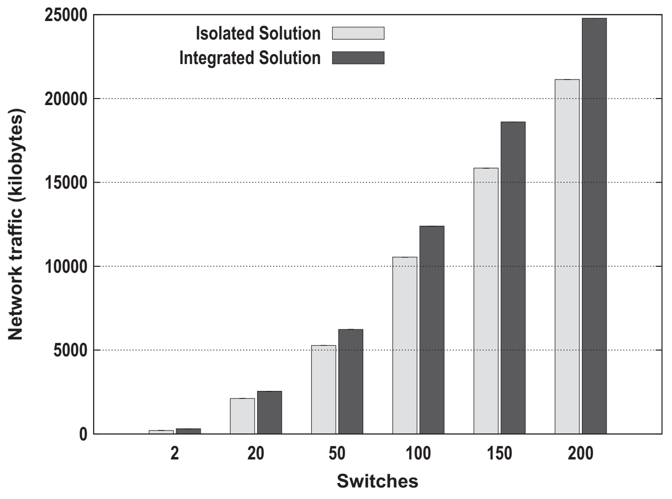

Fig. 10. Configuring evaluation: network traffic.

To compute the time composing overhead of the *Isolated Solution* ($Tc_{oh:isolated}$), we use the Keystroke-Level Model (KLM) [58] 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 [24,59]. 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 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, $Tc_{oh:isolated}$ 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 resources, 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 managed SDN architecture. 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*.

$$\begin{aligned}
Tt_{integrated} &= Tr_{integrated} + Tc_{integrated}\\
Tt_{isolated} &= Tr_{isolated} + Tc_{isolated}\\
Tr_{integrated} &= Tr_{oh:integrated} + Tr_{isolated}\\
Tc_{isolated} &= Tc_{oh:isolated} + Tc_{integrated}\\
Tt_{integrated} &= Tr_{isolated} + Tc_{isolated} + Tr_{oh:integrated} - Tc_{oh:isolated}\\
Tt_{integrated} &= \begin{cases} Tt_{isolated} - 0.4 - 0.55N_{sw}, & N_{sw} < 20\\ Tt_{isolated} - 0.4 - N_{sw}, & N_{sw} \geq 20 \end{cases} \qquad (1)
\end{aligned}$$

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 resources 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 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 to perform SDN management operations in heterogeneous environments, 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 all the SDN architecture 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 architecture from the management perspective. It is possible because the information model relies on CIM to provide a technology-agnostic and consistent characterization of SDN. Although CIM schemas might be complex to understand, the working groups from DMTF continuously develop management profiles for clarifying how to use the classes and associations from CIM for specific areas of management. Thus, our proposed CIM schema provides a reference to DMTF for building a management profile that clearly explains how to characterize managing functions in the SDN architecture. 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 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 architecture from a management perspective 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 [57], 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 SDN resources.

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. 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.

[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, in: AINTEC '11, ACM, New York, NY, USA, 2011, pp. 160–163, doi:10.1145/2089016.2089044.

[3] H. Kim, N. Feamster, Improving network management with software defined networking, Commun. Mag. 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, Commun. Mag. IEEE 53 (1) (2015) 278–285, doi:10.1109/MCOM.2015.7010546.

[5] E. Haleplidis, K. Pentikousis, S. Denazis, J.H. Salim, D. Meyer, O. Koufopavlou, Software-Defined Networking (SDN): Layers and Architecture Terminology, Informational, RFC 7426, IETF, 2015. URL https://tools.ietf.org/html/rfc7426.

[6] ONF, SDN AArchitecture V1.0, Technical reference TR-502, Open Network Foundation, 2014. URL https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf.

[7] ITU, Framework of Software-Defined Networking, Recommendation Y.3300, International Telecommunication Union, 2014. URL https://www.itu.int/rec/T-REC-Y.3300-201406-I.

[8] 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, in: Hot-ICE'11, USENIX Association, Berkeley, CA, USA, 2011. 12–12. URL http://dl.acm.org/citation.cfm?id=1972422.1972438.

