Improving Northbound Interface Communication in SDWSN

Sean W. Pritchard*, Reza Malekian† and Gerhard P. Hancke‡
Department of Electrical, Electronic
and Computer Engineering
University of Pretoria
Pretoria, South Africa
Email: *spritchard001@gmail.com, †reza.malekian@up.ac.za, ‡gerhard.hancke@up.ac.za

Adnan M. Abu-Mahfouz
Meraka Institute
Council for Scientific and Industrial
Research (CSIR)
Pretoria, South Africa
Email: a.abumahfouz@ieee.org

***Abstract*—Software-Defined Wireless Sensor Networking (SDWSN) is an emerging paradigm that seeks to alleviate the inherent resource constraint issues present in Wireless Sensor Networks (WSN) by adopting a Software-Defined Networking (SDN) approach to the management of WSN. This SDWSN paradigm is said to play a crucial role in both the developing Internet of Things (IoT) paradigm and the development of smart city grids. The northbound and southbound SDWSN interfaces are important for realizing efficient network understanding and programmability, however there has been a lack of attention towards the northbound interface as most work done has been surrounding the southbound interface. Therefore some work is needed to improve the northbound interface so that it may allow for a better degree of network programmability. In order to achieve network programmability and automation, there is a need for a metadata based Application Programing Interface (API). The work done in this paper seeks to improve the northbound interface communications by addressing the issue of a metadata in REST as well as identifying potential platforms for the development of a metadata framework.**

I. Introduction

Software-Defined Networking (SDN) is a networking paradigm that seeks to simplify network management and configuration by separating control plane which encompasses the network intelligence, from the packet forwarding engine of the network [1]. This results in centralized control of the network which allows for dynamic control and network programmability.

Wireless Sensor Networks (WSN) consist of multiple smart sensor nodes capable of data acquisition and wireless communications [2]. The advancement in Micro-Electrical-Mechanical Systems (MEMS) as well as the development of the Internet of Things (IoT) paradigm have resulted in research surrounding WSNs as platform for both the IoT paradigm and smart city grid systems (for example smart water management systems [3] - [5]).

However there are many inherent issues present in WSNs which are intensified when trying to expand these networks. Poor network management and the inability to realize heterogeneous-node networks, increase the severity of the resource constraints present in WSNs such as limited processing and communication bandwidth. Therefore current WSNs are not able to meet the challenges presented by the IoT and smart city grids.

This brings about a new paradigm called Software-Defined Wireless Sensor Networks (SDWSN) which seeks to solve the inherent issues present in WSNs by using a SDN approach to WSNs [6], [7]. By combining the two paradigms, most of the inherent issues of WSNs are alleviated resulting in a new paradigm which is expected to play a large role in both the IoT and smart city grid systems.

The SDN paradigm introduces three layers within network architecture, the application layer which handles network applications such as management applications, the control layer which encompasses all control functionalities within the network and the data layer which contains all data forwarding functionality. However as SDWSN is a combination of two developing paradigms, a lot of development is still needed. The northbound interface is the interface between the application layer and control layer and essentially provides the global view of the entire network and its functions. This interface is important as it provides for efficient network understanding and resource optimization [8] and has received little attention when considering both SDN and SDWSN as most of the focus has been surrounding the southbound interface between the control plane and data plane [9].

This lack of focus within the northbound interface arises from the fact that the SDWSN northbound interface is said to be application specific due to the application specific nature of WSNs, and it is built on the SDN controllers, for which there are multiple different types resulting in no commonality within the Northbound interface and a lack of standards [10]. Therefore there is a need for a common northbound API to increase interoperability.

In order to improve northbound communications and provide a more efficient network understanding, the SDWSN specific design requirements must first be discussed in order to identify specific areas for improvement.

There are few papers that address the northbound interface [9], [10] and they are mainly focused on SDN. Within SDWSN few papers address this issue [6], however, they do not investigate possible solutions to this issue, thus asserting the need for an investigation into possible solutions.

II. Northbound Design Requirements

The SDWSN architecture is split into three different layers as shown in Fig. 1 [6]. Here it can be seen just how important the northbound and southbound interfaces are for vertical cross layer communication. Although both interfaces are vital to the development of the SDWSN paradigm, most of the attention has been placed on southbound communications.

