IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 19, 2020, accepted October 17, 2020, date of publication October 23, 2020, date of current version November 9, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3033486

Enhancing Orchestration and Infrastructure Programmability in SDN With NOTORIETY

KEVIN BARROS COSTA[1,2], FELIPE S. DANTAS SILVA[1,2], (Member, IEEE), LUCAS M. SCHNEIDER[1], EMÍDIO PAIVA NETO[1,2], AUGUSTO VENANCIO NETO[2,3], (Member, IEEE), AND FLAVIO ESPOSITO[4]

[1]LaTARC Research Laboratory, Federal Institute of Education, Science and Technology of Rio Grande do Norte (IFRN), Natal 59015-000, Brazil
[2]Graduate Program in Systems and Computing (PPgSC), Department of Informatics and Applied Mathematics (DIMAp), Federal University of Rio Grande do Norte (UFRN), Natal 59078-970, Brazil
[3]Instituto de Telecomunicações, 3810-164 Aveiro, Portugal
[4]Department of Computer Science, Saint Louis University, Saint Louis, MO 63103, USA

Corresponding author: Felipe S. Dantas Silva (felipe.dantas@ifrn.edu.br)

This work was supported in part by the Brazilian National Council for Scientific and Technological Development (CNPq) through the PIBITI Program; in part by the H2020 4th EU-BR Collaborative Call [Novel Enablers for Cloud Slicing (NECOS)], funded by the European Commission and the Brazilian Ministry of Science, Technology, Innovation, and Communication (MCTIC), through Rede Nacional de Ensino e Pesquisa (RNP) and Centro de Pesquisa e Desenvolvimento em Tecnologias Digitais para Informação e Comunicação (CTIC), under Grant 777067; and in part by the NSF under Award CNS1647084, Award CNS1836906, and Award CNS1908574.

ABSTRACT Software-Defined Networking (SDN) controllers are nowadays expected to manage large infrastructures and services in a practical, efficient, and optimized way. To achieve such control, SDN networks should increase the automation of orchestration functions to fullfil both service and network management lifecycles. However, network management applications have stringent resource demands that orchestrators need to honor. Most of the available orchestrators focus on high-level approaches that rely exclusively on the Northbound API (NB-API). A few strategies have also been designed to optimize the controllers' internal mechanisms through features enabled by the Southbound API (SB-API). However, such solutions do not focus on optimal information delivery to the orchestrator, a crucial property to provide prompt feedback in response to network events, and to potentially drive self-healing and auto-scaling properties. To address such need, we present NOTORIETY, an orchestration system that provides an abstraction for real-time SDN network controller event message handling. NOTORIETY is able to maximize the orchestration capabilities of SDN controllers including innovative features for processing and delivering network event information. NOTORIETY's design consists of mechanisms that empower SDN-controlled entities by applying filtering rules to efficiently control and optimize data flows provided by the SDN controller. We implemented a testbed to assess our NOTORIETY proposal following the guidelines provided in the RFC 8456, and we benchmark its performance over three SDN controllers (OpenDaylight, ONOS, and Floodlight). The evaluation results reveal the effectiveness of NOTORIETY in reducing the execution time of SDN requests, load processing, and overall data marshaling volume in several scenarios.

INDEX TERMS Software-defined networking, SDN controller, orchestration, network programmability.

I. INTRODUCTION

The advent of innovative paradigms such as Software-Defined Networking (SDN) [1] has changed the way computer network architectures and their services implement and deliver applications [2], providing flexibility and agility. In particular, network programmability through the use of open communication interfaces has allowed the creation and management of customized functionality, e.g., detection and reaction to meaningful network events [3]. Such evolution enabled functionalities such as topology management or traffic control mechanisms that increased performance efficiency and programmability of the underlying network infrastructure [4].

The adoption of the SDN paradigm in scenarios consisting of emerging technologies, such as Information-Centric Networks (ICN) [5], Cloud Computing [6], Internet of Things (IoT) [7], 5G [8], among others, has proved to be a key factor in making it possible to meet the stringent requirements of each provided service (e.g., ultra-low latency, reliability,

The associate editor coordinating the review of this manuscript and approving it for publication was Kashif Sharif.

and availability). In these challenging scenarios, the SDN orchestration capabilities mean the perspectives provided by the SDN approach by employing flexibility and dynamicity by reducing the complexity imposed by several control plane technologies [9].

The facilities that emerged through network softwarization [10] have paved the way for Network Service Providers (NSPs) to make their infrastructures more flexible, with a high degree of scalability and elasticity, as well as offering reactive mechanisms tailored to orchestrating the infrastructure [11]. Additionally, another advantage of this approach is that it can lead to a reduction of Capital Expenses (CAPEX) and Operating Expenses (OPEX) [12], and can enable new networking services and applications to achieve a better time-to-market [13].

The related literature highlights several cases of NSPs adoption, along with other paradigms and emerging technologies. In [14], a self-adaptive and latency-sensitive system is outlined, which uses state-of-the-art topology information to assist in the intelligent management of network services provided by the controllers – such as Quality of Service (QoS) policy managers and traffic analyzers – as well as to fulfill the minimum end-to-end communications latency requirements for 5G scenarios. Another study [15] introduces a system for real-time error correction in multimedia transmissions through machine learning techniques and traffic analysis applications provided by SDN controllers.

However, a key factor that is addressed by some of the related studies concerns the overload on SDN controllers, particularly when they are faced with scenarios where there is a high degree of complexity and granularity in the devices that can sometimes jeopardize the availability of the network infrastructure [16]. In regards of scenarios considered to be latency-sensitive, changing the topology or layout in real-time (e.g., when the failure of a given link causes it), together with the SDN controller overload, may result in delaying the delivery of data that is crucial to the optimal functioning of applications running atop, meaning those that yield intelligence at the control of the network operation and management [1], [17].

A recent survey of the literature shows that many works seek to provide greater reliability in terms of availability and performance for SDN-based ecosystems. However, most studies only concentrate on higher-level solutions and rely on NB-APIs provided by the controller to develop applications that seek to optimize the control plane through particular strategies, such as load balancing [18], controller placement [19], and network orchestration [20].

A. PROBLEM STATEMENT

There is a gap in the literature that has not been widely explored by previous studies, which concerns the optimization of network event delivering at the southbound level, seeking to increase network efficiency, allowing less volume of receiving data, and, consequently, reduced processing time.

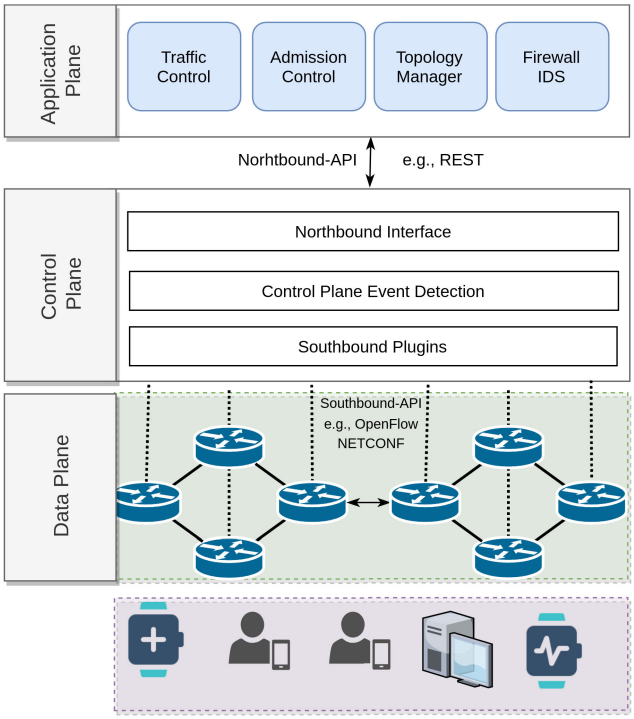


FIGURE 1. SDN architecture abstraction detailed by the main layers, interfaces and standard protocols.

Currently, widely-used SDN controller solutions (i.e., OpenDaylight, Floodlight, and ONOS) proceed by delivering all incoming message content to the applications running atop. This approach imposes all atop applications the challenging task of selecting the incoming content and then triggering the appropriate target procedure. The papers [21] and [22] highlight the importance of performing optimizations on internal components of SDN controllers to achieve improved performance. These optimizations, which generally take place at the level of algorithms and data structures involving intrinsic procedures for control operations, are essential to maintain data consistency and ensure higher levels of performance by the remote applications and plugins developed for the northbound (NB-API) and southbound interfaces (SB-API), respectively. Figure 1 shows, in a high-level scheme, the SDN architecture abstraction, detailing the main layers, interfaces, and standard protocols [1].

The control plane and the data plane communicates via the SB-API, which enables the identification of new events on the network and the installation and setup of flow rules on the devices. In the topmost layer, the Application Plane communicates with the control plane through the NB-API, making it possible to extend the network's programmability capabilities to applications such as firewalls, admission control, and traffic control systems [23].

The current state-of-the-art does not act as a preset for developing solutions designed to optimize controllers' internal optimization. Having this in mind, different types of control applications, which are those that reside in the Control Plane [1] (e.g., orchestrators [24], traffic control mechanisms [25] or load balancing [18]), that need constant access to

network event data so that operating effectively, find that the optimization levels of algorithms and internal mechanisms of SDN controllers [26] have a direct influence on their levels of performance.

In light of this, we claim that the employment of new methods capable of allowing information about the state of the network to be summarized so that remote applications can use it, stands to a key need. For instance, a new method can lead to the development of particular functionalities (e.g., customized filters) that determine which data should be selected during network events and sent to remote or control applications (e.g., for topology management). These mechanisms should enable data to be delivered to network operators with a minimum degree of complexity to aid in processing and using the information in the infrastructure orchestration systems [24]. An example of the critical nature of such data can be seen in an Intrusion Detection System (IDS) application for mitigating a Distributed Denial of Service (DDoS) attack [27], where the retrieving time of network events is a crucial factor when adopting strategies to prevent any possible further attacks. In addition, the amount of data generated by an event of this nature can impair the performance of pattern matching search engines and affect the storage of data in database systems because of the large volume of information that controllers generate (and usually unnecessary for specific applications).

