A Secure Northbound Interface for SDN Applications

Christian Banse and Sathyanarayanan Rangarajan
Fraunhofer AISEC
Garching b. München, Germany
Email: {firstname.lastname}@aisec.fraunhofer.de

***Abstract*—Software-Defined Networking (SDN) promises to introduce flexibility and programmability into networks by offering a northbound interface (NBI) for developers to create SDN applications. However, current designs and implementations have several drawbacks, including the lack of extended security features. In this paper, we present a secure northbound interface, through which an SDN controller can offer network resources, such as statistics, flow information or topology data, via a REST-like API to registered SDN applications. A trust manager ensures that only authenticated and trusted applications can utilize the interface. Furthermore, a permission system allows for fine-grained authorization and access control to the aforementioned resources. We present a prototypical implementation of our interface and developed example applications using our interface, including an SDN management dashboard.**

***Keywords*-Software-Defined Networking; SDN; network security; northbound interface; trust**

I. Introduction

Software-Defined Networking (SDN) is a new network paradigm which enables dynamic programmability of the underlying network infrastructure. This is achieved by decoupling the decision logic (the control plane) from the individual network elements (the data plane) [1]. According to the SDN architecture [2], established by the Open Networking Foundation (ONF)[1], *SDN controllers* act as entities which manage the control plane and offer the possibility for *SDN applications* to further extend the functionality of a network. Possible use cases for SDN applications include virtualized network functions and business applications that retrieve data from external sources, such as an enterprise resource planning (ERP) system, to enforce business policies on a network level. As seen in Figure 1, these applications communicate with the controller and have the ability to program the underlying infrastructure through interfaces, usually called *SDN northbound interfaces (NBIs)*.

However, current implementations of these interfaces have several drawbacks. Firstly, despite the importance of such critical interfaces, they do not offer security features required to monitor and enforce the access control requirements of SDN applications. Without these security features in place, by default, all SDN applications have full access to the underlying network enabling the possibility for potential malicious applications. Secondly, the lack of standardization of NBIs has forced current designs to be controller-dependent. As a result, they do not offer the flexibility required by developers to build applications across a wide range of controller platforms. Thirdly, most designs focus on applications which reside directly within the controller, for example as a controller module, rather than external applications.

In this paper, we present a web-based northbound interface which is secure, controller-independent, and supports the deployment of external applications. Its architecture adheres to a Security-by-Design approach and incorporates security features such as encrypted communication, capability-based trust management and a resource-based access control model. This allows to restrict access to only authenticated and trusted applications. By encrypting the communication between SDN applications and the controller, we ensure the confidentiality and integrity of the transmitted messages. Additionally, while most web-based NBIs only offer unidirectional communication, for example by an SDN application accessing resources of the controller, our architecture enables bi-directional communication. This allows us to push events, such as network topology and application state changes, from the controller to the SDN applications.

The rest of the paper is structured as follows. Section II gives an overview of related work on the topic of SDN security. Section III postulates design requirements for a secure northbound interface. Section IV provides an overview of our proposed design and Section V describes its prototype implementation. Finally, Section VI concludes this paper.

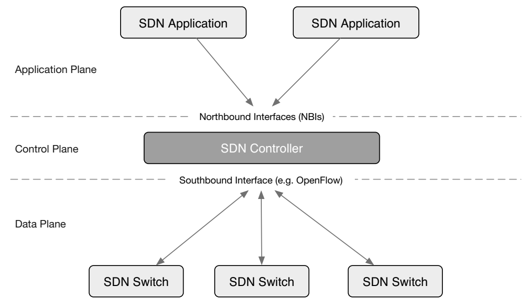

Figure 1. Generalized SDN architecture.

[1]http://www.opennetworking.org

II. Related Work

Previous work, including [3] and [4], has detailed security challenges and threats that arise from SDN. They primarily highlight the importance of achieving several security goals such as confidentiality, authenticity, integrity and availability of various entities within the SDN architecture. However, they do not propose any concrete security mechanisms required to achieve these goals.