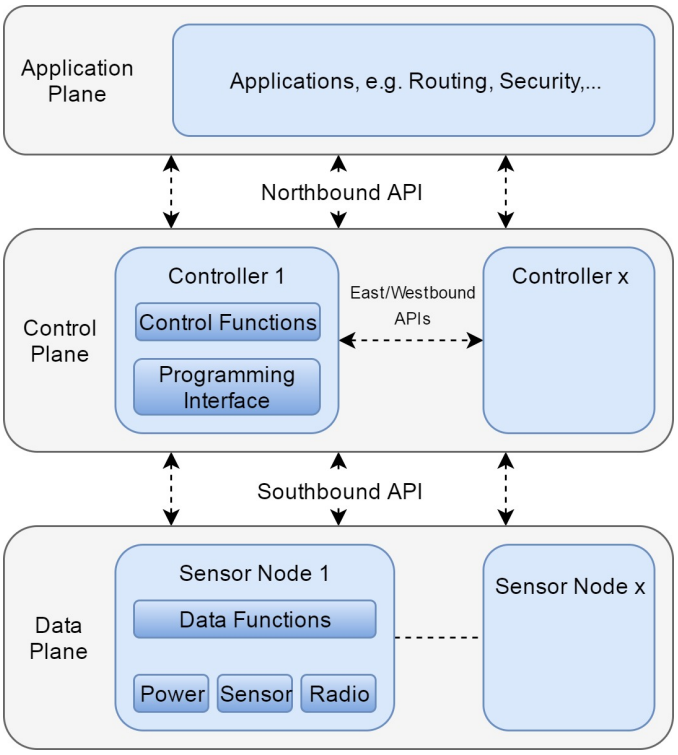

Fig. 1. SDWSN architecture [6]

Due to the multitude of SDWSN controller implementations, and given the fact that the northbound interface is built on network SDN controllers and the fact that it is somewhat application specific, there are no standards for northbound interface communications. However, there is a need for a standard framework that allows for interoperability.

Kobo *et al.* [6] propose the following design requirements with regards to SDWSN northbound communications.

- Rigidity with built in security and conflict prevention.
- The interface can be used in conjunction with a visualization layer to address high heterogeneity.
- In order to achieve better network programmability, a metadata based interface is required.

In terms of rigidity and security, applying SDN application layer security to the SDWSN application layer potentially solves most of the security concerns within the northbound interface [11]. These solutions include validation and verification models such as PermOF [12], VeriFlow [13] and Flover [14]. For rigidity, cryptographic solutions outlined by WSN security measures can be applied to the northbound interface.

Currently, the OpenRoads [15] (also known as wireless OpenFlow) API is the most common northbound API. It is a management interface that allows for the management of wireless networks using network visualization. This visualization layer approach can be used to address the design requirements for high heterogeneity as mentioned above.

The remaining design requirement is the most important in terms of network programmability, to implement a metadata based northbound API. This metadata based API is said to be REST [16] based as REST API's are easy for developers to use. Most of the existing SDN interfaces are REST based, however the problem with REST is that it does not offer metadata [6]. Metadata is crucial for obtaining information from applications, therefore without metadata it becomes challenging to automate programming.

An alternative to REST API's would be Service Orientated Architecture (SOA) which does offer metadata through services such as Web Service Descriptive Language (WSDL) [17]. However these services define different interfaces for each service application, thus removing commonality and heterogeneity.

Thus in order to improve the northbound interface communication, the metadata challenge within REST must be addressed. Various models and languages have been developed in order to incorporate metadata into REST API's in order to develop a northbound API. These are discussed in the section below.

III. REST-based API Interface Models

Since the adoption of REST-ful APIs by the development community, multiple languages and frameworks have been made to document the behavior of REST-ful APIs. Although most of the developed frameworks have focused on documentation rather than metadata-based description due to the fact that many believe that REST-ful APIs must be self describing and therefore do not need metadata support [18]. Despite this, some models provide promising solutions when considering metadata support within REST APIs.

A. WSDL 2.0

WSDL is a language (typically in XML format) that is used to describe web services which has previously not supported REST web service descriptions due to the inadequate HTTP binding used to describe HTTP and XML communications. However, the second version of WSDL, WSDL 2.0, was created to address this issue and as a result is now able to describe REST web services [19].

WSDL can be used to describe all the details of a web service such as the URL, communication mechanisms, interface and message types. A WSDL 2.0 document contains a root *description* element containing four sub-elements, namely *types, interface, binding* and *service*.