Although the SDN paradigm provides numerous facilities for the prototyping and implementation of innovative mechanisms for network orchestration [28], the application of the methods outlined above is not a trivial task, largely because of the degree of complexity that characterizes SDN controllers. The optimization or updating of algorithms and the data structure that is an intrinsic part of the internal components of the SDN controller is a costly and complex task: this is particularly the case because there is a need to fully understand (at least at a low level) the SB-API that are offered by each controller, which requires an in-depth knowledge of each implemented mechanism [21].

Our previous work [29] introduces the Network Orchestration Agent (NOA), a middleware running on top of the OpenDaylight SDN controller. NOA acts as a filtering mechanism placed above the OpenDaylight REST API, which is designed to optimizing in real-time the amount of network event data sent by the SDN controller to the remote applications. In summary, NOA does not optimize the SDN controller's internal mechanisms that it is designed for, but rather, carry out data filtering procedures related to network events. Our findings on NOA's restricted employment to orchestration scenarios with a high level of criticality reveal that it is not capable of deciding what kind of optimization is required to deal with the application context for meeting stringent latency requirements.

B. OUR PROPOSAL

Based on the problem statement described hereinabove, together with the limitations of earlier proposals (including our previous work [29]), this research addresses the existing gap of optimizing the volume of network event delivering at the SB-API, previously unexplored for the best of our knowledge. As a solution, we put forward the mechanism designated to as NOTORIETY, which overcomes current SDN controller approaches by assuming the charge of delivering specific knowledge about the underlying networking infrastructure (e.g., link failure, OpenFlow device connection) to atop-running control plane applications, in an optimized manner. The idea behind NOTORIETY stands to anticipate itself the SDN controller at the southbound level to intercept incoming network event-messages and deliver only intent-matching content to atop-running control plane applications. To afford this, NOTORIETY implements a publish-subscribe scheme that standardizes the way control plane applications can express messages carrying data type of their intent. As a result of such optimization, NOTORIETY yields the prospect to deliver significantly less volume of network event data for control plane applications to process in a more agile way. Owing to its modular architecture, NOTORIETY is entirely independent of the SDN controller's procedures design, which takes the system beyond the state-of-the-art.

A proof-of-concept (PoC) of the NOTORIETY proposal is obtained from an OpenDaylight-based, currently one of the most popular and widely used SDN controllers in the industrial and academic premises, testbed implementation. The evaluations are conducted through the virtualization of SDN infrastructures using the Mininet[1] emulator tool, with the aim to validate the NOTORIETY approach and assess its impact on the native OpenDaylight controller workflows against the widely used state-of-the-art SDN controllers, namely ONOS and Floodlight. The proposal's evaluation follows the guidelines provided in the RFC 8456 [30], to benchmark the performance of SDN controllers, which include recommendations ranging from topology configuration to data collection techniques. The outcomes suggest the benefits that NOTORIETY provides, both in terms of reducing the volume of network event data (both generated and delivered) and the total processing time of requests (measured from their receipt by the controller until their delivery to the requesting application).

C. CONTRIBUTIONS

The main research contributions of this study are as follows:

1) A publish-subscribe in scheme that standardizes the way remote control plane applications indicate the messages carrying data type of their intent for NOTORIETY filtering;
2) An enhanced filtering processor, which prevents controller's network-data load overhead even during standby periods (i.e., if no application is requiring data from the system);
3) Optimization of the delivery time of network events to remote applications. This is achieved by standardizing the delivering engines, which forward events (that are

[1] http://mininet.org/

processed by the SB-API) directly to the subscribed applications, thus avoiding the need for the additional processing of other components in the top layers of the controller;
4) A communication interface which is designed to update configuration parameters of filtering mechanisms in real-time, to provide a higher degree of configurability;
5) The validation of the system following the benchmarking methodology guidelines of the IETF RFC 8456, which provide the means of assessing the performance of SDN controllers.

D. PAPER ORGANIZATION

The remainder of this article is structured as follows: Section II examines relevant research studies that are related to the NOTORIETY proposal, and draws attention to our research contribution. Section III introduces the NOTORIETY modular architecture, together with main participating components and supporting functionalities, as well as points out how it differs from our previous work. Section IV outlines in detail the workflows of NOTORIETY's main operational procedures. Section V explains the assessment methodology, setup of the evaluation testbed, and conducts an analysis of the PoC outcomes. Finally, Section VI wraps-up our paper through final considerations and future work suggestions.

II. RELATED WORK

The optimization of the internal SDN controller's architecture is a cutting-edge research area, which seeks to achieve high-performance levels, especially with regard to customization, flexibility, and agility perspectives. In this section, we survey the most significant and relevant research efforts launched by the scientific community, focusing on the enhancements at the NB-API and SB-API architectural levels.

A. NB-API BASED PROPOSALS

Some approaches [18], [31], and [32] make use of the NB-API to carry out load-balancing procedures aiming to alleviate the system when confronted with unforeseen situations (e.g., overload and operational failure). However, these schemes do not provide perspectives of configuration management (e.g., changing the multiple parameters related to the triggers that activate the balancing mechanisms or even switching the multiple load-balancing algorithms implemented by each project). In addition, the solutions have to change the architecture of the controllers to synchronize and enter into the negotiation at the time of the load balancing between the different controllers.

Another group of studies provides individual optimizations for each SDN controller regarding several different aspects such as network monitoring optimization [33], control plane fault tolerance [21], and traffic engineering [25]. However, those proposals lack several management features which can restrict their employment in large scale network systems, such as (*i*) configuration perspectives for the planned mechanisms and other entities (e.g., application and service orchestrators, load balancers, etc.); (*ii*) need for architectural modifications to the upper layers of the controller to provide a new midpoint of communication between the applications and the controller through the system API; and (*iii*) lack of communication interfaces to ensure a better degree of orchestration for other available resource management solutions.

B. SB-API BASED PROPOSALS

The decoupling of SDN controller mechanisms has been employed as a feature of some proposals for optimizing the SB-API performance. In [34], the authors design a data filtering system which is positioned outside the SDN controller, aiming to replace its native functionalities, such as those that ensure processing the data delivered by the SB-API. The authors in [35] introduce an approach for the data processing carried out at the SB-API to reduce the volume of traffic between the controller and orchestration applications and provide an application perspective of data filtering for each involved entity. By using the entity outside of the SDN controller, the solutions significantly increase the processing time, and consequently, the whole delivery time that network events take to be delivered to remote applications running atop the SDN controller. Along with, the solutions require the controller applications (both remote and local) to be re-implemented to receive the events directly filtered by them.

A group of proposals [36], [37] are designed to optimize resource management processes in mobile network infrastructures through latency reduction in the communication between the entities involved (i.e., wireless access points, mobile devices, vehicles, etc.). In all cases, the proposed schemes lack providing compatibility evidence with the other available controllers and, likewise, does not address the implementation features of the solution.

Although the review of the related literature reveals several attempts devoted to optimize SDN controllers, it was found that only a few studies deal with optimizing the internal mechanisms of these SDN controllers, especially with regard to the respective SB-API. References [11] and [21] state that, despite the numerous benefits expected from optimizing the controller's SB-API, this process raises a number of challenges, particularly in terms of implementation complexity and the need for in-depth knowledge of the low-level layers.

C. RELATED WORK WRAP UP

The literature analysis reveal that most of the previous studies use the NB-APIs of the controllers concerned with making improvements for controller load distribution, traffic engineering optimizations, and control plane fault tolerance. Few works stand to mechanisms proposals aimed at optimizing the internal components of the SDN controller (e.g., algorithms and data structure), in particular with regard to the SB-API, due to the complexity involved, as [11] and [21] explain. As Table 1 raises, three parameters are used for making a comparison between most relevant related solutions in the state-of-the-art and the NOTORIETY proposal:

1) Multi controller abstraction: provides enforcement perspectives (e.g., modular architecture, communication with the controller via native API's) regardless of the type of controller(s) that the underlying system supports;
2) Management interfaces: provides support for multiple well-defined interfaces (e.g., communication – socket or REST, configuration, etc.) enabling real-time customization of the operating parameters according with the underlying controller(s) particularities;
3) Real-time system: offers mechanisms that can react to network events, on-demand and in real-time.

The definition of the comparison parameters that Table 1 uses as reference to discuss the solutions set out here, is based on the analysis of the state-of-the-art. Aside of that, it takes into account the main features adopted by each of the related solutions which have functionalities that are essential for the implementation, deployment and compatibility of the solutions with regard to the respective controllers. In the following, we provide a brief discussion about each of the parameters.

The *Multi-controller abstraction* parameter refers to the diversity of SDN controllers currently available and used by the industry and academic community, as reported by [38] and [39]. These studies discuss the importance of the detachment of orchestration architectures from controllers to the SDN ecosystem development. The *Management interfaces* parameter concerns about the automation of network decisions and the management of the resources that are SDN crictical Key Performance Indicators (KPIs) [40], [41]. Since the controller is the central element of network administration and decision making, it is expected that optimization solutions for SDN controllers can provide means for changing configuration parameters so that orchestrating mechanisms can better enforce their functionality. The *Real-time System* refers to the ability to react to network events in real-time, maximizing applications Quality of Experience (QoE). As described in Section III, NOTORIETY's architecture design is based on several interworking modules integration that enables network event handling without service interruption [42].

