On Standards for Application Level Interfaces in SDN

Yousef Ibrahim Daradkeh[1]
College of Engineering at Wadi Aldawaser
Prince Sattam bin Abdulaziz University
18611, Kingdom of Saudi Arabia

Dmitry Namiot[3]
Faculty of Computational Mathematics and Cybernetics
Lomonosov Moscow State University
Moscow, Russia

Mujahed ALdhaifallah[2]
College of Engineering at Wadi Aldawaser
Prince Sattam bin Abdulaziz University
18611, Kingdom of Saudi Arabia

Manfred Sneps-Sneppe[4]
Ventspils International Radioastronomy Centre
Ventspils University College
Ventspils, Latvia

***Abstract*—In this paper, authors discuss application level interfaces for Software Defined Networks. While the Application Programming Interfaces for the interaction with the hardware are widely described in Software Defined Networks, the software interfaces for applications received far less attention. However, it is obvious that interfaces to software applications are very important. Actually, application level interfaces should be one of the main elements in Software Defined Networks. It is a core feature. In this article, we want to discuss the issues of standardization of software interfaces for applications in Software Defined Networks area. Nowadays, there are several examples of unified Application Program Interfaces in the telecommunications area. Is it possible to reuse this experience for Software Defined Networks or Software Defined Networks standards are radically different? This is the main question discussed in this paper.**

***Keywords*—SDN; REST API; Northbound interface; application**

I. Introduction

Software-Defined Networking (SDN) is a paradigm that separates network's control logic from the underlying hardware (e.g., routers, switches etc.). SND paradigm promotes the centralization of network control and ability to program the network. It let introduce new abstractions, simplify the network management, and simplify the application programming. Most authors highlight two basic moments for this paradigm:

the abstraction of the network logic from hardware implementation – network logic is a software;

the separation of a control panel and network forwarding

SDN assumes the presence of network controller that coordinates the above-mentioned tasks.

So, SDN concept, by the definition, is based on the various programming interfaces. Actually, SDN controller is a bunch of programming interfaces by itself. In Figure 1, we present the classical model for SDN.

In this paper, we will discuss so-called northbound API. This open entity enables the network application ecosystem. Actually, this ecosystem is the main promise of SDN. It is what SDN networks are for. The idea is to create an intermediate level independent from equipment vendors. In this case, Network Operators can quickly modify or customize their network control through the application API.

Basically, anyone who wants to develop network applications is the potential user for Northbound API. Of course, the question is to propose a common API on this level. Otherwise, developers will face many different proposals from the vendors. There will be no portability, as well as no way to create 'application store' for network programming.

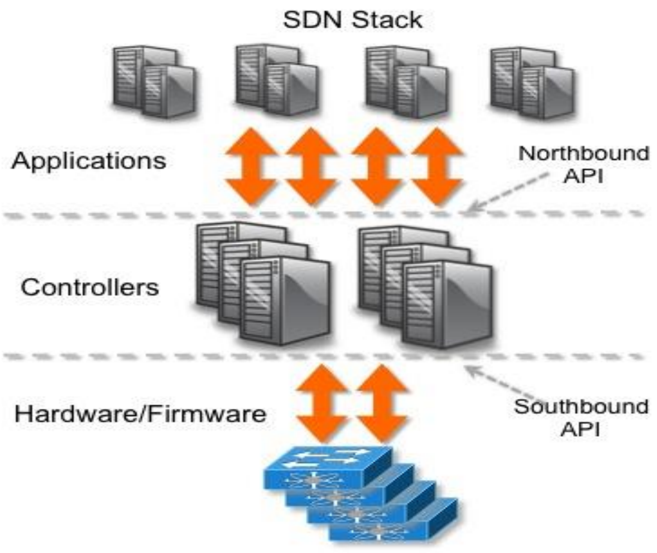

Fig. 1. SDN APIs [1]

Originally, many different sets of northbound APIs are emerging [2]. Currently, more than 20 different SDN controllers are available -- all featuring different northbound APIs. And the Open Networking Foundation (ONF), a consortium dedicated to promoting and commercializing SDN, is studying their variation and why they're all so different [3].