[9] 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, in: NSDI'10, USENIX Association, Berkeley, CA, USA, 2010. 17–17. URL http://dl.acm.org/citation.cfm?id=1855711.1855728.

[10] 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, in: WREN '09, ACM, New York, NY, USA, 2009, pp. 11–18, doi:10.1145/1592681.1592684.

[11] 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, in: WINTECH '09, ACM, New York, NY, USA, 2009, pp. 59–66, doi:10.1145/1614293.1614304.

[12] 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.

[13] B. Pfaff, B. Davie, The Open vSwitch Database Management Protocol, Informational RFC 7047, IETF, 2013. URL https://tools.ietf.org/html/rfc7047.

[14] 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, in: HotSDN '12, ACM, New York, NY, USA, 2012, pp. 19–24, doi:10.1145/2342441.2342446.

[15] 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, in: INM/WREN'10, USENIX Association, Berkeley, CA, USA, 2010. 3–3. URL http://dl.acm.org/citation.cfm?id=1863133.1863136.

[16] 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.

[17] S. Natarajan, X. Huang, An interactive visualization framework for next generation networks, in: Proceedings of the ACM CoNEXT Student Workshop, in: CoNEXT '10 Student Workshop, ACM, New York, NY, USA, 2010, pp. 14:1–14:2, doi:10.1145/1921206.1921222.

[18] 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.

[19] 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.

[20] 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.

[21] 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, in: HotSDN '12, ACM, New York, NY, USA, 2012, pp. 43–48, doi:10.1145/2342441.2342451.

[22] O.M.C. Rendon, C.R.P. dos Santos, A.S. Jacobs, L.Z. Granville, Monitoring virtual nodes using mashups, Comput. Netw. 64 (0) (2014) 55–70. http://dx.doi.org/10.1016/j.comnet.2014.02.007. URL http://www.sciencedirect.com/science/article/pii/S1389128614000498.

[23] 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.

[24] 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.

[25] 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.

[26] ONF, Common Information Model (CIM) Overview, Technical Reference TR-513, Open Network Foundation, 2015. URL https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/Common_Information_Model_CIM_Overview_1.pdf.

[27] ONF, Core Information Model (CoreModel) v1.1, Technical Reference TR-512, Open Network Foundation, 2015. URL https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/ONF-CIM_Core_Model_base_document_1.1.pdf.

[28] E. Haleplidis, J. Hadi Salim, S. Denazis, O. Koufopavlou, Towards a network abstraction model for SDN, J. Netw. Syst. Manage. 23 (2) (2015) 309–327, doi:10.1007/s10922-014-9319-3.

[29] 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.

[30] ISO, Information Technology – Open Systems Interconnection – Systems Management Overview, ISO/IEC 10040:1998, International Organization for Standardization, Geneva, Switzerland, 1998.

[31] DMTF, Common Information Model (CIM) Infrastructure v2.7.0, Specification DSP0004, Distributed Management Task Force, 2012. URL http://www.dmtf.org/sites/default/files/standards/documents/DSP0004_2.7.0.pdf.

[32] K. McCloghrie, D. Perkins, J. Schoenwaelder, J. Case, M. Rose, S. Waldbusser, Structure of Management Information version 2 (SMIv2), Standards Track RFC 2578, IETF, 1999. URL https://tools.ietf.org/html/rfc2578.

[33] ITU, Information Technology Open Systems Interconnection Structure of Management Information: Guidelines for the Definition of Managed Objects, Recommendation X.722, International Telecommunication Union, 1992. URL https://www.itu.int/rec/T-REC-X.722-199201-I.

[34] J. de Vergara, V. Villagra, J. Asensio, J. Berrocal, Ontologies: giving semantics to network management models, Netw. IEEE 17 (3) (2003) 15–21, doi:10.1109/MNET.2003.1201472.