It is clear from Table 1 that although most of the proposals hold both different and independent controller architectures perspectives, they have been negligent about enabling adopting communication interfaces (e.g., NB-API and SB-API) devoted to enforcing the configuration of the intended mechanisms, as well as the ability to react to network events, on-demand and in real-time. As a result, there is a reduction in both ubiquity and orchestration perspectives that each of them affords. Furthermore, only four out of ten analyzed solutions were designed without imposing changes in the controller architecture to implement the proposed optimizations. In essence, it should be emphasized that this constraint reduces flexibility when seeking improvements (new solutions and mechanisms) in the different SDN controllers and affects the timescale of implementing and evaluating their optimizations.

TABLE 1. Analysis of contributions provided by NOTORIETY concerning featured parameters with respect to plain related proposals.

Proposal	Year	#1 Multi controller	#2 Management interfaces	#3 Real-time system
[37]	2014	-	✓	✓
[35]	2014	✓	✓	✓
[31]	2014	✓	-	-
[33]	2016	-	-	-
[36]	2016	✓	-	-
[21]	2016	✓	-	✓
[32]	2017	-	-	✓
[25]	2018	-	-	-
[43]	2018	-	-	-
[34]	2018	✓	-	-
[18]	2019	-	-	-
NOTORIETY	2020	✓	✓	✓

III. NOTORIETY PROPOSAL

This section examines NOTORIETY, which seeks to provide an optimization mechanism for SDN controllers with enhanced data delivery and filtering capabilities, and thus make improvements for applications that require real-time network event data. The following subsections provide an overview of NOTORIETY's architecture and give a detailed account of the functionalities of each component involved. We also conduct a comparative analysis of the differences between the components and functionalities of our previous work [29] and the current scheme.

A. OVERVIEW

The main objective of NOTORIETY is to maximize the capabilities of the SDN controllers with regard to the delivery of network state information. By standardizing network event messages, it enables them to be delivered to remote applications with a lower degree of data complexity. Figure 2 shows the positioning of NOTORIETY in a reference architecture for SDN controllers, that highlights native mechanisms and communication interfaces.

The objectives outlined above are achieved by implementing mechanisms connected to the SB-API of the SDN controller. Thus, the system's architecture consists of several components that interoperate with the controller's native mechanisms during the network event detection process. It means that NOTORIETY directs the controller's SB-API to forward the network events to the other components of its architecture, which, in their turn, check the corresponding type of event so that the existing filtering rules (previously defined by applications subscribed to the system) can be applied. In this way, remote applications select which types of events they need to receive, and NOTORIETY takes care of the necessary procedures for filtering and standardizing the messages.

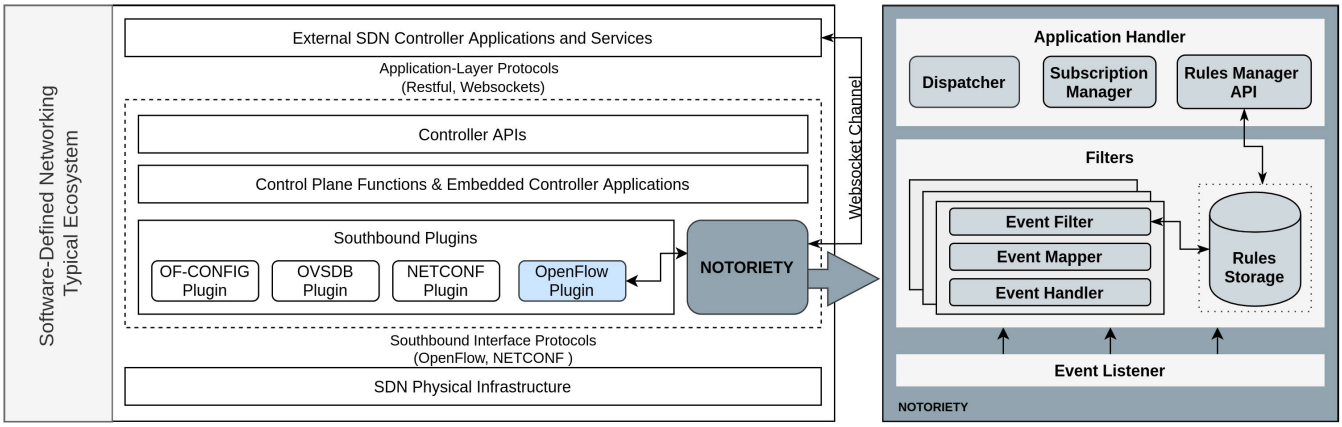


FIGURE 2. Positioning of NOTORIETY on a reference architecture for SDN controllers.

From a high-level standpoint, the components of NOTORIETY's architecture can be described as (*i*) Event Listener; (*ii*) Filters; and (*iii*) Application Handler. Figure 2 displays the modular architecture of the NOTORIETY system and highlights its main components, control functions, and interfaces. The following subsection supplies details of the functionality of each of its components.

B. EVENT LISTENER

The Event Listener is the component responsible for receiving network events processed by the southbound SDN controller interface and forwarding them to the NOTORIETY architecture's filtering components. The Event Listener consists of a set of functions that retrieve the network events processed by the SB-API (by the OpenFlow protocol plugin) of the SDN controllers and detects the type (e.g., OpenFlow device connection and disconnection, PacketIn, etc.) of the retrieved event converting it into a data structure based on hash tables. When the SB-API processes a new network event of the SDN controller, an instance of the Event Listener is created, and the processed event is received as data input.

The Event Listener implementation and the entire NOTORIETY occur at a lower level in the architecture. This is because when a networking event (e.g., the connection of a new device) happens, and the SDN controller retrieves it, it is necessary to operate at the same level as the native mechanisms of the SDN controller (e.g., OpenFlow Protocol Plugin) in order to retrieve the processed event as fast as possible (implemented on the SB-API). This makes it possible to ensure a shorter data delivery time in the communication between the controller and the remote applications since the events can be processed by a minimum number of layers and controller components.

Another critical factor in its operation concerns the SB-API of the SDN controllers: these have functionalities in their implementations that provide the crucial data that is usually requested by applications (e.g., DatapathId of the device, IPv4 address of hosts, MAC address of devices, statistics of links, etc.) for network orchestration. It enables the Event Listener to acquire all this information without the need to request it from higher-level applications and controller's components (an approach that involves adding more processing layers).

C. FILTERS

The Filters component manages the life cycle of the network events during the process of filtering the data received by the Event Listener to abstract the data's complexity. This component associate, based on the data forwarded by the Event Listener, the type of network event to which the message is related. This association is made through a previous mapping of OpenFlow[2] messages for connection and disconnection of devices and communication links. In addition, the Filters store a set of rules (filters customized by the remote applications subscribed to NOTORIETY) that are responsible for delineating how to deliver network events to remote applications.

Figure 2 describes the internal architecture of Filters and displays the main functions responsible for filtering network events, which can be categorized as follows:

- **Rules Storage**: hash table-type data structure, responsible for storing the filtering rules;
- **Event Handler**: function responsible for receiving network events sent by the Event Listener;
- **Event Mapper**: function responsible for mapping the type of network event received by the Event Handler using tags, inserted within the event messages. This tag is based on the identified event type;
- **Event Filter**: function responsible for performing the filtering of the mapped event after checking the rules stored in the Rules Storage and forwarding the filtered event to remote applications through the Application Handler.

[2]http://flowgrammable.org/sdn/openflow/message-layer/

TABLE 2. Specification of filtering rules.

Field	Data	Description
`eventType`	Text	Event type specification
`fields`	List	Filtered fields specification
`contextTag`	Object	Specification of application connection context
`status`	Boolean	Rules status specification
`id`	Text	Unique identifier for a filtering rule

By adopting a standardized approach, which sets templates to meet the format of a filtering rule's requirements, remote applications only need to specify the customized filter, as outlined in Table 2. The `eventType` field specifies the type of event (e.g., connecting and disconnecting devices) that must be processed by NOTORIETY, which is the first method for filtering network events. Therefore, `fields` specify a list of parameters to be filtered by the rule (e.g., IPv4 address, MAC address, DatapathID, flow tables, ports, and links), being the second filtering method, excluding any information present in areas that are not in the fields list. The `contextTag` field informs the connection context of the remote application responsible for inserting the filtering rule. The `status` field determines the operating state of the rule (whether it is active or not). By default, when an application starts the subscription process to NOTORIETY, it does not receive any network events. After its subscription, at least one filtering rule must be defined to set in motion the process of obtaining the network events specified by a filtering rule. Finally, the `id` field is created at runtime for management purposes.

D. APPLICATION HANDLER

The Application Handler serves as an intermediary between network events (previously filtered) and applications outside of the SDN controller, and is responsible for three key tasks: (i) managing the connections of external applications that are subscribed to NOTORIETY; (ii) forwarding network events to subscribed applications; and (iii) managing the filtering rules that are stored in the Rules Storage. It is also responsible for assisting in the structuring of all the NOTORIETY components and implementing a WebSocket server that acts as the main point of communication between the external applications and the other components of the architecture. As shown in Figure 2, the Application Handler is structured as follows:

- **Rules Manager API**: communication interface used by subscribed applications to manage filtering rules (add, modify, and remove). This function assigns a marker tag of identification (based on the information contained in the `eventType` and `fields` areas of the filtering rules template) to the equivalent connection context provided by the Subscription Manager;
- **Subscription Manager**: a set of functions responsible for managing the connection contexts of the applications that are subscribed to NOTORIETY. A connection context represents the link established between the NOTORIETY and a remote application. This information is stored in a hash table data structure called Subscription Storage;
- **Dispatcher**: the function responsible for forwarding the previously filtered events to subscribed applications, based on a comparative analysis between the event identification tag (previously attached by the Event Mapper) and the equivalent marker tag for the connection context of subscribed applications (previously attached during the filtering rule addition process).