To address some of these security goals, the authors of [5] have proposed *PermOF*, a permission system for OpenFlow implementations. However, the focus is on what the authors refer to as *OF apps*, which represent applications that reside directly within the control plane. This approach is only effective as long as the applications reside in the controller and fails to address the security challenges that arise while applications are deployed external to the controller.

In a similar approach, [6] proposes the *OperationCheckpoint* system which enables the restriction of applications using the API of the open source controller Floodlight. This approach, however, is controller-dependent and also lacks a mechanism to verify the authenticity of external applications.

Additionally, SE-floodlight[2], which is an extension of FortNOX [7], also provides a controller-dependent solution and is more focused on addressing the security issues surrounding application residing within the controller.

Lastly, the authors of [8] examined the possibility of malicious network services, by analyzing the security of different controller implementations. As a countermeasure, they introduce a sandbox which controls system-level operations of modules running inside a controller.

Thus, no solution for a controller-independent secure integration of external SDN applications, as we introduce in this paper, exists.

III. Design Requirements

In order to support the design of a secure and developer-friendly northbound interface, this section formulates a set of requirements that are to be fulfilled.

A. Security Challenges and Goals

To adhere to a Security-by-Design approach, several security challenges and goals have to be addressed when designing a web-based northbound interface.

1) Confidentiality: The confidentially of messages exchanged between the applications and the controller may contain sensitive information, such as the internal topology of networks or sometimes even user data, and therefore needs to be ensured. Additionally, network resources managed by a controller must not be accessible by applications that do not possess the appropriate permission.

[2]http://www.openflowsec.org/Technologies.html

2) Integrity: The integrity of the transmitted messages needs to be protected as well. This is required in order to protect against a malicious entity which may modify commands issued by a legitimate application, resulting in faulty or harmful messages. Furthermore, the integrity of the controller and SDN applications themselves also needs to be protected. However, since this paper is focused on the communication aspect between the entities, this is not within our current scope. It is, however, a potential challenge that we want to address in the future. Promising technologies that can be leveraged include remote attestation techniques and integrity monitoring of the container or host that the application is running on.

3) Authenticity: Additionally, the authenticity of all involved communication parties, such as the controller and SDN applications, needs to be enforced. This is necessary in order to prevent the deployment of malicious applications that masquerade as legitimate ones.

4) Accountability: Lastly, it must be possible to trace the source of each request on the northbound interface. In case of a misconfiguration or an attack on the network, the faulty request can then be traced back to its originating application.

B. Application Life Cycle

In order to achieve the design of a developer-friendly interface, we propose, that a northbound interface needs to at least partially support an SDN application's life cycle.

1) Deployment: As a prerequisite, the actual source or binary code of an application needs to be deployed. Common scenarios include deployment on physical machines, VMs or cloud infrastructures, such as OpenStack[3] or Amazon EC2, as well as new container technologies, such as Docker[4]. Regardless of the type of deployment, the application requires network connectivity to the SDN controller. Furthermore, this stage should also encompass the creation of certificates for all involved communication parties as well as the exchange of trust anchors, if a non-public Public Key Infrastructure (PKI) is used.

2) Registration: In the next step, the controller should be made aware of available SDN applications. For example, the SDN controller can offer a registration service, which applications can use to register themselves. To fulfill the security goals confidentiality and authenticity, this registration process needs to incorporate an identification of the application as well as the negotiation of access control mechanisms, such as permissions.

3) Operations: During operations, an application accesses, modifies or creates network resources through the controller. To achieve the aforementioned security goals, communication between the SDN applications and the controller needs to be encrypted. Furthermore, each resource

[3]http://www.openstack.org
[4]https://www.docker.com

Table I
RESOURCES OFFERED BY THE CONTROLLER SERVICE

Resource	Associated Permission Groups
Hosts	permission.hosts.view
Switches	permission.switches.view
Applications	permission.applications.view
Application Events, Topology Events, Message Events	permission.applications.events, permission.topology.view, permission.messages.* (depends on message, see Table II)
Network Topology	permission.topology.view
Network Statistics	permission.messages.statistics
Messages	permission.messages.* (depends on message, see Table II)

request needs to undergo an access control check, to ensure that the application is authorized for the particular request.