One possible reason is that requirements for a northbound API vary, depending on the needs of the applications and orchestration systems above it. It complicates the collaboration on common API. Actually, one popular opinion is based on ideas to collect market feedback (responses from the developers) first. It makes sense because many programming standards (and Northbound API is about programming only) are based on de-facto approaches, adopted by the majority of developers. For example, as per ONF vision, “the northbound API is a software interface inside a server, and API standards generally emerge from the market, not necessarily from a committee” [4].

The Architecture and Framework Working Group in ONF, originally, set three goals for the Northbound API development:

1) to collect use cases for the Northbound API;

2) to collect a list of examples of the Northbound API and perform some sort of reverse engineering. The goal is to explore what applications do, describe their data model, and what they require from SDN controller;

3) to provide recommendations to industry on required actions.

At this moment, there is no “standard” document that will describe the common requirements to Northbound (application level) API. Typically, a Northbound interface abstracts the low-level instruction sets used by Southbound interfaces to program forwarding devices. Probably, application level interface is the less elaborated area in SDN world [5]. As per this review, most of the existing solutions are either some ad-hoc (proprietary) API or a pure REST API. Authors conclude that it is unlikely that a single Northbound interface emerges as the winner, as the requirements for different network applications are quite different. One possible path of evolution for Northbound APIs are vertically-oriented proposals, before any type of standardization occurs. It is discussed in section 2.

The whole idea of this paper is to discuss the need for application level API for SDN, as well as the possible model for such standard. There are several attempts to create a unified application level program interfaces for telecommunication services. Because this area is very close to SDN, it would be interesting to discuss re-usage of the telecom experience (in the terms of APIs) for SDN. Potentially, it could save a lot of resources (at the first hand, a time for training of developers). It is the main motivation for this paper.

SDN model introduces many new concepts. So, the standardization for SDN is a multi-aspect problem too. In the subsequent sections, the related works in the various aspects of SDN stadardization have been discussed.

The rest of our paper is organized as follows. In Section 2, we present the short history of common APIs. In Section 3, we discuss Network Functions Virtualization. In Section 4, we will talk about the Remote Procedure Calls (RPC) and Representational State Transfer (REST) in SDN. In Section 5, we discuss the possible sources for Northbound API requirements. Section 6 presents the discussion. The key question is: what should be included in basic requirements for application level API and how can we reuse existing unified APIs?

II. On Telecom Standards for Application Level API

Probably, the most notable example of application-level API in the area very close to SDN was Parlay. Parlay (Parlay X) is an attempt to present common application level API in telecom world [6]. The main idea behind the Parlay is to combine service delivery mechanisms for network-centric communications (intelligent network) and service delivery approaches in the enterprise world. By enabling access to network capabilities via an API, any solution provider (independent software vendor) can produce new applications that add value to functionality resident in communications networks. The Parlay API hides the basic network complexity (e.g., signaling capability), but is still able present indirect access to them to enterprise applications and maintain the security level like Network Operators do. This can be achieved by creating an API that resides between the application layer and the service component layer (Figure 2).

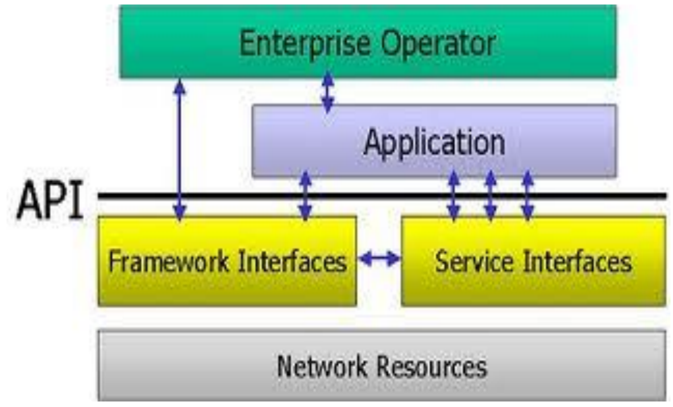

Fig. 2. The Parlay API [7]

The basic principles for API are very transparent:

1) The Parlay API is about programming interface, rather than wire-protocols.

2) The Parlay API should be network independent.

3) End-to-end security.