E. EVOLVING THROUGH THE PREVIOUS APPROACH

The main architectural differences between the previous work and the new approach are set out to illustrate our system's research contribution:

1) **DataStore Listener**: uses the OpenDaylight REST API to intercept network state data and then processes information at each DataStores change. This component was specially designed for the OpenDaylight, while in the current version, the Event Listener can fit into scenarios composed by any SDN controller.
2) **Event Notifier**: this component is not included in the new approach since there is a need for optimization when communicating with the subscribed applications, now carried out by the Application Handler.
3) **Topology Monitor**: this component has become unnecessary in the new architecture since maintaining the network state for a network event analysis (e.g., check for data duplicity and consistency) can generate overhead in the application (in large scale scenarios). There is no need to store this data in the current approach, where there is a direct connection to the SB-API.
4) **Network State Loader**: like the Topology Monitor, this component has become unnecessary because there is no need to maintain information about the state of the network, due to the new perspectives that have been incorporated by employing the SB-API.
5) **Configure Loader**: this component has become unnecessary because there is a need to reduce complexity in the current system, which uses a WebSocket connection to manage the Filters component as well as improve the communication logistics for the applications in a single component (Application Handler). This approach reduces the complexity by enabling the applications to perform all the interactions (receiving network events, controlling filters, and subscribing to NOTORIETY) through a single instance.
6) **Filters**: the functionality of this component is enhanced with real-time communication capabilities (through WebSocket). As a result, it is no longer dependent on static information provided in the configuration files.

IV. OPERATIONS AND WORKFLOW

This section examines the sequences of activities and operations carried out by the main mechanisms provided by

NOTORIETY. The following subsections describe the activities carried out by NOTORIETY during the basic operational process, which comprises the subscription of remote applications, the management of filtering rules, and the detection of new network events.

A. SUBSCRIPTION MANAGEMENT

NOTORIETY meets the particular demands from external applications by enabling the network operator to subscribe to these applications to receive network events with a lower degree of data complexity. Figure 3 displays the sequence of messages involved during the subscription procedure executed for a new application's request. This process starts when the application makes an HTTP request to the WebSocket server, which is implemented within the Application Handler (Step 1.0).

During the process of establishing a connection, the Application Handler is responsible for generating a connection context that contains information about the communication (e.g., IP address and TCP port) from the external application (Step 1.1). Once this communication has been established, the Subscription Manager inserts this new connection context into the Subscription Storage through an internal function call (Step 1.2). Finally, a successful operation's confirmation response is returned to the subscribed application (Steps 1.3 and 1.4).

B. FILTERING RULES MANAGEMENT

After the subscription process, NOTORIETY enables the application to adopt procedures for managing filtering rules (creation, modification, and deletion). Figure 4 displays the sequence of messages involved during the filtering rule creation procedure.

First of all, the remote application requests creating the filtering rule for the WebSocket server and uses the rule model described in the previous section (Step 2.1) as a parameter. In this step, the Rule Manager API is triggered, and the respective connection context of the requesting application is added to the `contextTag` field of the rule template.

Subsequently, an internal function call is made by the Rules Manager API to create a query rule in the Rules Storage and check whether the filtering rule is already stored (Step 2.2). Assuming the rule does not already exist, the Rules Storage returns the answer with a notification that the requested rule is missing (Step 2.3). Once the Rule Manager API receives the confirmation response, an internal function call is performed to create the filtering rule into the Rules Storage data structure (Step 2.4). The Rules Storage, in turn, stores the rule in the corresponding data structure and returns a confirmation response to the Rules Manager API (Step 2.5).

The Rules Manager API then calls the Subscription Manager, informing the connection context of the requesting application and the values contained in the `eventType` and `fields` of the object rule (Step 2.6). After this, the Subscription Manager generates a tag from the available parameters and then combines it with the application's respective connection context. Once the combination has been made, the Subscription Manager returns the confirmation of the successful operation to the Rules Manager API (Step 2.7). Finally, the Rules Manager API forwards the confirmation that the filtering rule has been successfully created to the remote application (Step 2.8).

C. DETECTION OF A NEW NETWORK EVENT

The life cycle of how a new network event is detected is shown in detail in Figure 5. The Business Process Model and Notation (BPMN)[3] methodology is employed to document the NOTORIETY workflow following the detection of a new network event. This life cycle covers the activities from the moment the network receives an event until the moment the event is delivered, based on the assumption that at least one application subscribed to NOTORIETY aims to receive it.

During the process of detecting a new network event through the SB-API, the SDN controller forwards this event by making a function call to the Event Listener instance. The Event Listener then checks the event details for consistency, which is undertaken by employing a function that verify the information linked to the device related to the event for null data (e.g., DPID = null). If no inconsistencies are detected, the event delivery is effected to the Filters component, which is instantiated by sending the event's raw data as a parameter.

Once the Filters have been instantiated, the Event Handler function is activated, and the data about the previously processed event is received as input. After this, the Event Mapper function is triggered, and, depending on the event type, the identification tag attached to the event message is created. Afterward, the Event Filter function checks with Rules Storage, whether there are rules that have an `eventType` that corresponds to the event tag of the event. If it does, the message resulting from the filtering process is forwarded to the Application Handler component via a function call to the Dispatcher method.

In turn, the Dispatcher function queries the Subscription Manager so that it can receive the connection contexts of the applications subscribed to NOTORIETY. After this, it compares the tag attached in the network event with the equivalent tag linked to connection contexts. Hence, the event message is adapted to ensure it only contains the information specified in the connection `contextTag`. If no field is specified in this tag, the Dispatcher will send the entire event. Finally, the events are sent through a WebSocket channel to all the applications subscribed following the fields specified in the context of the connection tag (`contextTag`).

V. EVALUATION OF THE NOTORIETY PROPOSAL

This section provides a detailed account of the procedures employed to assess the NOTORIETY proposal's performance impact and effectiveness. On the basis that the NOTORIETY proposal is designed to perform real-time network event message handling seamlessly, the benchmarking methodol-

[3]https://www.omg.org/spec/BPMN/2.0/PDF

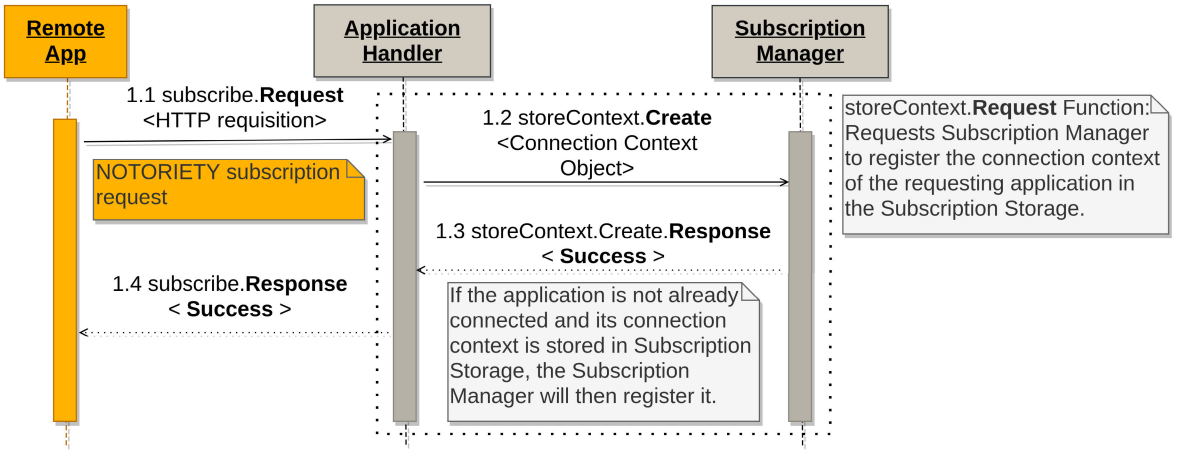


FIGURE 3. Sequence diagram for the application subscription procedure in NOTORIETY.

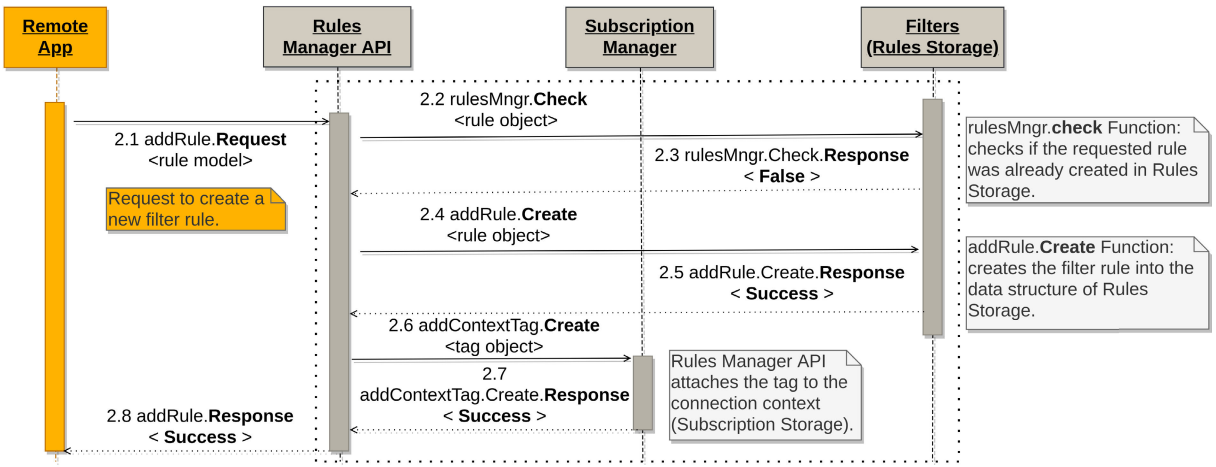


FIGURE 4. Sequence Diagram of the Filtering Rules creation procedure in NOTORIETY.

ogy for SDN controller performance is set out in the RFC 8456 [30] is employed for an accurate assessment. With this in mind, a whole NOTORIETY architecture (detailed in Section III) implementation is put against the mainstream SDN controllers OpenDaylight, Floodlight, and ONOS in their native configuration denoted to as the baseline. In this assessment, the NOTORIETY implementation is put forward as an extension of the OpenDaylight controller.