C. Controller Independence

As stated in Section II, most of the previous work related to securing northbound interfaces has been controller-dependent. We, however, want to achieve the design of an interface that is open, flexible, and independent from the underlying controller platform.

A possible way to achieve this, is to rely on a web-based interface and a service-oriented approach. In this approach, the controller offers the possibility to access and modify resources through a web-service. The web-service acts as an abstraction of the actual controller implementation.

IV. DESIGN

Based on the requirements stated in Section III, we propose the following design of a northbound interface (see Figure 2). Being heavily influenced by the REST paradigm [9], a controller service offers resources to applications by using standard HTTP methods such as GET or POST.

Our proposed interface does not fully comply to all constraints of the REST paradigm, mainly because it is not completely stateless. It is necessary for the controller service to keep state information in order to implement registration, authentication and authorization of its applications. Hence, we refer to our interface as being *REST-like*.

A. Controller Service

The *Controller Service* is a web-service comparable to existing controller REST APIs and offers a set of resources representing the SDN network. Table I provides an overview of resources that a controller service should offer according to our architecture. Depending on the actual implementation and the designated role of the underlying SDN controller, the service may offer additional resources, e.g. virtualized networks or routers.

All resources are associated with a certain set of permissions. These permissions are enforced every time an application is trying to access, modify or create a specific resource. The granularity of permissions can vary, as demonstrated in the resource *Messages*, which allows the creation of SDN control messages. For example, an application might be allowed to create control messages containing new flow rules, but not allowed to create messages related to configuration.

In our reference design, we utilize the popular SDN protocol OpenFlow [10]. However, our approach is generic and the control plane protocol could be exchanged or even a multitude of different protocols could be used. Furthermore, the mapping of these permissions is configurable. We have chosen a specific set of permissions based on the functionality of the OpenFlow message type (see Table II).

Table II
MAPPING PERMISSIONS TO OPENFLOW MESSAGES

Permission	OpenFlow Messages
permission.messages.configuration	OFFeaturesRequest, OFFeaturesReply, OFSetConfig, OFGetConfigRequest, OFGetConfigReply
permission.messages.flow	OFFlowMod, OFFlowRemoved
permission.messages.packet.incoming	OFPacketIn
permission.messages.packet.outgoing	OFPacketOut
permission.messages.port	OFPortMod, OFPortStatus
permission.messages.statistics	OFStatsRequest, OFStatsReply

B. Events

An application can register several *listeners* to receive the following event types:

1) Topology Events: An application can be informed about changes in the topology, e.g. when a new switch is added to the control plane or a switch is removed. It needs to possess the permission *permissions.topology.view* to receive this kind of information.

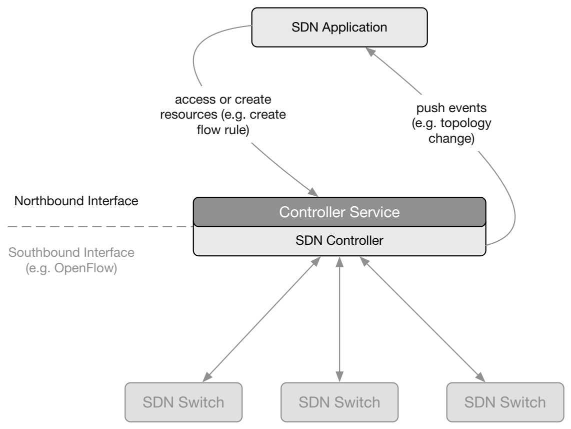
SDN Application
access or create resources (e.g. create flow rule)
push events (e.g. topology change)
Northbound Interface
Controller Service
SDN Controller
Southbound Interface (e.g. OpenFlow)
SDN Switch
SDN Switch
SDN Switch

Figure 2. Service Architecture.

2) Message Events: An application can register to receive OpenFlow messages originating from the switches or from the controller itself. The application only receives those messages for which it has been granted permissions.