4) Manageability support. It is the ability to manage the operation and provision of the API.

5) Simplicity. It should be easy to use for software developers.

6) Extensibility. The idea was to expand the API in a series of phases.

At the first hand, it looks very similar to SDN conception. It the main reason we choose Parlay API for the comparison. It is absolutely the same idea – separate a logic and hardware, convert logical part into pure software services. Actually, even the target areas (developers) are similar. In some sense, a Parlay-related movement in programming was even bigger, because there are more developers of telecom services, rather than programmers for network management systems. So, in our opinion, the lessons from Parlay development (end especially, from Parlay failure) could be used for SDN Northbound APIs.

Let us see the components (parts of API) in Parlay [8]. They are vertical oriented:

1) Call Control APIs. It is how to setup and control of connections

2) User Interaction APIs. It is how to send SMS, how to recognize tones, etc. So, they are pure telecom services.

3) Terminal Capabilities API. It is again pure telecom-related services: how to query to terminal capabilities.

4) Connectivity Management API. It is a common API for Quality of Services (QoS).

5) User Status APIs. In practice, it is how to get a status of a mobile terminal.

6) Data Session Control and Account management. It is about billing and tariffs.

The main conclusion is very transparent. It is an attempt to present a standard for applied services in the telecom world. The key word here is "applied". The Parlay API was developed with some model for applied services in mind. The Parlay API assumes (proposes) some model for applied services and supports this model with the standard API. Applied services target the end users, at the first hand. Let us see the typical use cases, presented in [7]: services like 'Buddy List', 'Location-based ads', m-commerce, and 'Scheduler service using Outlook', etc. Each of the particular API could be a plain REST, but it is a vertically oriented solution.

In the case of SDN, the original model for application level API has no problem-oriented divisions. It is a conceptual difference. The question here is very obvious. Shall we talk about different types of SDN applications and originally present Northbound API as a collection of problem-oriented APIs? We see that some like this is mentioned in open-source SDN development [9], but have not seen practical results in direction.

III. On Standards for Network Functions Virtualization

The ONF is working closely with a group of service providers behind Network Functions Virtualization (NFV). The goal is to use Northbound APIs to build top layers for virtual appliances [10].

The NFV was created by a consortium of service providers. It is an attempt to speed up a deployment of new network services. In the basic NFV paper, European Telecommunications Standards Institute described the basic ideas behind NFV [11]. Network Operators' networks are populated with a large and increasing variety of proprietary hardware appliances. It increases the cost of launching new network services. Moreover, hardware-based appliances rapidly reach an end of life, requiring much of the procure-design-integrate-deploy cycle to be repeated with little or no revenue benefit. NFV aims to address these problems by leveraging standard IT virtualization technology to consolidate many network equipment types onto industry standard high volume servers, switches, and storage, which could be located in Data-Centers, Network Nodes and in the end user premises.

NFV is highly complementary to SDN, but not dependent on it (or vice-versa). NFV can be implemented without any SDN being required, although the two concepts and solutions can be combined. The approaches relying on the separation of the control and data forwarding planes as proposed by SDN can enhance performance, simplify compatibility with existing deployments, and facilitate operation and maintenance procedures. In the same time, NFV is able to support SDN by providing the infrastructure upon which the SDN software can be run [12].

So, SDN Application API should be able to play a role of application API in NFV. It means that the area of applications is firmly bounded. It is networking. The typical areas for NFV are:

Virtual Switching. In this case, physical ports are connected to virtual ports on virtual servers. VPN gateways could be virtualized too.

Virtualized Network Appliances. For example, firewalls could be virtualized.

Virtualized Network Services. The typical examples of the virtualized services are network monitoring tools, load balancers, SSL accelerators.

Virtualized Applications. The typical example is virtualized data storage.

In our opinion, this classification could be used for the problem-oriented splitting for developers API (see Section 2). NFV by its nature is "close" to the applied application development and Northbound API can borrow ideas from here.