The following subsection introduces the main concepts of the OpenDaylight controller and investigates its architectural components and limitations. Following this, there is a description of the scenarios and methodology employed for the evaluation and results obtained.

A. THE OPENDAYLIGHT SDN CONTROLLER

The modular architecture of OpenDaylight is defined in terms of the specifications provided by the Object Management Group (OMG) [44], namely, Model Driven Architecture [45]. In this context, OpenDaylight has two main features related to its architecture: (*i*) the internal components of its system become independent from an implementation perspective and are called modules or objects. Its concept can be defined by Model-Driven (MD) architecture; (*ii*) the internal components of the system communicate through predefined APIs that make services available to other internal components. This strategy is called the Service Abstraction Layer (SAL).

Although the OpenDaylight architecture provides a high degree of modularity and scalability for internal controller applications, a top-level domain for each internal system component's APIs is still required. OpenDaylight implements data structures in its system called DataStores to provide a more suitable means of delivering data from components located at the NB-API and even external applications. These structures keep information that has been processed by their internal components (e.g., network status and details about infrastructure devices) and, through REST APIs, provides mechanisms that can enable applications to retrieve this data in a static way or in real-time.

However, the mechanisms provided by the OpenDaylight REST APIs for applications to retrieve data stored

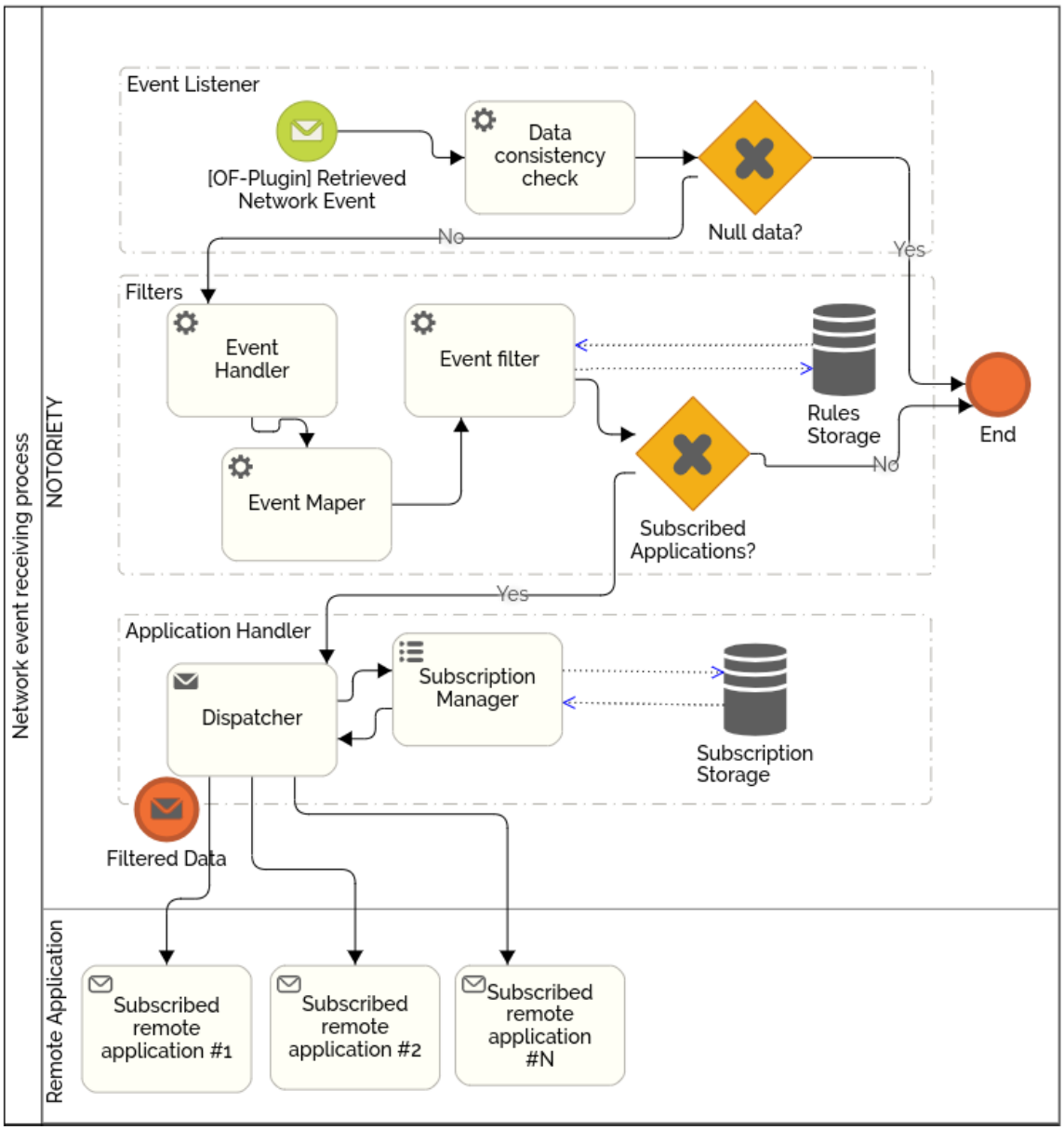


FIGURE 5. Activity diagram for network event receiving.

in DataStores do not offer a way to determine which, and how, these kinds of data should be delivered to applications. Concerning this, a DataStore that is responsible for storing network status contains all the information about the devices connected to OpenDaylight at a given time and details of each device (e.g., flow tables, device ports, DatapathID4, IP address, and MAC). Hence, applications that request data from these structures have to retrieve a large volume of data that needs to be filtered before they can be extracted. In this way, they impose additional processing overhead on the applications, which can impair their performance. Additionally, from the standpoint of real-time information delivery, the approach adopted by the OpenDaylight architecture compels a large number of internal components to process the data received by the infrastructure until that data is stored in the DataStores (owing to the high degree of modularity). After this, the mechanisms provided by REST API, which are responsible for delivering this data to interested applications, can operate. This restriction leads to an increase in the recovery time of events by the applications connected to OpenDaylight and imposes performance limitations on the system as a whole.

B. SCENARIO CONFIGURATION

Figure 6 sketches the testbed evaluation scenario, where the virtualized infrastructure setup corresponds to each experimental set arranged by the several topology configurations described in Table 3. The control plane of each experimen-

tal set is managed by the different SDN controllers configurations described in Table 4. For computational resource optimization purposes, the OpenDaylight with a native implementation and the one implemented with NOTORIETY do not coexist during the experiments. Thus, each controller can be assessed on an individual basis. A remote application was developed to receive the network events generated by each SDN controller during the assessment procedures. For reasons of real-time network event accurate benchmarking, our remote application is responsible for graphically plot the network topology.

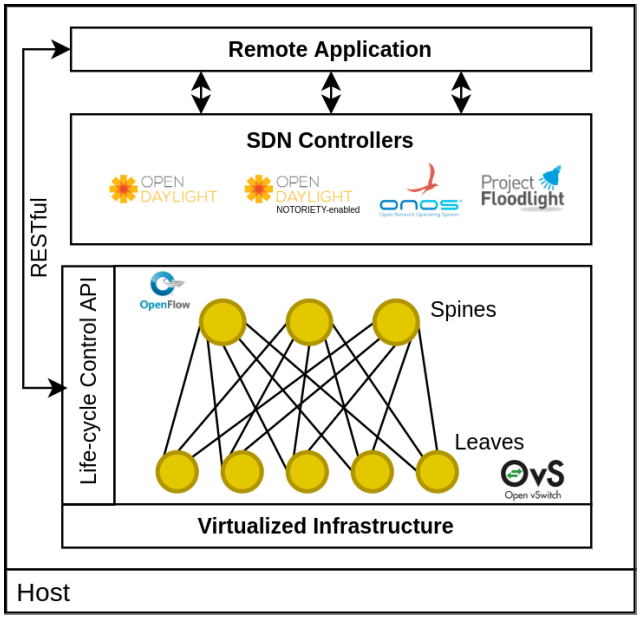


FIGURE 6. Example of the setup employed in the testbed evaluation scenario.

The whole network infrastructure scenario is logically virtualized into a physical host equipped with Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz (16 vCPUs), 32GB of RAM, and Ubuntu Server (18.04) 64-Bit operating system harnessing the features of the Mininet[4] emulator version 2.2.2. For each configuration, connections between network devices are made in a Leaf-spine topology arrangement. By aiming to reproduce a real scenario, each topology configuration is arranged in a pre-determined number of subnets, spines, and leaves, as shown in Table 3.

We also developed an API to control the topologies' operating life cycle during the experiments so that the evaluation could be carried out more effectively. The API enables the remote application to follow the initialization procedure and shutdown of topologies following its needs.

The scenario configuration follows the test considerations for network topology arrangement guidelines, outlined in the item 4.1[5] of the RFC 8456.

[4]http://mininet.org/

[5]https://tools.ietf.org/html/rfc8456#section-4.1

TABLE 3. Configuration of the leaf-spine virtualized topology employed in the evaluation.

Topology configuration	Switches	No of leaves	No of spines	No of subnets
T1	18	12	6	3
T2	36	24	12	6
T3	54	36	18	9
T4	72	48	24	12
T5	90	60	30	15
T6	108	72	36	18

TABLE 4. Setup of the SDN controllers employed in the assessment testbed.