1) Types: Describes the web service's messages by listing all XML schema elements and type definitions for the messages. WSDL 2.0 is typically used with XML schema, however it can also support other type systems such as JSON.

2) Interface: Defines all operations and includes all input, output and fault messages passed, as well as the order in which they were passed.

3) Binding: Defines client-service communications. For REST services, the binding would specify that the clients can communicate using HTTP.

4) Service: Connects the web service's address to a specific *interface* and *binding*.

In the past, WSDL 2.0 has not been adopted due to the fact that most believe that REST messages must be self-describing and therefore there is no need for a description language. There is also the issue of security in that WSDL files result in vulnerabilities due the XML-based specification.

B. WADL

The direct REST-based alternative to the WSDL 2.0 specification is Web Application Description Language (WADL), which provides a machine-readable description of HTTP web applications in an XML format [20]. It is similar to WSDL in that is is language independent and describes a list of resources, the relationships between the resources, the HTTP method that can be applied to the resources and finally the resource representation formats.

Although WADL was submitted to the World Wide Web Consortium (W3C), it has not been considered for standardization, and has not been widely adopted in industry. However, it has support from organizations such as Apache and Oracle and some favor WADL as an attempt to add metadata support to REST [9].

C. RESTCONF

RESTCONF is a model that describes the mapping of YANG data to a REST-ful API [21]. It is a REST-based protocol that runs over HTTP and is used to access YANG defined data, using Network Configuration Protocol (NETCONF) defined datastores.

NETCONF is a configuration protocol used to configure network devices. It uses basic operations to edit and query the network configuration on a device which are realized using CML encoded Remote Procedure Calls (RPCs).

The YANG data modeling language is used to define the data sent over NETCONF. It can model both the configuration data as well as the manipulated state data. A YANG model provides a hierarchical description of all NETCONF-based operational data sent between a client and server.

When considering REST-ful APIs, RESTCONF provides an interface to map the NETCONF operations to REST-like operations in order to access the hierarchical YANG data [10]. This allows for a programmable network API, especially when considering the SDN paradigm.

D. REST API Documentation Frameworks

In terms of generating documentation for REST-ful API's, different frameworks have been developed in order automatically and semi-automatically generate the required documentation. These different documentation frameworks are discussed below.

1) Swagger: Swagger is becoming one of the most popular API framework development tools available and enables the development of REST-ful API's, from design and documentation to testing and implementation [22]. The framework itself is built on the OpenAPI Specification [23] (previously known as the Swagger Specification) which aims to standardize REST APIs and allow for human- and machine-readable documentation so that they can understand services without access to source code.

Using a tool called Swagger UI, documentation for the entire API can be automatically generated in both human and machine understandable formats (through an HTML interface with JSON or XML formats). Swagger can be used to describe high-level API behavior and also allows for API design from the bottom-up through the use of code generation tools that make use of Swagger documentation [24]. Also due to the fact that Swagger is not language specific, code can be generated for various coding languages, making Swagger a popular choice for APIs that run on multiple platforms.

2) RAML: REST-ful API Modeling Language (RAML) [25] is another framework that has been developed to describe and document REST APIs. However, RAML can also be used to describe APIs that are not truely REST-ful as well such as APIs that make use of SOAP or RPC. Similar to Swagger, it can also be used to design, document and test REST-ful APIs and has a large industry backing.

At its core, however, RAML is a language used to describe and document REST services, making use of YAML file formats, although it also supports JSON formatting. The hierarchical nature of RAML results in the visualization of the requests and responses of the API, and therefore it can be used to not only document the API, but also plan the API with great detail.

The major disadvantage of RAML is due to its lack of openness resulting in the need for an adapter when developing a northbound API [9].

3) API Blueprint: The final popular documentation framework for describing REST-ful APIs is API Blueprint [26]. It is a documentation-orientated language making use of Markdown as its format. Unlike Swagger and RAML, it does not specify its own server code and focuses on C++ and C# implementations [22].

It is by far the easiest documentation-framework to understand and is very simple to write, however the main drawback of API Blueprint is the lack of advanced constructs. Due to this lack of advanced constructs, API Blueprint has not been largely adopted by the development community.

IV. Summary

TABLE I gives a summary of the popular available languages and models that can be used to describe REST-ful APIs. The table presents the main objective of each model as well as their drawbacks and output file formats.