IV. On REST Model as Standards Base for SDN Apis

According to the fundamental review [5], devoted to SDN, most of the programming interfaces for SDN are based on REST protocol. The conception of programmability is also evolving and is not the pure REST in the case of SDN. Actually, the word 'Northbound' for SDN API could be also outdated [13]. Instead of Northbound and Southbound, ONF uses terms data interface and application interface. It could be strange because application interfaces could be data program interfaces too [14]. But this naming (terminology) is not so interesting comparing with the new model based on RESTCONF [15]. There are several new acronyms that we would like to present here: NETCONF, YANG, RESTCONF, TOSCA.

In general, all of them are about describing data for the calls in REST model.

The Network Configuration Protocol (NETCONF) [16] provides mechanisms to install, manipulate and delete the configuration of network devices. It uses an Extensible Markup Language (XML)-based data encoding for the configuration data as well as the protocol messages.

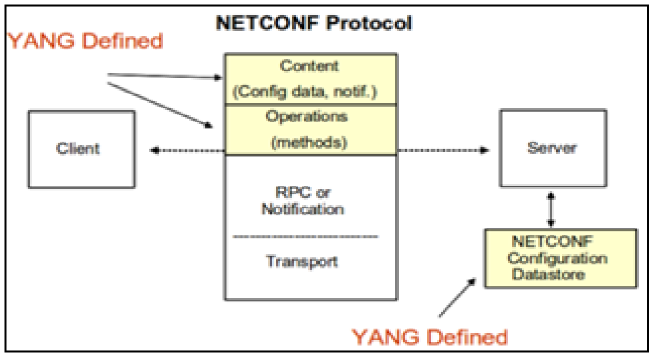

Fig. 3. NETCONF layers [17]

The NETCONF protocol operations are realized as remote procedure calls (RPCs). NETCONF supports devices with multiple configuration datastores. Furthermore, we can also subscribe to notifications or perform other Remote Procedure Calls (RPCs) using NETCONF (Figure 3).

YANG is a data modeling language used to model configuration and state data manipulated by the Network Configuration Protocol (NETCONF), NETCONF remote procedure calls, and NETCONF notifications [18]. As per the specification, YANG is a language used to model data for the NETCONF protocol. A YANG module defines a hierarchy of data that can be used for NETCONF-based operations, including configuration, state data, Remote Procedure Calls, and notifications. This allows a complete description of all data sent between a NETCONF client and server (Figure 4).

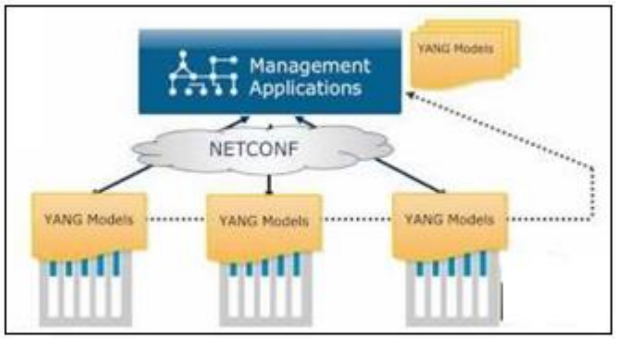

Fig. 4. NETCONF and YANG

YANG models the hierarchical organization of data as a tree in which each node has a name, and either a value or a set of child nodes. YANG provides clear and concise descriptions of the nodes, as well as the interaction between those nodes.

Here is the typical YANG description:

```
list interface {
  key "interface-name";
  leaf interface-name {
    type string;
  }
  leaf speed {
    type string;
  }
  leaf duplex {
    type string;
  }
}
```

And here is NETCONF XML:

```
<interface>
  <interface-name>TenGigabitEthernet 1/0/1</login-name>
  <speed>10Gbps</speed>
  <duplex>full</duplex>
</user>
<interface>
  <interface-name>TenGigabitEthernet 1/0/2</login-name>
  <speed>10Gbps</speed>
  <duplex>full</duplex>
</user>
```

RESTCONF is a model describes a REST-like protocol that provides a programmatic interface over HTTP for accessing data defined in YANG, using the datastores defined in NETCONF [19].

As per RESTCONF specification, the NETCONF protocol defines configuration datastores and a set of Create, Retrieve, Update, Delete (CRUD) operations that can be used to access these datastores. CRUD operation is a standard programming approach for databases. The YANG language defines the syntax and semantics of datastore content, operational data, protocol operations, and notification events. REST-like operations are used to access the hierarchical data within a datastore. So, it is a mapping from NETCONF's CRUD operations to HTTP requests.