Controller	Release	Version
OpenDaylight (Native implementation)	Sodium-SR2	0.11.2
ONOS (Native implementation)	Sparrow	2.2.0
Floodlight (Native implementation)	-	1.2
OpenDaylight (NOTORIETY-aided)	Sodium-SR2	0.11.2

C. EVALUATION METHODOLOGY

The evaluation experiments mainly aim to (*i*) validate whether the NOTORIETY architecture supports the list of features that Section III details, along with (*ii*) assessing the effectiveness and performance impact that NOTORIETY features take to carry out real-time network SDN controller event message handling.

With this goal in mind, we set a first experiment carrying out the Network Topology Discovery Time (NTDT) procedure following the benchmarking tests for network topology discovery time guidelines provided by the RFC 8456 (item 5.1.1[6]). The NTDT experiment aims to come up with the total time spent in the SDN controller for converging topology bootstrap information and generating the corresponding network events. The NTDT interval is defined by the cumulative time from the first `OFPT-HELLO Exchange` message (sent by a topology-participating switch) that the SDN controller receives until it finishes composing the underlying topology graph state information.

To achieve this, we take into account the whole life cycle that NOTORIETY and the baseline SDN controllers need to perform the necessary operations to process all the network event messages and adequately deliver it to the remote application. With regard NOTORIETY, it includes the application subscription and filtering rules creation (see Figures 3 and 4) procedures, as well as the network event receiving (see Figure 5). For the goal of providing a graphical plotting of the topology, the NOTORIETY subscription scheme includes only the necessary information required by the remote application to perform the topology plotting, namely OpenFlow switches- (*i*) DPID, (*ii*) connected ports, and (*iii*) link status descriptions. On the other hand, the baseline SDN controllers send all available information generated through this process.

In order to obtain experiment outcomes in a 95% confidence level, each of the four implementation trials (i.e., three native SDN controllers and one NOTORIETY-aided version of the OpenDaylight), topology, and network procedure were

[6]https://tools.ietf.org/html/rfc8456#section-5.1.1

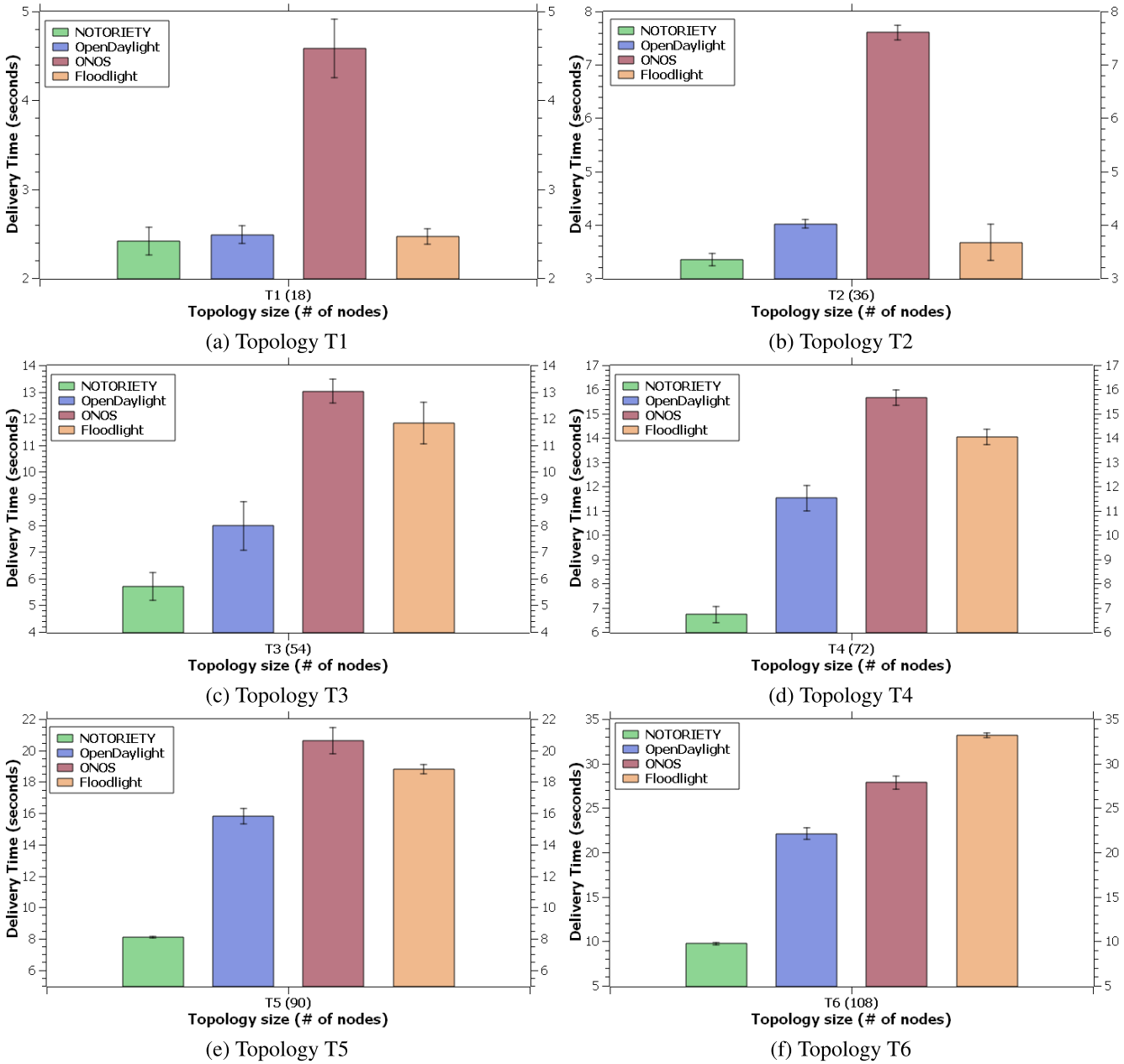


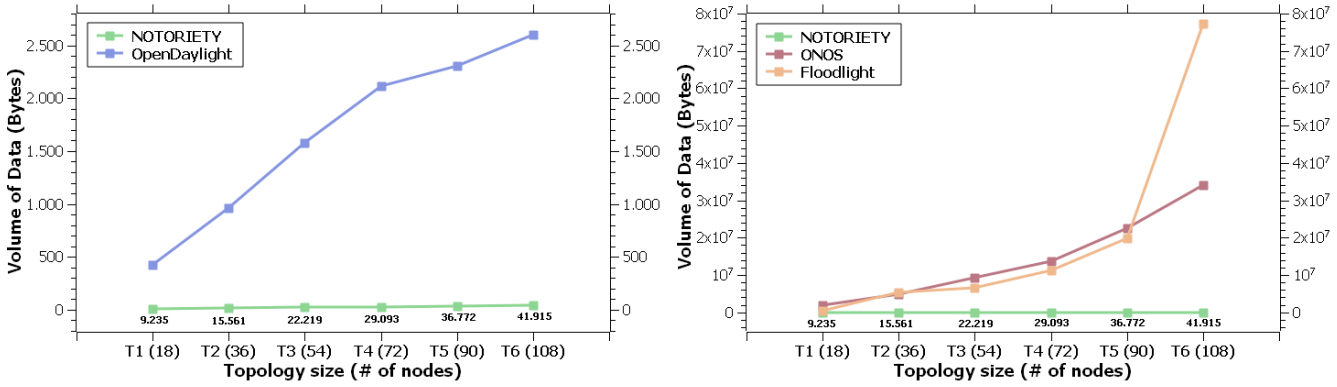
FIGURE 7. Delivery time for NTDT events of each controller compared to NOTORIETY's delivery time.

executed ten times. At the end of each run, the remote application books both network event delivery time spent (in seconds), as well as the total volume of generating data events (in bytes). Afterward, the averaging delivery time of the ten trials' network events is calculated so that revealing the standard deviation. Finally, the total data-collected results obtained in the native versions of the OpenDaylight, ONOS, and Floodlight SDN controllers are compared over the NOTORIETY set of tests.

D. RESULTS

Figure 7 provides evidence of the NOTORIETY architecture featuring network event message handling optimizations by successfully reducing Network Topology Discovery Time (NTDT) message delivery. The results depicted by Figure 7 reveals that NOTORIETY spends a lower network topology delivery time for all topology configurations than the three implementation trials with a great confidence level. It also proves NOTORIETY's system stability by confirming that it outperforms the NTDT achieved by the three baseline native controllers implementations in a scalable fashion. It means that as the topology size rises (i.e., the number of devices increase), the percentage of optimization of NOTORIETY over the baseline implementation also grows.

For instance, as shown in Figure 7(f), NOTORIETY reaches a great optimization level by reducing the NTDT of



(a) NOTORIETY vs OpenDaylight

(b) NOTORIETY vs ONOS vs Floodlight

FIGURE 8. Volume of data from NTDT events, delivered for the remote application, on each controller scenario, compared to NOTORIETY's volume of data.

TABLE 5. Summarization of NOTORIETY'S optimization over OpenDaylight.

Topologies Configuration	Topology Discovery	Volume of data
T1	3.01%	97.84%
T2	16.82%	98.38%
T3	28.47%	98.39%
T4	41.68%	98.40%
T5	48.71%	98.49%
T6	55.95%	98.62%

TABLE 6. Summarization of NOTORIETY'S optimization over ONOS.

Topologies Configuration	Topology Discovery	Volume of data
T1	47.33%	99.54%
T2	56.07%	99.67%
T3	56.19%	99.76%
T4	57.13%	99.79%
T5	60.67%	99.84%
T6	65.07%	99.88%

TABLE 7. Summarization of NOTORIETY'S optimization over floodlight.

Topologies Configuration	Topology Discovery	Volume of data
T1	2.15%	97.16%
T2	8.91%	99.70%
T3	51.76%	99.66%
T4	52.20%	99.74%
T5	56.90%	99.81%
T6	70.62%	99.95%

OpenDaylight, ONOS, and Floodlight by 55.95%, 65.07%, and 70.61%, respectively. Tables 5, 6 and 7 summarize the NTDT optimization percentage of NOTORIETY over all the three baseline controllers.