V. Northbound API for SDWSN

When considering a northbound API for SDWSN, the same conclusion that has been stated for the northbound API in

TABLE I
SUMMARY OF DIFFERENT REST-BASED API INTERFACE MODELS

Model	Objective	Output Formats	Drawback
WSDL 2.0	Provide model describing web services Improve HTTP binding	XML	Lack of support tools Hard to understand/implement due to complexity Lack of backwards compatibility
WADL	Provide a machine-readable description of HTTP services REST equivalent to WSDL	XML	Lack of community support Incomplete descriptions Time-consuming manual documentation
RESTCONF	Mapping YANG data to a REST-ful interface	XML JSON	Still in development No existing API support
Swagger	REST API development platform based on OpenAPI Specification Automatically generate documentation for APIs	JSON XML	Lack's API testing interaction
RAML	Documentation and description platform API development platform	XML JSON	Lack of openness
API Blueprint	Documentation framework for REST APIs	Markdown	Low adoption Lack of advanced constructs

SDN also applies [10]. Due to the application specific nature of SDWSN brought about by WSNs and because of the fact that the main use cases for applications in the northbound interface are the integration and analysis of traffic, a unified and standard northbound interface API is unlikely. Therefore, in order to facilitate the use cases of northbound interface applications, the focus should instead be on the approach to standardized description of the northbound interface API.

The standardization of metadata support in northbound APIs results in the ability to integrate applications automatically without the need to develop a new API for various implementations of SDWSNs. However, as is the case with most of the frameworks shown above, the focus is mostly on human-readable documentation rather than programmatically discoverable descriptions (i.e. metadata). Therefore, a standard, metadata-based framework for REST-ful APIs must be developed as discussed below.

A. Metadata Framework Basic Requirements

When considering the basic requirements for the required metadata within the SDWSN domain, Sneps-Sneppe *et al.* [10] have proposed a basic set of elements based off of the WSDL specification as discussed below.

For reference, consider the Open Network Operating System (ONOS) REST API. ONOS is a SDN management operating system which has extensive documentation and support [27]. Their REST API also makes use to Swagger to automatically generate their API documentation.

The typical ONOS REST API request is similar to that shown below [28]. The request creates a new host based on JSON input and adds it to the current inventory.

```
POST /onos/v1/hosts
Accept: application/json
{
  "mac":"46:E4:3C:A4:17:C8",
  "vlan":"-1",
  "ipAddresses": [
      "127.0.0.1"
  ],
  "location":{
  "elementId": "of:0000000000000002",
  "port": "3"}}
```

To which the response would be the HTTP 200 status code, meaning successful operation, without any other headers or payloads (due to the fact that it is a POST request).

1) Endpoint Specification: The endpoint refers to the URL for the requests. URLs as well as resource paths should be described within the metadata. In the example above, the URL would be "/onos/v1/hosts" and the resource it specifies is "hosts".

2) Access Method Specification: The access methods for REST would be HTTP commands such as GET, POST, HEAD and PUT. In the case of the example above, this is POST.

3) Queries for Requests: Queries are parameters for requests and is not shown in the example but is still important to define and should be described within the metadata. An example would be a GET request in order to obtain a host with a certain ID, for example

```
GET /onos/v1/hosts/id=0000002
Accept: application/json
```

In this case, the query is the "0000002", which is the ID of the host that is being requested. The response would then be the HTTP 200 status code and some payload pertaining to the host's information, in JSON format.

4) HTTP Headers: This allows additional information to be passed with the request or response and could contain valuable information. In the example above "Accept: application/json" provides important information as it specifies the input format as JSON.

5) Status Codes and Error Messages: This referes to the status of the response, in the example above the status code would be 200 and the error message successful operation.

6) Version: The version provides information about limitations to the current implementation. In the example above /v1/ states that it is the first version of ONOS.

7) Formatting: The final useful element that should be described is the input or output file formatting, for example, JSON or XML (due to the fact that the ONOS API uses the Swagger documentation framework). In the example, the input file is JSON.

B. Potential Models

Now that the basic requirements for the metadata framework have been identified, existing solutions can be explored in order to determine whether they can facilitate the development for a metadata standard in SDWSN. Three model specifications have been identified which can potentially facilitate the framework without the need to develop an entirely new metadata-based API, these models being WADL, the OpenAPI specification and RESTCONF.