What does it mean for SND API? It means a new trend for programmability and APIs for SDN controllers, based on RESTCONF. NETCONF and YANG describe the devices (virtualized devices) and REST (RESTCONF) could be used for access (Figure 5).

OpenDayLight (Open Source SDN controller [20]) proposes a list of Northbound interfaces. Let us see it [21]:

1) Top-Level Inventory: list of all nodes known to the controller.

2) OpenFlow Nodes: extends the top-level inventory node with OF-specific features that allow retrieving and programming of OF-specific states, such as ports, tables, flows, etc.

3) Topology. Base Topology: list of all topologies known to the controller.

4) BGP routing configuration.

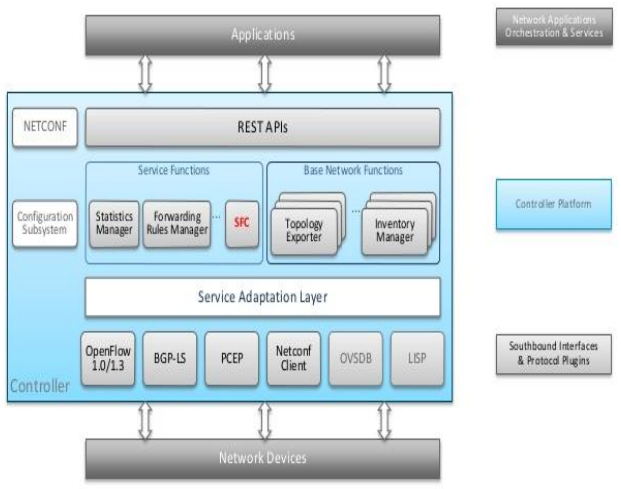

Fig. 5. OpenDayLight model [20]

In other words, it is a very specific networking APIs. In our opinion, the vertical splitting (according to applications area like the above-mentioned list borrowed from telecom world) should be more suitable.

TOSCA is Topology and Orchestration Specification for Cloud Applications [22]. TOSCA should facilitate the creation cloud applications and services. TOSCA provides mechanisms to control workflows, describe relationships and dependencies between resources. TOSCA and YANG can be used together. E,g., in some IaaS (Infrastructure as a Service) configuration the cloud components (compute and storage) could be described by TOSCA. And the connectivity service and networking equipment in the network would be described by YANG.

In other words, all these components bring nothing to Northbound API. All these components are various forms of top-level meta-data and nothing more.

V. On Requirements for SDN API

In 2013, the Open Networking Foundation (ONF) established a working group to focus on the Northbound Interfaces (NBI). One of ideas, proposed by this group was the conception of “scopes” for APIs. It is based on the idea that different applications would require the different granularity (levels of abstraction) from API. ONF’s papers use the term “latitude” [23].

One thing that is missed in the above-mentioned NBI paper is resource sharing. As soon as we separate our architecture on levels we need to some arbitrage for resources too. Otherwise, every application will command all the controller’s resources.

The next idea we’ve discovered from NBI paper is an intent-based interface. Technically, such kind of interfaces should be focused on what the application or service needs, rather than the commands to change the status. Intent-based interfaces are more natural for programmers because they do not need to study a new set of commands. Technically, intent-based interfaces have a natural support in the form of web intents. We’ve used them in M2M projects [24], but it looks like now this direction is closed by Google. So, it looks like the REST will be the prevailing model. But we can make the following important conclusion. Intent-based interface, in general, does not assume the unified model for all devices. So, we should talk, probably, about different NBIs for various SDN controllers.

Technically, building a robust application is not about splitting up their code into smaller services, but instead understanding the connections between these services. So, the connectivity between APIs is more important. And this fact requires a new set of development tools. As an example, we could mention PANE SDN controller [25]. The controller provides an API that allows applications to dynamically add autonomously request network resources. PANE includes a compiler and verification engine to ensure that bandwidth requests do not exceed the limits set by the administrator and to avoid starvation, i.e., other applications shall not be impaired by new resource requests [5].