As can be inferred by the results, the performance of NOTORIETY against the ONOS native implementation is more effective than the other controllers. During the experiments, we noticed that ONOS needed a higher number of modules (total of 179, including those responsible for the OpenFlowPlugin tasks) to deal with the controller system's basics, which creates an extra processing charge.

We now focus on assessing the effectiveness and performance impacts that the NOTORIETY filtering mechanism lays over the volume of data processed during the NTDT experiment, regarding the baseline solutions. Aiming to assess how effective both NOTORIETY and the baseline solutions come up in the experiments with the varying size of topologies, we adopt the methodology of analyzing the volume of data from NTDT events delivered for the remote application on each controller at ending the experiment. Figure 8 presents a comparison of each baseline solution with NOTORIETY's delivered volume of data. Aiming to provide a better understanding of the results, we split the results into two graphs: (*i*) Figure 8(a) compare NOTORIETY's volume of data with OpenDaylight; and (*ii*) Figure 8(b) compare NOTORIETY's volume of data with ONOS and Floodlight.

As confirmed by the results outlined in Figure 8, the implementation aided by NOTORIETY features achieves considerable levels of optimization in the total volume of data delivered to the remote application. Tables 5, 6 and 7 also show the percentage of optimization in the volume of data delivered to the remote application by NOTORIETY against the OpenDaylight, ONOS and Floodlight, respectively. From our studies in these outcomes, we note that even with the increase in the topology size, the capabilities provided by NOTORIETY features fit the remote application's needs through the mechanisms described in Section III. It means that NOTORIETY only forwards the information required to the subscribed remote application, instead of the

baseline solutions, which continues to forward all the produced data. This lack in the baseline native controllers mechanisms implementation result in additional network signaling and application overhead since it is given the responsibility of data filtering to the interested remote entity.

Although the approach introduced by NOTORIETY entails adding extra complexity to the SDN controller ecosystem (by including additional components), the results of the assessment prove that the optimizations allow it to implement the various stages that must be followed for classifying, processing, and delivering the raw data of network events. Moreover, the implementation of filtering mechanisms improves the efficiency of NOTORIETY by reducing the volume of data delivered to external entities and avoids overlapping those already in place at the SB-API.

The NOTORIETY approach's promising benefits trigger a discussion about how new network paradigms such as Edge/Fog computing hosting emerging critical applications (e.g., disaster recovery) can raise service expectations through the innovation introduced by NOTORIETY facilities. In these scenarios, where it is expected that the network infrastructure can keep a high-level QoS guarantee to provide service reliability [46], [47], the applications shall benefit from the real-time delivery messaging capabilities of the NOTORIETY approach.

VI. CONCLUSION

In this study, NOTORIETY is examined as a mechanism that can maximize the orchestration capabilities of SDN controllers by including innovative features for processing and delivering network event information. NOTORIETY's modular architecture provides a wide range of deployment perspectives regardless of controller type. Additionally, it provides interfaces that allow flexible customization of runtime operating parameters. By being provided with optimizations, NOTORIETY can achieve better performance for applications and tools designed for orchestrating SDN infrastructures that need real-time data to perform their functions when faced with network events.

Evaluations were conducted through a Proof-of-Concept (PoC) within the OpenDaylight controller and compared with a native implementation of the mainstream SDN controllers ONOS and Floodlight. The results reveal that NOTORIETY can optimize data delivery of network events in a shorter time and reduce the volume of data to remote applications.

Although all NOTORIETY advantages, some shortcomings need to be addressed, such as the limitation in handling the controller's internal mechanisms to retrieve requests from the applications and make decisions for specific network procedures (e.g., flow installation).

We intend to carry out the validation of NOTORIETY for large scale applications and network procedures in future work. Moreover, we plan to conduct several new assessments in experimental testing infrastructures (such as FIBRE[7]) to determine whether it is feasible to deploy NOTORIETY in production infrastructures. As well as this, alternative techniques (e.g., data structures, parallel computing, etc.) will be investigated to optimize the filtering rules' processing for the application's needs.

[7]https://www.fibre.org.br/

REFERENCES

[1] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, *Software-Defined Networking (SDN): Layers and Architecture Terminology*, Internet Requests for Comments, RFC Editor, document RFC 7426, Jan. 2015. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7426.txt

[2] F. S. D. Silva, A. V. Neto, D. Maciel, J. Castillo-Lema, F. Silva, P. Frosi, and E. Cerqueira, "An innovative software-defined WiNeMO architecture for advanced QoS-guaranteed mobile service transport," *Comput. Netw.*, vol. 107, pp. 270–291, Oct. 2016.

[3] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[4] R. Amin, M. Reisslein, and N. Shah, "Hybrid SDN networks: A survey of existing approaches," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3259–3306, May 2018.

[5] M. Vahlenkamp, F. Schneider, D. Kutscher, and J. Seedorf, "Enabling ICN in IP networks using SDN," in *Proc. 21st IEEE Int. Conf. Netw. Protocols (ICNP)*, Oct. 2013, pp. 1–2, doi: 10.1109/icnp.2013.6733634.

[6] P. Patel, V. Tiwari, and M. K. Abhishek, "SDN and NFV integration in openstack cloud to improve network services and security," in *Proc. Int. Conf. Adv. Commun. Control Comput. Technol. (ICACCCT)*, May 2016, pp. 655–660, doi: 10.1109/icaccct.2016.7831721.

[7] O. Flauzac, C. Gonzalez, A. Hachani, and F. Nolot, "SDN based architecture for IoT and improvement of the security," in *Proc. IEEE 29th Int. Conf. Adv. Inf. Netw. Appl. Workshops*, Mar. 2015, pp. 688–693, doi: 10.1109/waina.2015.110.

[8] A. A. Kadhim, "5G and next generation networks," in *Proc. Al-Mansour Int. Conf. New Trends Comput., Commun., Inf. Technol. (NTCCIT)*, Nov. 2018, p. 99, doi: 10.1109/ntccit.2018.8681173.

[9] L. Liu, "SDN orchestration for dynamic end-to-end control of data center multi-domain optical networking," *China Commun.*, vol. 12, no. 8, pp. 10–21, Aug. 2015.

[10] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2429–2453, 3rd Quart., 2018.

[11] J. H. Cox, J. Chung, S. Donovan, J. Ivey, R. J. Clark, G. Riley, and H. L. Owen, "Advancing software-defined networks: A survey," *IEEE Access*, vol. 5, pp. 25487–25526, 2017.

[12] L. Mamushiane, A. A. Lysko, and S. Dlamini, "SDN-enabled infrastructure sharing in emerging markets: Capex/opex savings overview and quantification," in *Proc. IST-Afr. Week Conf. (IST-Afr.)*, May 2018, pp. 1–10.

[13] A. Triki, A. Gravey, and P. Gravey, "CAPEX and OPEX saving in SDN-compliant sub-wavelength switching solution," in *Proc. Int. Conf. Photon. Switching (PS)*, Sep. 2015, pp. 262–264, doi: 10.1109/ps.2015.7329020.

[14] M. Gharbaoui, C. Contoli, G. Davoli, G. Cuffaro, B. Martini, F. Paganelli, W. Cerroni, P. Cappanera, and P. Castoldi, "Demonstration of latency-aware and self-adaptive service chaining in 5G/SDN/NFV infrastructures," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2018, pp. 1–2, doi: 10.1109/nfv-sdn.2018.8725645.

[15] A. Rego, A. Canovas, J. M. Jimenez, and J. Lloret, "An intelligent system for video surveillance in IoT environments," *IEEE Access*, vol. 6, pp. 31580–31598, 2018, doi: 10.1109/access.2018.2842034.

[16] S. Manzoor, S. M. A. Akber, M. I. Menhas, M. Imran, M. Sajid, H. Talal, and U. Samad, "An SDN enhanced load balancing mechanism for a multi-controller WiFi network," in *Proc. 1st Int. Conf. Power, Energy Smart Grid (ICPESG)*, Apr. 2018, pp. 1–5, doi: 10.1109/icpesg.2018.8384520.

[17] X. Jiang, H. Shokri-Ghadikolaei, G. Fodor, E. Modiano, Z. Pang, M. Zorzi, and C. Fischione, "Low-latency networking: Where latency lurks and how to tame it," *Proc. IEEE*, vol. 107, no. 2, pp. 280–306, Feb. 2019, doi: 10.1109/jproc.2018.2863960.

[18] W. H. F. Aly, "Controller adaptive load balancing for SDN networks," in *Proc. 11th Int. Conf. Ubiquitous Future Netw. (ICUFN)*, Jul. 2019, pp. 514–519, doi: 10.1109/icufn.2019.8805922.

[19] Z. Fan, J. Yao, X. Yang, Z. Wang, and X. Wan, "A multi-controller placement strategy based on delay and reliability optimization in SDN," in *Proc. 28th Wireless Opt. Commun. Conf. (WOCC)*, May 2019, pp. 1–5, doi: 10.1109/wocc.2019.8770551.

[20] A. Sgambelluri, F. Tusa, M. Gharbaoui, E. Maini, L. Toka, J. M. Perez, F. Paolucci, B. Martini, W. Y. Poe, J. Melian Hernandes, A. Muhammed, A. Ramos, O. G. de Dios, B. Sonkoly, P. Monti, I. Vaishnavi, C. J. Bernardos, and R. Szabo, "Orchestration of network services across multiple operators: The 5G exchange prototype," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Jun. 2017, pp. 1–5, doi: 10.1109/eucnc.2017.7980666.

[21] B. Chandrasekaran, B. Tschaen, and T. Benson, "Isolating and tolerating SDN application failures with LegoSDN," in *Proc. Symp. SDN Res.*, Mar. 2016, pp. 1–12.