1) WADL: The utilization of the WADL specification for documentation of REST-ful applications, although not widely adopted, has been previously identified as a promising attempt to add metadata support to REST-ful services [9], specifically in the development of SDN northbound interface APIs. The WADL specification itself has resulted in the development of other projects such as REST-ful Service Description Language, but as is the case with WADL itself, these projects have not gained traction within the development community.

It has been shown how the WADL specification can be used to describe a tool that can be used for REST-ful application development [29]. This tool was named WSDawin and is able to automatically generate WADL descriptions for REST-ful services, which can then be used to perform tasks such as version comparisons and client proxy generation. They generated WADL documents for three real world REST APIs which were then compared to the WADL schema in order to validate the generated documents. It is important to note that not only did this tool document the REST API using the WADL specification, but it also provided machine-readable documentation which can be used for the management of client applications.

Although this tool is not entirely what is required when considering the northbound interface API within SDWSN, it is an example of how the WADL specification can be used to provide a metadata framework. However, the fundamental problem with using the WADL specification is due to the fact that generating documentation using the specification is a mostly manual and time consuming process [22] (hence the appeal to develop tools such as WSDawin), as well as the fact that the WADL specification has also been known to inconsistently describe REST-ful services. The specification sometimes fails to describe the link between resources as well as duplicate descriptions. This is why the WADL specification has not been adopted by the development community.

2) OpenAPI Specification: The OpenAPI specification [30] (formally known as the Swagger specification) is the foundation for the Swagger framework. As stated, it aims to provide a standard for the description of REST-ful APIs and when used it is able to provide service interaction with minimal implementation logic. The main appeal of the OpenAPI specification, ignoring the fact that it is open source and contains a vast amount of development tools and community support, is due to the fact that OpenAPI can bind itself to any existing API and therefore there is no need to develop a completely new API.

Musyaffa *et al.* [30] investigate the automation of REST-ful services by creating semantics for lightweight service descriptions using the OpenAPI specification. Their work results in an approach to describe services by extending the OpenAPI specification, thus adding semantic descriptions which allows for the automation of service discovery. Therefore, third-party applications can be automatically integrated on the web service using these descriptions.

Additionally, Haupt *et al.* [24] present a framework for the structural analysis of REST APIs by using Swagger documentation as the first validation of their framework and then transforming the OpenAPI specification into a metamodel for REST APIs.

This again illustrates how the OpenAPI specification can be used to develop metadata support for REST APIs. Combined with the fact that the OpenAPI specification is language independent, thus allowing it to be implemented on systems that run on multiple platforms using a variety of different languages, the OpenAPI specification is a good consideration for the development of a metadata specification for northbound interface APIs in SDWSN.

3) RESTCONF: Although RESTCONF is a relatively new management protocol that is still under design, it is already has a foot in the door of the SDN development community as evident in the fact that the OpenDaylight controller includes RESTCONF on its northbound interface [31]. Within the SDN domain, RESTCONF provides programmability and APIs for controllers due to the fact that NETCONF and YANG is able to describe the devices and resources, and REST is used to access them.

Daradkeh *et al.* [10] state that RESTCONF would be a promising platform for metadata within the SDN domain due to the fact that the YANG defines the semantics of datastore content configured within NETCONF as well as other information such as operational data. This hierarchical data can then be accessed using REST-ful operations. The only challenge would be expanding the RESTCONF protocol to develop a REST API.

This also presents the disadvantage of RESTCONF as an API. Unlike the OpenAPI specification where documentation can be generated on existing REST APIs, RESTCONF would need to be developed to consider the API aspect. This may result in the development of an entirely new API that is built on RESTCONF protocols. Although there is a RESTCONF agent included in the OpenDaylight controller within the SDN domain, which could assist in the development of a RESTCONF-based API, it would be better if the RESCONF protocol could be implemented on any API as is the case with the OpenAPI specification.

VI. Discussion and Remaining Challenges

There is much need for improvement within the northbound communications of SDWSN. This is due to the fact that there has been a greater focus on southbound communications as well as the SDWSN architecture itself. The fundamental problem within the northbound communications is the lack

TABLE II
SUMMARY OF POTENTIAL PLATFORMS FOR METADATA FRAMEWORK

Model	Advantages	Drawback	Potential metadata framework
WADL	XML based description (machine understandable)	Lack of community support Incomplete descriptions Time-consuming manual documentation	No
RESTCONF	Existing network functionality description Already within the SDN domain (OpenDaylight) Built on NETCONF and YANG	Still in development May not be able to implement on existing APIs	Yes
OpenAPI	Vast development tools Good community support Can be adopted by existing APIs Language independent	Lack's API testing interaction	Yes

of network programmability as a result of REST-ful API development.