Top-level classification for NBI could be borrowed from telecom applications. For example, we should follow to [26, 27] and define the following classes:

Model API

Interfaces and objects comprising the domain model. For example the devices, ports, network topology, and related information about the discovered network environment.

Control API

Interfaces to access the modeled entities, control their life-cycles and in general to provide the basis for the product features to interact with each other.

Communications API

Interfaces which define the outbound forms of interactions to control, monitor, and discover the network environment.

Health Service API

Allows an application to report its health to the controller and listen to health events from the controller and other applications.

VI. The Discussion

Comparing SDN user cases and Parlay. Let us return to the original ONF’s plans and describe the potential use cases for Northbound (Application) SDN API. In general, we present the following uses cases for SDN [28]:

1) *Cloud Orchestration*

Traditionally, networks and servers were managed separately and independently. SDN is a proper way to integrate management of both network and cloud frameworks. It is, actually, what SDN are for. And for 3-rd party software applications, the API behind SDN is the natural way to get access to the abstracted hardware [29].

2) *Load Balancing*

Online services, e.g., search engines and web portals, are often replicated on multiple hosts in a data center for efficiency. The load balancer dispatches client requests to a selected service replica based on certain metrics such as server load. With SDN, the load balancing can be integrated within any element in the network.

3) Monitoring and Measurement

It is a yet another classical task for SDN. We can perform network monitoring operations and measurements without any additional equipment. Also, the monitoring can be integrated with any network element.

4) Routing

The idea is very similar to load balancing. The routing services can be virtualized and implemented via programming modules [30].

5) Network Management and QoS

With SDN, it is very easy to build a centralized solution for traffic analysis, for example. SDN software can analyze traffic patterns as well as a quality of services.

As we see, all the above-mentioned tasks are specific network applications. There is almost nothing common (probably, except management and QoS) with applications for telecom. Parlay (in telecom world) offers (indirectly) some model for the possible services. The key idea for SDN could be shortly described as integration. SDN is about the integration of networking into 3-rd party applications.

This conclusion has a direct implication to the possible solutions for Northbound API. Without Network Functions Virtualization, something conceptually similar to Parlay cannot be expected. Actually, just a list of the various technical APIs with the simple form for 3-rd party integration could be provided here.

REST and meta-data. In this sense, the idea of RESTCONF looks more promising. REST API are easy to use, they are simple and very understandable for the developers. As per [5], most of the existing SDN program interfaces are REST-based. But with REST APIs (in general) programmers will face another issue. In practice, REST is missing meta-data [31]. Let us see one example from Neutron API (OpenStack) [32]. The request is a typical REST:

```
GET /v2.0/networks?limit=2
Accept: application/json
And here is a response (part of it):
{ "networks":[ {
"status":"ACTIVE",
"subnets":[
"a318fcb4-9ff0-4485-b78c-9e6738c21b26"
],
"name":"private",
"admin_state_up":true,
"tenant_id":"625887121e364204873d362b553ab171",
"id":"9d83c053-b0a4-4682-ae80-c00df269ce0a",
"shared":false
} ] }
```

REST approach proposes the uniform interface. It means that all resources present the same interface to clients. And it is one of the reasons for REST popularity.

SOA and meta-data. Alternatively, the Service Oriented Architecture (SOA) approach (where the REST is from) may offer personalized interfaces for the different resources [33]. The whole SOA model often compared with REST is based on the idea that different services have different interfaces. It means, immediately, that we need to provide the definition for used interfaces. Indeed, the definition of the services is a key part of SOA. For example, Web Service Definition Language (WSDL) [34] is a part of SOA specification. A WSDL definition of a Web Service defines operations in terms of their underlying input and output messages. Unlike this, REST is based on the self-described messages. WSDL defines the form of the data that accompany the messages in SOA. REST does not provide this information. In other words, SOA has got a rich set of metadata.

On meta-data for REST. The problem with meta-data support is very transparent. Let us see the above-mentioned Neutron API example. How the developers can get information (“get” means programmatically discovery) about the following elements:

HTTP command (it is GET in this case)

URI (it is /v2.0/networks)

Output formats (it is JSON)

Version (it is 2.0 in this case)

Optional and mandatory parameters (it is limit=2)?