[35] ONF, OpenFlow Switch Specification v1.5.0, Technical Specification TS-020, Open Network Foundation, 2014. URL https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf.
[36] L. Yang, R. Dantu, T. Anderson, R. Gopal, Forwarding and Control Element Separation (ForCES) Framework, Informational RFC 3746, IETF, 2004. URL http://datatracker.ietf.org/doc/rfc3746/.
[37] 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, in: HotSDN '13, ACM, New York, NY, USA, 2013, pp. 127–132, doi:10.1145/2491185.2491190.
[38] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, G. Paraulkar, FlowVisor: A Network Virtualization Layer, Technical Report OpenFlow-tr-2009-1, OpenFlow Team, Standford University, 2009. URL http://www.openflow.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf.
[39] 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, in: HotSDN '14, ACM, Chicago, Illinois, USA, 2014, pp. 187–192, doi:10.1145/2620728.2620731.
[40] D. Drutskoy, E. Keller, J. Rexford, Scalable network virtualization in software-defined networks, IEEE Internet Comput. 17 (2) (2013) 20–27, doi:10.1109/MIC.2012.144. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6362137.
[41] Project Floodlight, The floodlight controller, [Accessed: June 16, 2016]. URL https://floodlight.atlassian.net/wiki/display/floodlightcontroller/The+Controller.
[42] Ryu Project Team, Ryu application API, [Accessed: June 16, 2016]. URL https://ryu.readthedocs.io/en/latest/ryu_app_api.html.
[43] Project Floodlight, Floodlight REST API, [Accessed: June 16, 2016]. URL https://floodlight.atlassian.net/wiki/display/floodlightcontroller/Floodlight+REST+API.
[44] 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, in: HotNets-XII, ACM, New York, NY, USA, 2013, pp. 2:1–2:7, doi:10.1145/2535771.2535789.
[45] P. Lin, J. Bi, S. Wolff, Y. Wang, A. Xu, Z. Chen, H. Hu, Y. Lin, A west-east bridge based SDN inter-domain testbed, IEEE Commun. Mag. 53 (2) (2015) 190–197, doi:10.1109/MCOM.2015.7045408.
[46] J. Halpern, J.H. Salim, Forwarding and Control Element Separation (ForCES) Forwarding Element Model, Standards Track RFC 5812, IETF, 2010. URL https://tools.ietf.org/html/rfc5812.
[47] T. Bray, The JavaScript Object Notation (JSON) Data Interchange Format, Standards Track RFC 7159, IETF, 2014. URL https://tools.ietf.org/html/rfc7159.
[48] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, F. Yergeau, J. Cowan, Extensible Markup Language (XML) 1.1 (Second Edition), Recommendation, W3C, 2006. URL https://www.w3.org/TR/xml11/.
[49] 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, IETF, 1999. URL https://tools.ietf.org/html/rfc2616.
[50] 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, 2007. URL https://www.w3.org/TR/2007/REC-soap12-part1-20070427/.
[51] R. Enns, M. Bjorklund, J. Schoenwaelder, A. Bierman, Network Configuration Protocol (NETCONF), Standards Track RFC 6241, IETF, 2011. URL https://datatracker.ietf.org/doc/rfc6241/.
[52] D. Harrington, R. Presuhn, B. Wijnen, An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks, Standards Track RFC 3411, IETF, 2002. URL https://tools.ietf.org/html/rfc3411.
[53] ITU, TMN Management Functions, Recommendation M.3400, International Telecommunication Union, 2000. URL https://www.itu.int/rec/T-REC-M.3400-200002-I.
[54] DMTF, Network Management Profile v1.0.0b, Specification DSP1046, Distributed Management Task Force, 2015. URL http://www.dmtf.org/sites/default/files/standards/documents/DSP1046_1.0.0b.pdf.
[55] DMTF, Virtual Networking Management, White Paper DSP2025, Distributed Management Task Force, 2012. URL http://www.dmtf.org/sites/default/files/standards/documents/DSP2025_1.0.0.pdf.
[56] A. Pras, J. Schoenwaelder, On the Difference between Information Models and Data Models, Informational RFC 3444, IETF, 2003. URL https://tools.ietf.org/html/rfc3444.
[57] S. Joines, R. Willenborg, K. Hygh, Performance Analysis for Java Websites, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
[58] 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.
[59] 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.