A standard northbound API to achieve network programmability is unlikely due to the application specific nature of SDWSN brought about by WSN. Therefore, in order to elevate this application dependence, a standardized metadata-based framework is needed which can be applied to any existing REST-ful API. This will result in greater network programmability which can be used to automate application management within the northbound interface.

TABLE II provides a brief summary of the potential platforms for a metadata framework. When considering the development of this metadata-based framework, both the OpenAPI specification and the RESTCONF protocol seem to be promising candidates to provide a platform for the development of this framework. WADL, although previously though to be a promising specification, is not promising due to the fact that its mostly manual and time consuming documentation generation and lack of community support.

Due to the fact that RESTCONF is still being developed, it will have to be further developed if it is to be used as a REST API metadata framework and therefore it may require the development of entirely new APIs, although once fully developed, the protocol could be adapted to consider documentation for existing APIs.

Therefore the OpenAPI specification is identified as the most likely candidate for a metadata framework for REST APIs. Not only does the specification have a vast amount of community support and development tools, but it can also be applied to existing APIs (for example the ONOS REST API) and because of its language independents, it can be implemented on various existing platforms.

One of the biggest remaining issue brought about by the development of a metadata framework is that of security. Exposing metadata descriptions is a potential security risk to the network as it exposes information that may be used to compromise the network. Therefore, the metadata framework must be developed in such a way that it takes into account security as well. This is also one of the design requirements for the northbound interface API.

VII. CONCLUSION

The design requirements for the northbound interface API within the SDWSN domain have been discussed and the issue of metadata has been identified as one of the crucial considerations to be addressed. The lack of a metadata-based northbound API results in the lack of network programmability and therefore the lack of automation within the network.

Due to the application specific nature of the northbound interface, it is unlikely that there will be a standardized northbound API for SDWSN, however, in order facilitate automation within the application layer, there is a need for a standardized metadata framework within the northbound API. The OpenAPI specification is recommended as a platform for this framework owing to its community support and its documentation of existing APIs. This metadata framework must describe the basic elements of the API at the same time addressing the issue of security.

REFERENCES

[1] D. Kreutz, F. M. V. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proc. Second ACM SIGCOMM Work. Hot Top. Softw. Defin. Netw. - HotSDN 13*, 2013, p. 55.

[2] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Comput. Networks*, vol. 52, no. 12, pp. 2292–2330, 2008.

[3] M. Mudumbe and A. M. Abu-Mahfouz, "Smart water meter system for user-centric consumption measurement," in *Proceedings of the IEEE International Conference on Industrial Informatics*, Cambridge, UK, Jul. 2015, pp. 993–998.

[4] A. Abu-Mahfouz, Y. Hamam, P. R. Page, K. Djouani, and A. Kurien, "Real-time dynamic hydraulic model for potable water loss reduction," *Procedia Engineering*, vol. 154, no. 7, pp. 99–106, Aug. 2016.

[5] A. M. Abu-Mahfouz, T. O. Olwal, A. M. Kurien, J. L. Munda, and K. Djouani, "Toward Developing a Distributed Autonomous Energy Management System (DAEMS)," in *Proceedings of the IEEE AFRICON 2015 Conference on Green Innovation for African Renaissancce*, Addis, Ababa, Ethiopia, 2015, pp. 14–17.

[6] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke, "A survey on software-defined wireless sensor networks: Challenges and design requirements," *IEEE Access*, vol. 5, pp. 1872–1899, Feb. 2017.

[7] M. Ndiaye, G. P. Hancke, and A. M. Abu-Mahfouz, "Software defined networking for improved wireless sensor network management: A survey," *Sensors*, vol. 17, no. 5:1031, pp. 1–32, 2017.

[8] K. M. Modieginyane, B. B. Letswamotse, R. Malekian, and A. M. Abu-Mahfouz. (2017, Mar.) Software defined wireless sensor networks application opportunities for efficient network management: A survey. Computers and Electrical Engineering. [Online]. Available: http://dx.doi.org/10.1016/j.compeleceng.2017.02.026

[9] M. Sneps-Sneppe and D. Namiot, "Metadata in SDN API for WSN," in *2015 7th International Conference on New Technologies, Mobility and Security (NTMS)*, 2015, pp. 1–5.