3) Application Events: Lastly, by registering this listener, an application can receive all activities originating from other SDN applications. This is useful for the implementation of monitoring applications. It requires the permission *permission.applications.events*.

C. Trust Management

All communication end-points in our proposed architecture require valid certificates, e.g. based on X.509 [11], [12]. The SDN controller needs to possess a valid server certificate, called the *Controller Service Certificate*. This certificate must be issued to the domain name of the host running the controller and it must be signed by an appropriate certificate authority (CA). As a first step in establishing the trust relationship between the controller and SDN applications, the SDN applications need to either directly trust this particular CA or trust one of the CAs in the certificate chain.

Furthermore, each application needs to possess an individual *Application Certificate*, signed by a CA representing the vendor of the application. We denote the certificate of this CA as *Vendor Certificate*. Since a commercial application is usually deployed to multiple customers, it is recommended to generate individual application certificates for each deployment.

With each deployed application having its own set of application and vendor certificates, it is easy to establish a trust management based on the aforementioned certificates. This trust management can then be used to restrict access to the northbound interface to only authenticated and trusted applications.

D. Application Registration and Permission Management

Each application that requests access to the interface needs to undergo a registration process. During this process, the application can request a set of permissions that it requires. The controller service can then decide, whether these permissions seem appropriate or not. There are several approaches for this decision process.

1) Manual approval: Similar to installing an application on certain smart phone operating systems, the administrator or network operator will have to manually approve the application registration. This would mean, that the flow of the application life-cycle is stopped until a human decision is made. While this would certainly be the most careful approach, it is also impractical to a certain degree. For example, even if well-known applications are trying to re-register with the controller, the manual re-approval process would be triggered.

2) White-listing based on Trust: This approach requires that the administrator maintains a list of possible vendors and a respective *trust entry*. A trust entry basically contains a list of permissions that applications from a particular vendor are allowed to request. Furthermore, an increased level of granularity can also be obtained by creating trust entries for specific applications.

Within our architecture, since all applications and their vendors can be identified by certificates, we propose a hybrid approval approach based on trust. To automate the approval during the registration process, a decision is made based on the trust entry of the application or its vendor. If no trust entry is found for a particular vendor or application, the registration is denied or forwarded to manual approval.

Furthermore, during runtime of an application, all requests are checked against the registered set of permissions. Hence, those requests which exceed the allowed set of permissions are blocked while those which fall within the registered set are carried out successfully.

E. Secure Communication

All connections from applications to the *Controller Service* must be established using Transport Layer Security (TLS) [13] with mutual certificate-based authentication. In the proposed architecture, an application certificate is used as a TLS client certificate. We recommend the usage of TLS protocol version 1.2 in combination with a cipher suite that supports the Diffie Hellmann key exchange with ephemeral keys, to achieve the paradigm of *Perfect Forward Secrecy*. With perfect forward secrecy, the disclosure of the long-term key material does not lead to the compromise of the exchanged keys used to encrypt a session [14]. Hence, when used with TLS within our architecture, the disclosure of the private asymmetric key of the controller does not lead to the leakage of the ephemeral key used to encrypt the actual communication between two parties.

Furthermore, it must be ensured that weak ciphers such as RC4 [15] are excluded by the endpoint's TLS configuration. Instead ciphers which are still considered secure, such as AES, are to be used.

V. Implementation

To evaluate the effectiveness and usability of our design, we created a reference implementation of the proposed northbound interface. We deployed our implementation in the SDN test bed at Fraunhofer AISEC. This testbed consists of physical SDN switches (e.g. HP 3500-24), using OpenFlow 1.0 [16] as well as several Open vSwitch virtual switches using the newer version 1.3 [10] of OpenFlow.

Our reference implementation is Java-based, mainly because it is extending a number of open source projects, also written in Java. However, the design of the API is platform-independent and can also be implemented in other programming languages. By using the Oracle implementation of

Java 8, the TLS versions 1.1 or 1.2 are automatically chosen as default when securely communicating with web-services through the Java Secure Socket Extension (JSSE)[5].