[22] T. F. Oliveira and L. F. Q. Silveria, "Distributed SDN controllers optimization for energy saving," in *Proc. 4th Int. Conf. Fog Mobile Edge Comput. (FMEC)*, Jun. 2019, pp. 86–89.

[23] Z. Latif, K. Sharif, F. Li, M. M. Karim, S. Biswas, and Y. Wang, "A comprehensive survey of interface protocols for software defined networks," *J. Netw. Comput. Appl.*, vol. 156, Apr. 2020, Art. no. 102563.

[24] C. S. Gomes, F. S. D. Silva, E. P. Neto, K. B. Costa, and J. B. da Silva, "Towards a modular interactive management approach for SDN infrastructure orchestration," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2016, pp. 1–6, doi: 10.1109/nfv-sdn.2016.7919467.

[25] M. Moradi, Y. Zhang, Z. Morley Mao, and R. Manghirmalani, "Dragon: Scalable, flexible, and efficient traffic engineering in software defined ISP networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 12, pp. 2744–2756, Dec. 2018, doi: 10.1109/jsac.2018.2871312.

[26] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2012, pp. 19–24.

[27] F. S. Dantas Silva, E. Silva, E. P. Neto, M. Lemos, A. J. Venancio Neto, and F. Esposito, "A taxonomy of DDoS attack mitigation approaches featured by SDN technologies in IoT scenarios," *Sensors*, vol. 20, no. 11, p. 3078, May 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/11/3078

[28] J. Bhatia, R. Govani, and M. Bhavsar, "Software defined networking: From theory to practice," in *Proc. 5th Int. Conf. Parallel, Distrib. Grid Comput. (PDGC)*, Dec. 2018, pp. 789–794.

[29] K. B. Costa, E. P. Neto, F. S. D. Silva, C. H. F. D. Santos, M. O. O. Lemos, and A. V. Neto, "NOA: A middleware to maximize the OpenDaylight SDN controller orchestration perspectives," in *Proc. IEEE Int. Symp. Local Metrop. Area Netw. (LANMAN)*, Jun. 2018, pp. 67–72.

[30] V. Bhuvaneswaran, A. Basil, M. Tassinari, V. Manral, and S. Banks, *Benchmarking Methodology for Software-Defined Networking (SDN) Controller Performance*, Internet Requests for Comments, RFC Editor, document RFC 8456, Oct. 2018.

[31] Y. Zhou, M. Zhu, L. Xiao, L. Ruan, W. Duan, D. Li, R. Liu, and M. Zhu, "A load balancing strategy of SDN controller based on distributed decision," in *Proc. IEEE 13th Int. Conf. Trust, Secur. Privacy Comput. Commun.*, Sep. 2014, pp. 851–856, doi: 10.1109/trustcom.2014.112.

[32] R. Li, J. Zhang, Y. Tan, and Q. Wu, "An enhanced virtual cluster embedding strategy with virtualized SDN," in *Proc. IEEE 9th Int. Conf. Commun. Softw. Netw. (ICCSN)*, May 2017, pp. 974–981, doi: 10.1109/iccsn.2017.8230256.

[33] D.-E. Henni, Y. Hadjaj-Aoul, and A. Ghomari, "Probe-SDN: A smart monitoring framework for SDN-based networks," in *Proc. Global Inf. Infrastruct. Netw. Symp. (GIIS)*, Oct. 2016, pp. 1–6, doi: 10.1109/giis.2016.7814940.

[34] R. Hark, N. Aerts, D. Hock, N. Richerzhagen, A. Rizk, and R. Steinmetz, "Reducing the monitoring footprint on controllers in software-defined networks," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 4, pp. 1264–1276, Dec. 2018, doi: 10.1109/tnsm.2018.2876654.

[35] T. Choi, S. Song, H. Park, S. Yoon, and S. Yang, "SUMA: Software-defined unified monitoring agent for SDN," in *Proc. IEEE Netw. Oper. Manage. Symp. (NOMS)*, May 2014, pp. 1–5, doi: 10.1109/noms.2014.6838355.

[36] H. Li, M. Dong, and K. Ota, "Control plane optimization in software-defined vehicular ad hoc networks," *IEEE Trans. Veh. Technol.*, vol. 65, no. 10, pp. 7895–7904, Oct. 2016, doi: 10.1109/tvt.2016.2563164.

[37] J. Schulz-Zander, N. Sarrar, and S. Schmid, "Towards a scalable and near-sighted control plane architecture for WiFi SDNs," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2014, pp. 217–218. [Online]. Available: http://eprints.cs.univie.ac.at/5748/

[38] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. L. Smeliansky, "Advanced study of sdn/openflow controllers," Assoc. Comput. Mach., New York, NY, USA, Tech. Rep. Article 1, 2013, pp. 1–6.

[39] L. Zhu, M. Monjurul Karim, K. Sharif, F. Li, X. Du, and M. Guizani, "SDN controllers: Benchmarking & performance evaluation," 2019, *arXiv:1902.04491*. [Online]. Available: http://arxiv.org/abs/1902.04491

[40] M. Seufert, S. Lange, and M. Meixner, "Automated decision making based on Pareto frontiers in the context of service placement in networks," in *Proc. 29th Int. Teletraffic Congr. (ITC)*, Sep. 2017, pp. 143–151, doi: 10.23919/itc.2017.8064350.

[41] S.-C. Chen and R.-H. Hwang, "A scalable integrated SDN and OpenStack management system," in *Proc. IEEE Int. Conf. Comput. Inf. Technol. (CIT)*, Dec. 2016, pp. 532–537, doi: 10.1109/cit.2016.27.

[42] M. Grossmann and C. Ioannidis, "Continuous integration of applications for ONOS," in *Proc. IEEE Conf. Netw. Softwarization (NetSoft)*, Jun. 2019, pp. 213–217, doi: 10.1109/netsoft.2019.8806696.

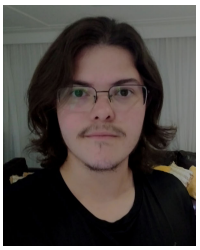
[43] W. Lan, F. Li, X. Liu, and Y. Qiu, "A dynamic load balancing mechanism for distributed controllers in software-defined networking," in *Proc. 10th Int. Conf. Measuring Technol. Mechatronics Autom. (ICMTMA)*, Feb. 2018, pp. 259–262, doi: 10.1109/icmtma.2018.00069.

[44] The Linux Foundation Projects. (2019). *Sodium–OpenDaylight*. [Online]. Available: https://www.opendaylight.org/what-we-do/currentrelease/sodium

[45] OMG. (2007). *Model Driven Architecture*. [Online]. Available: https://www.omg.org/mda

[46] G. Secinti, P. B. Darian, B. Canberk, and K. R. Chowdhury, "SDNs in the sky: Robust end-to-end connectivity for aerial vehicular networks," *IEEE Commun. Mag.*, vol. 56, no. 1, pp. 16–21, Jan. 2018, doi: 10.1109/mcom.2017.1700456.

[47] C. Mouradian, N. T. Jahromi, and R. H. Glitho, "NFV and SDN-based distributed IoT gateway for large-scale disaster management," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 4119–4131, Oct. 2018, doi: 10.1109/jiot.2018.2867255.



KEVIN BARROS COSTA is currently pursuing the master's degree in computer systems with the Federal University of Rio Grande do Norte (UFRN). He is currently a Researcher with the LaTARC Research Laboratory, Federal Institute of Education, Science, and Technology of RN (IFRN), Brazil. His research interests include software-defined networking, network management, and orchestration.

FELIPE S. DANTAS SILVA (Member, IEEE) is currently pursuing the Ph.D. degree in computer science with the Federal University of Rio Grande do Norte (UFRN). He was a Visiting Researcher with the Network Research Group (NRG), Department of Computer Science, Saint Louis University (SLU), USA, in 2020. He is currently a Professor with the Federal Institute of Education, Science, and Technology of Rio Grande do Norte (IFRN), Brazil. He is also the Research Team Lead of the LaTARC Research Laboratory and a member of the Research Group in Future Internet Service and Applications (REGINA). His research interests include SDN, NFV, 5G, mobility management, cloud/fog computing, network/cloud slicing, and QoS/QoE.

LUCAS M. SCHNEIDER is currently pursuing the B.Tech. degree in computer networks with the Federal Institute of Education, Science, and Technology of RN (IFRN), Brazil. He is currently a Researcher with the LaTARC Research Laboratory. His research interests include software-defined networking, network management, and orchestration.

EMÍDIO PAIVA NETO is currently pursuing the master's degree in computer science with the Federal University of Rio Grande do Norte (UFRN). He is currently a Researcher with the LaTARC Research Laboratory and a Substitute Professor of computer networks with the Federal Institute of Education, Science, and Technology of RN (IFRN), Brazil. His research interests include software-defined networking, network management, and orchestration.

AUGUSTO VENANCIO NETO (Member, IEEE) received the Ph.D. degree in computer science from the University of Coimbra, in 2008. He is currently an Associate Professor with the Informatics and Applied Mathematics Department (DIMAp) and a Permanent Member of the Graduate Program of Systems and Computing (PPgSC), Federal University of Rio Grande do Norte (UFRN), Brazil; a member of the Instituto de Telecomunicações (IT), Portugal; and a Researcher on productivity with the National Council of Scientific Researches (CNPq)–Level 2. His research interests include future Internet, 5G/6G, mobile computing, smart spaces, SDN, NFV, and cloud computing fields.

FLAVIO ESPOSITO received the M.Sc. degree in telecommunication engineering from the University of Florence, Italy, and the Ph.D. degree in computer science from Boston University, in 2013. He is currently an Assistant Professor with the Department of Computer Science, Saint Louis University (SLU). His research interests include network management, network virtualization, artificial intelligence, and distributed systems. He was a recipient of several awards, including three NSF projects and a Comcast Innovation Award in 2020.