There are no ways to discover this information programmatically in the modern implementations of REST approach. The key work here is “programmatically”. The typical model for REST-based API deployment includes the “manual” consulting (reading) with the API manual. So, without the metadata, there is no way to automate programming [35]. That is why we can conclude that metadata for Northbound API (application API) is a key problem.

Another classical example is network management. With SNMP [36], an application can programmatically detect new devices and their features. It is a classical example of meta-data deployment.

VII. The Conclusion

As the conclusion of this review, the following suggestions have been made. Unified API for NBI is unlikely. This conclusion is based on the fact that SDN NBI API is not designed to solve applied-level problems. In SDN model, NFV works on an application level. In the same time, we can reuse some developments from telecommunication APIs for SDF. Firstly, in our opinion, the classification for NBI API could be and should be unified. This classification could be directly borrowed from the telecommunications API.

But inside of top-level classes (and this is important in our proposal), the effort should not be focused on developing a common API. The efforts should be concentrated on the search for a unified approach to the description of the APIs. In other words, in our opinion, the developers need unified metadata description, rather than unified API. In practice, this conclusion proposes a fix for meta-data in REST APIs.

The main use cases for NBI applications are the integration and analysis of traffic. These applications should be automated.

It is what NBI APIs are for. But the basis for automating the programming is exactly the meta-data. And the harmonization of metadata is, in our opinion, the main task.

Of course, the lack of meta-data in REST model is not a specific SDN problem. As we stated above, it is a payment for REST model simplicity. But the network programming (network management, for example) really requires automated solutions. And REST programming cannot be automated without some form of meta-data. In this connection, we can conclude that the real problem for application level API in SDN is the way for describing meta-data. In this connection, RESTCONF approach looks promising. It has meta-data for network elements. The question is how to expand this information to REST API. In our opinion, it is what Northbound (or Application level) API standardization should be about.

ACKNOWLEDGMENT

The project was supported by the deanship of scientific research at Prince Sattam bin Abdulaziz University (Kingdom of Saudi Arabia) under the research project # 2014/1/863.

REFERENCES

[1] The Northbound API- A Big Little Problem http://networkstatic.net/the-northbound-api-2/ Retrieved: Jul, 2016.

[2] Sezer, Sakir, et al. "Are we ready for SDN? Implementation challenges for software-defined networks." Communications Magazine, IEEE 51.7 (2013): 36-43.

[3] Ortiz, Sixto. "Software-defined networking: On the verge of a breakthrough?" IEEE Computer 46.7 (2013): 10-12.

[4] Do SDN northbound APIs need standards? http://searchnetworking.techtarget.com/feature/Do-SDN-northbound-APIs-need-standards Retrieved: Jul, 2016

[5] Kreutz, Diego, et al. "Software-defined networking: A comprehensive survey." Proceedings of the IEEE 103.1 (2015): 14-76.

[6] Yates, M. J., and I. Boyd. "The Parlay network API specification." BT Technology Journal 25.3-4 (2007): 205-211.

[7] Uve Herzog "OSA & Parlay. Enabling an open services market" http://archive.eurescom.eu/message/messageJun2002/tutorial.asp Retrieved: Jul, 2016.

[8] Sneps-Sneppe, Manfred, and Dmitry Namiot. "About M2M standards and their possible extensions." Future Internet Communications (BCFIC), 2012 2nd Baltic Congress on. IEEE, 2012.

[9] C. E. Rothenberg, R. Chua, J. Bailey, M. Winter, C. Correa, S. Lucena, and M. Salvador, "When open source meets network control planes," IEEE Computer Special Issue on Software-Defined Networking, November 2014

[10] Schneider, Fabian, et al. "Standardizations of SDN and ITs practical implementation." NEC Technical Journal 8.2 (2014): 16-20.

[11] NFV White Paper http://portal.etsi.org/nfv/nfv_white_paper.pdf Retrieved: Sep, 2016

[12] Nunes, Bruno, et al. "A survey of software-defined networking: Past, present, and future of programmable networks." Communications Surveys & Tutorials, IEEE 16.3 (2014): 1617-1634.