A. Controller Service

The implementation of the controller service is based on the Java API for RESTful Services (JAX-RS) and its reference implementation Jersey[6]. By following a resource-based approach in our architecture, it is easy to export internal controller objects (such as switch, host or topology information) through JAX-RS as a web-service. Furthermore, Jersey offers an extension to support Server-Sent Events (SSE)[7], which is used to implement the event system.

The controller service needs to keep track of the state of applications to support registration, authentication and authorization.

First, an application needs to register itself with the *Controller Service*. This is done by issuing a POST request with a JSON or XML object which includes, among other things, the name of the application and the permissions it requires. The fields contained in the registration object are matched with the certificates exchanged in the associated TLS session. The name of the application must match the common name of the application certificate and the vendor name must be represented in the vendor certificate.

The *Trust Manager*, looks for a specific a specific trust entry matching either the application name or the vendor. If a trust entry is found, the set of allowed permissions contained in the trust entry are compared against the requested permissions. If the requested permissions exceed the allowed ones, the application is *Rejected*. Otherwise, the state of the application is set to *Registered*. In any case, an identifier for each application is generated by applying a SHA-1 hash to the application certificate's distinguished name (DN) as well as the vendor certificate's DN.

After a successful registration, the application can now issue commands to the controller service and receive events by registering the appropriate event listeners. During the runtime phase, the certificates exchanged in the associated TLS sessions are used to identify the individual applications. This is done by mapping the SHA-1 hash of the application and vendor certificate's DN to an application ID.

To enforce the requirement for access control, each command issued by an application is checked against the set of allowed permissions. The permission system is implemented using the Java *annotation* paradigm[8], which allows the annotation of meta-data to individual Java classes. In our implementation, a Java class representing a certain resource is annotated with a set of required permissions. While accessing this particular resource through its Java class, the controller retrieves the associated permission annotations. If the associated permissions exceed the allowed permissions for the issuing application, the command is blocked.

To not be constrained by placing permissions directly into the source code, we employ the Java project annox[9]. It enables the storage of annotations in external XML files, allowing us to re-configure the mapping of permissions to resources without the need to change the source code.

As mentioned before, the controller service is independent from actual the underlying SDN controller implementation. We make heavy use of the Java interface paradigm to achieve this. It is necessary to develop a controller-specific stub, forwarding the request to the actual internal controller code. For our reference implementation, we developed this controller-specific module for Floodlight[10], an open-source SDN controller, written in Java. It supports OpenFlow versions up until 1.4 through a flexible OpenFlow library.

B. Applications

We developed a Java framework for SDN applications, which allows them to use the northbound interface offered by the controller service. It utilizes the REST client implementation of JAX-RS, to register the application with a controller and to issue commands after a successful registration.

To test our framework, we created several example SDN applications that access or modify the network through the designed northbound interface. Specifically, we implemented a simplistic rule-based firewall application and an SDN dashboard which serves as a management console for the network.

This dashboard makes use of almost the full range of functionality that the controller service is offering. It includes an overview of all applications and network nodes, such as switches and hosts.. It registers several event listeners to be immediately informed about changes in the topology. Additionally, the dashboard is able to provide insight into the configuration of the trust manager, to see the set of permissions certain applications or vendors can request.

The dashboard makes use of the fact, that the northbound interface offers the possibility for an application to retrieve all events generated by other applications. By observing these events, it is possible to calculate a *Security Status* for each application. This status can have the following characteristics, which are assigned to the colors red, yellow and green, respectively.

- *Critical*: An application has shown a critical policy violation, e.g. by requesting permissions during registration that exceed the assigned trust.

[5]http://docs.oracle.com/javase/8/docs/technotes/guides/security/enhancements-8.html

[6]https://jersey.java.net

[7]http://www.w3.org/TR/eventsource/

[8]https://docs.oracle.com/javase/8/docs/technotes/guides/language/annotations.html

[9]https://java.net/projects/annox

[10]http://www.projectfloodlight.org/floodlight/

- *Warning*: A warning is triggered, if an application shows unusual behavior during runtime, e.g. by executing commands which exceed the allowed permissions.
- *Good*: The default case, if no policy violations or other warnings occur.