[10] Y. I. Daradkeh, M. Aldhaifallah, D. Namiot, and M. Sneps-Sneppe, "On standards for application level interfaces in sdn," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 10.

[11] S. W. Pritchard, G. P. Hancke, and A. M. Abu-Mahfouz, "Security in software-defined wireless sensor networks: Threats, challenges and potential solutions," in *IEEE International Conference of Industrial Informatics INDIN'2017*, Emden, Germany, Jul. 2017.

[12] A. Aissioui, A. Ksentini, A. M. Gueroui, and T. Taleb, "Toward elastic distributed SDN/NFV controller for 5G mobile cloud management systems," *IEEE Access*, vol. 3, pp. 2055–2064, 2015.

[13] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey, "VeriFlow: Verifying network-wide invariants in real time," in *Proceedings of the first workshop on Hot topics in software defined networks - HotSDN 12*, 2012, p. 49.

[14] S. Son, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Model checking invariant security properties in openflow," *IEEE Int. Conf. Commun.*, pp. 1974–1979, 2013.

[15] K. Yap, M. Kobayashi, R. Sherwood, T. Huang, M. Chan, N. Handigol, and N. Mckeown, "Openroads : Empowering research in mobile networks," *ACM SIGCOMM Comput. Commun. Rev.*, no. 1, pp. 125–126, 2010.

[16] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[17] C. Ferris and J. Farrell, "What are web services?" *Commun. ACM*, vol. 46, no. 6, pp. 31–, Jun. 2003.

[18] X. Feng, J. Shen, and Y. Fan, "REST: An alternative to RPC for web services architecture," in *First International Conference on Future Information Networks*, Beijing, China, Oct. 2009, pp. 7–10.

[19] L. Mandel. (2008, May) Describe rest web services with wsdl 2.0. IBM. [Online]. Available: https://www.ibm.com/developerworks/library/ws-restwsdl/

[20] Y. Xue, C. Zhang, and Y. Ji, "RESTful web service matching based on WADL," in *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, Xi'an, China, 2015, pp. 364–371.

[21] M. Jethanandani, "YANG, NETCONF, RESTCONF: What is this all about and how is it used for multi-layer networks," in *Optical Fiber Communications Conference and Exhibition*, Los Angeles, CA, USA, Mar. 2017, pp. 1–65.

[22] K. Sandoval. (2015) Top specification formats for REST APIs. [Online]. Available: http://nordicapis.com/top-specification-formats-for-rest-apis/

[23] OpenAPI Initiative, "OpenAPI-Specification," https://github.com/OAI/OpenAPI-Specification, 2017.

[24] F. Haupt, F. Leymann, A. Scherer, and K. Vulkojevic-Haupt, "A framework for the structural analysis of REST APIs," in *IEEE International Conference on Software Architecture*, Gothenburg, Sweden, Apr. 2017, pp. 55–58.

[25] W. Li and P. Svard, "REST-based SOA application in the cloud: A text correction service case study," in *World Congress on Services*, Miami, FL, USA, Jul. 2010, pp. 84–90.

[26] M. Bieg, "A Web-based Tool to Semi-automatically Import Data from Generic REST APIs," Master's thesis, Swiss Federal Institute of Technology Zurich, 2014.

[27] P. Berde, W. Snow, G. Parulkar, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. OConnor, and P. Radoslavov, "ONOS," in *Proceedings of the third workshop on Hot topics in software defined networking - HotSDN 14*, 2014, pp. 3–3.

[28] S. Lele. (2016) Generating swagger documentation for the REST API. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Wiki+Home

[29] M. Fokaefs and E. Stroulia, "Using WADL specifications to develop and maintain REST client applications," in *IEEE International Conference on Web Services*, New York, NY, USA, Jul. 2015, pp. 81–88.

[30] F. A. Musyaffa, L. Halilaj, R. Siebes, F. Orlandi, and S. Auer, "Minimally invasive semantification of lightweight service descriptions," in *IEEE International Conference on Web Services*, San Francisco, CA, USA, Jul. 2016, pp. 672–677.

[31] A. G. Prieto, A. Leung, and K. Rockwell, "Automating the testing of RESTCONF agents," in *IEEE International Symposium on Integrated Network Management*, Ottawa, ON, Canada, May 2015, pp. 984–989.