[13] Medved, Jan, et al. "Opendaylight: Towards a model-driven sdn controller architecture." Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014. 2014.

[14] Namiot, Dmitry, and Manfred Sneps-Sneppe. "On software standards for smart cities: API or DPI." ITU Kaleidoscope Academic Conference: Living in a converged world-Impossible without standards?, Proceedings of the 2014. IEEE, 2014.

[15] Bierman, A., Bjorklund, M., Watsen, K., & Fernando, R. (2014). RESTCONF protocol. IETF draft, work in progress.

[16] Enns, R., Bjorklund, M., Schoenwaelder, J., & Bierman, A. (2011). Network configuration protocol (NETCONF) (No. RFC 6241).

[17] NETCONF layers http://itential.com/tech-briefs/ Retrieved: Jul, 2016

[18] YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF) https://tools.ietf.org/html/rfc6020

[19] RESTCONF Protocol http://tools.ietf.org/html/draft-bierman-netconf-restconf-04

[20] Baucke, Stephen, et al. "OpenDaylight: An Open Source SDN for your OpenStack Cloud." An Open-Stack Summit, Hong Kong (2013).

[21] OpenDayLight Northbound API https://wiki.opendaylight.org/view/OpenDaylight_Controller:RESTCONF_Northbound_APIs Retrieved: Jul, 2016

[22] Mijumbi, Rashid, et al. "Management and orchestration challenges in network functions virtualization." IEEE Communications Magazine 54.1 (2016): 98-105.

[23] Northbound Interfaces Working Group https://www.opennetworking.org/images/stories/downloads/working-groups/charter-nbi.pdf Retrieved: Oct, 2016

[24] Sneps-Sneppe, Manfred, and Dmitry Namiot. "About M2M standards and their possible extensions." Future Internet Communications (BCFIC), 2012 2nd Baltic Congress on. IEEE, 2012.

[25] Guha, Arjun, Mark Reitblatt, and Nate Foster. "Machine-verified network controllers." ACM SIGPLAN Notices. Vol. 48. No. 6. ACM, 2013.

[26] Marie-Paule Odini, Hewlett-Packard, "SDN and NVF for Carriers", ETSI Future Networks Workshop, April 10, 2013

[27] Chung-Shih Tang, Chin-Ywu Twu, Jen-Hong Ju, Ying-Dian Tsou "Collaboration of IMS and SDN to enable new ICT service creation". Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific 17-19 Sept. 2014 pp.:1-4.

[28] Jarschel, Michael, et al. "Interfaces, attributes, and use cases: A compass for SDN." IEEE Communications Magazine 52.6 (2014): 210-217.

[29] M.Banikazemi, D.Olshefski, A.Shaikh, J.Tracey, G.Wang, "Meridian: an SDN platform for cloud network services," IEEE Communications Magazine, vol. 51(2), 2013, pp. 120-127

[30] C.E. Rothenberg, M.R. Nascimento, et.al. "Revisiting routing control platforms with the eyes and muscles of software-defined networking," Proceedings of the first workshop on Hot topics in software defined networks, Aug. 2012, pp. 13-18.

[31] Sneps-Sneppe, Manfred, and Dmitry Namiot. "Metadata in SDN API for WSN." 2015 7th International Conference on New Technologies, Mobility and Security (NTMS). IEEE, 2015.

[32] Denton, James. Learning OpenStack Networking (Neutron). Packt Publishing Ltd, 2014

[33] Al-Rawahi, N., and Y. Baghdadi. "Approaches to identify and develop Web services as instance of SOA architecture." Proceedings of ICSSSM'05. 2005 International Conference on Services Systems and Services Management, 2005.. Vol. 1. IEEE, 2005.

[34] Newcomer, Eric, and Greg Lomow. Understanding SOA with Web services. Addison-Wesley, 2005.

[35] Namiot, Dmitry, and Manfred Sneps-Sneppe. "On IoT Programming." International Journal of Open Information Technologies 2.10 (2014): 25-28.

[36] Harrington, David, Bert Wijnen, and Randy Presuhn. "An architecture for describing simple network management protocol (SNMP) management frameworks." (2002). https://tools.ietf.org/html/rfc3411 Retrieved: Aug, 2016.