This way, an administrator can use the dashboard to easily observe all SDN applications and check whether they are operating according to their assigned policy.

VI. Conclusion

Software-Defined Networking (SDN) promises to introduce flexibility and programmability into the network by extracting the control plane of a network into a dedicated *controller* entity. By offering a *northbound interface*, the functionality of a controller can greatly be extended. However, current approaches have several drawbacks, including the lack of extended security features and the dependence on a specific controller implementation.

In this paper we present an open, flexible and secure northbound interface based on a *REST-like* architecture. The NBI utilizes an encrypted connection to maintain the confidentially, authenticity and integrity security goals. Furthermore, the integrated trust manager enables the restriction of the interface to only authenticated and trusted applications. Finally, a configurable permission system allows the enforcement of an authorization concept.

In future work, we want to further strengthen the security level of our approach by introducing an integrity monitoring of external SDN applications.

Acknowledgment

This work has been partially funded in the project SASER-SIEGFRIED by the German Federal Ministry of Education and Research under the reference 16BP12301.

References

[1] O. N. Foundation, "Software-Defined Networking: The New Norm for Networks," Open Networking Foundation, Tech. Rep., 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[2] "SDN architecture," Open Networking Foundation, Tech. Rep., June 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf

[3] L. Schehlmann, S. Abt, and H. Baier, "Blessing or curse? Revisiting security aspects of Software-Defined Networking," in *Network and Service Management (CNSM), 2014 10th International Conference on*, Nov 2014, pp. 382–387.

[4] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "SDN Security: A Survey," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, Nov 2013, pp. 1–7.

[5] X. Wen, Y. Chen, C. Hu, C. Shi, and Y. Wang, "Towards a Secure Controller Platform for Openflow Applications," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 171–172. [Online]. Available: http://doi.acm.org/10.1145/2491185.2491212

[6] S. Scott-Hayward, C. Kane, and S. Sezer, "OperationCheckpoint: SDN Application Control," in *Proceedings of the 2014 IEEE 22Nd International Conference on Network Protocols*, ser. ICNP '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 618–623. [Online]. Available: http://dx.doi.org/10.1109/ICNP.2014.98

[7] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for openflow networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 121–126. [Online]. Available: http://doi.acm.org/10.1145/2342441.2342466

[8] C. Röpke and T. Holz, "Retaining Control Over SDN Network Services," in *Proceedings of the International Conference on Networked Systems*, ser. Netsys'15. IEEE, 2015.

[9] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *ACM Trans. Internet Technol.*, vol. 2, no. 2, pp. 115–150, May 2002. [Online]. Available: http://doi.acm.org/10.1145/514183.514185

[10] "OpenFlow Switch Specification Version 1.3.4 (Wire Protocol 0x04)," Open Networking Foundation, Tech. Rep., 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.3.4.pdf

[11] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280 (Proposed Standard), Internet Engineering Task Force, May 2008, updated by RFC 6818. [Online]. Available: http://www.ietf.org/rfc/rfc5280.txt

[12] P. Yee, "Updates to the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 6818 (Proposed Standard), Internet Engineering Task Force, Jan. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc6818.txt

[13] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246 (Proposed Standard), Internet Engineering Task Force, Aug. 2008, updated by RFCs 5746, 5878, 6176, 7465. [Online]. Available: http://www.ietf.org/rfc/rfc5246.txt

[14] W. Diffie, P. C. V. Oorschot, and M. J. Wiener, "Authentication and Authenticated Key Exchanges," 1992.

[15] A. Popov, "Prohibiting RC4 Cipher Suites," RFC 7465 (Proposed Standard), Internet Engineering Task Force, Feb. 2015. [Online]. Available: http://www.ietf.org/rfc/rfc7465.txt

[16] "OpenFlow Switch Specification Version 1.0.0 (Wire Protocol 0x01)," Open Networking Foundation, Tech. Rep., 2009. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf