A RESTful Northbound Interface for Applications in Software Defined Networks

Abdullah Alghamdi[a], David Paul[b] and Edmund Sadgrove[c]
School of Science & Technology, University of New England, Armidale, NSW 2351, Australia

Keywords: Software Defined Networking, Northbound Interface, RESTful.

Abstract: Software Defined Networking (SDN) aims to help overcome the complexities inherent in traditional networks. The main concept in SDN is the decoupling of the data layer from the control layer, the latter of which is centralised in a controller. OpenFlow has been adopted as the standard protocol for the southbound interface, where the controller communicates with forwarding devices. However, the northbound interface (NBI), connecting the controller with end-user business applications, does not have an open standard. NBIs have accelerated application development because developers can implement required functionality without the need to consider matters related to the data layer, but there is an issue of compatibility because each SDN has its own NBI. In this position paper we present a plan to design a RESTful NBI for SDN applications to improve compatibility across SDN technologies.

1 INTRODUCTION

A rise in the number of devices connected to the Internet has made network management difficult. Some problems are widespread, including configuration errors, increasing sizes of routing tables, and issues related to security (Akcay & Yiltas-Kaplan, 2017). The inflexible behaviour of traditional network elements makes it hard for network administrators to manage them.

Software Defined Networking (SDN) is a relatively new technology to design and control networks (Singh & Jha, 2017). It is a network programming framework that allows administrators to intelligently and centrally control networks using software applications. SDN networks are inexpensive, relatively easy to implement, and provide opportunities to innovate with new applications (Zhang, Cui, Wang, & Zhang, 2018). SDN has support from many vendors, including Google, Cisco, and HP (Shahid, Fiaidhi, & Mohammed, 2016).

Network applications can be written to achieve different functionalities in a network, such as improving security or traffic management. To ensure network applications perform correctly, it is necessary for them to understand the current state of the network. An SDN architecture changes the way network state is maintained and made available to applications (Scott-Hayward, Kane, & Sezer, 2014).

SDN separates the control function from forwarding devices, logically centralising the control function that maintains network state in a controller, and having it send instructions to forwarding devices in the data layer. The forwarding devices then use these instructions to forward incoming packets appropriately. The interface between the controller and forwarding devices is called the Southbound Interface (SBI). The Open Network Foundation (ONF), which works to popularise SDN techology through the development of open standards (Open Networking Foundation, 2021), considers OpenFlow (McKeown et al., 2008) to be the standard SBI.

However, for SDN to reach its full potential, it is also necessary for applications to communicate with an SDN controller, both to determine the current state of the network, and to give commands which can be applied over the forwarding devices. SDN applications can either be internal, reacting to events that occur on the network, or exernal, proactively

[a] https://orcid.org/0000-0003-0616-4121
[b] https://orcid.org/0000-0002-2428-5667
[c] https://orcid.org/0000-0002-8710-9900

Alghamdi, A., Paul, D. and Sadgrove, E.
A RESTful Northbound Interface for Applications in Software Defined Networks.
DOI: 10.5220/0010713300003058
In *Proceedings of the 17th International Conference on Web Information Systems and Technologies (WEBIST 2021)*, pages 453-459
ISBN: 978-989-758-536-4; ISSN: 2184-3252

modifying the network without considering network events.

The interface between applications and an SDN controller is called the Northbound Interface (NBI) and there is currently no open standard NBI that can be used by all controllers (Du, Lee, & Kim, 2018). Instead, different incompatible interfaces have been implemented for various SDN controllers (Latif et al., 2020). This leads to a loss of compatibility, requiring significant time and resources to port applications to different controllers (Coutinho, 2017).

The motivation for this paper is to contribute to efforts to standardise the NBI to enhance portability of SDN applications and interoperability between SDN controllers. To better define the problem, Section 2 reviews SDN, and especially the NBI, in more detail. Section 3 then describes some requirements for an open RESTful Application Programming Interface (API) to allow both external and internal SDN applications to communicate with a controller. Finally, Section 4 concludes the paper to summarise our position and describe the next steps required in this research.

2 LITERATURE REVIEW

2.1 Software Defined Networking

In a traditional network, both the control and data planes are contained within a single entity. The control plane acts as the brain of the network, adding instructions to tables that are then consulted by the data plane to determine how to handle incoming and outgoing packets. Network nodes utilise the control plane to communicate with other nodes in the network through the use of distributed protocols such as BGP (Rekhter, Li, & Hares, 1994), OSPF (Moy, 1998) or MPLS (Rosen, Viswanathan, & Callon, 2001). Data from other nodes can then be used to modify the information stored in each node's tables.

SDN separates the control functions from forwarding devices in the hope of overcoming limitations in traditional networks (Haji et al., 2021). Figure 1 presents a high-level overview of a typical SDN architecture. The figure shows that the control layer communicates with the data layer through a southbound interface (SBI) using the OpenFlow protocol (Priya & Radhika, 2019). There is also a northbound interface (NBI) from the control layer that allows communication with applications, though there are no open standard protocols for this purpose (Latif et al., 2020).

In the lowest layer, forwarding devices could be either traditional hardware switches that provide a programmable interface, or software switches such as Open vSwitch (Wang et al., 2020). When a packet arrives at an SDN forwarding device, the forwarding device parses the packet's header to determine whether it already knows how to handle the packet, or whether it needs to communicate with the control layer.

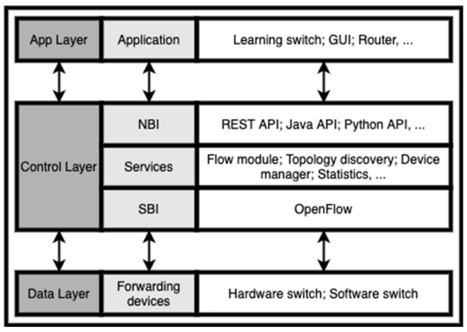
App Layer
Application
Learning switch; GUI; Router, ...
Control Layer
NBI
REST API; Java API; Python API, ...
Services
Flow module; Topology discovery; Device manager; Statistics, ...
SBI
OpenFlow
Data Layer
Forwarding devices
Hardware switch; Software switch

Figure 1: SDN architecture.

The control layer is implemented as a logically centralised network operating system called a controller. The controller has a global view over all forwarding devices in the data layer and uses the SBI to communicate packet forwarding instructions (e.g., whether to modify, drop, or forward the packet) to them. These instructions are called flows, and forwarding devices deny by default unless a flow specifies otherwise. The SBI is also used to alert of packet arrivals, notify of any status change, and to provide statistical information. OpenFlow is the standard protocol for all interactions between a controller and any forwarding devices (Dang et al., 2019).

To allow proper management of the network, controllers maintain some core services. These typically include:

- Topology service: builds a network topology graph by instructing switches to send certain packets and discovering where they arrive.
- Inventory service: tracks SDN devices attached to the network and records basic information about them.
- Host tracking service: discovers IP and MAC addresses of hosts connected to the network.
- Statistics service: provides network statistics based on counter information in switches.

The controller can use these services to provide a view of the network to any network application using its NBI. Unlike with the SBI, there is no standard protocol for the NBI, with different SDN controller implementations providing their own interface. For example, some controllers only offer a Java API (Goransson, Black, & Culver, 2016), or, when a more open RESTful API is provided, it is not standardised and is designed only to work with one particular controller (Comer & Rastegarnia, 2019).

Regardless of the implementation, the NBI provides an abstracted view of the network to any network applications. This abstracted view does not necessarily match the physical network implementation. For example, it is often sufficient for a network application to view the entire network as a single large switch, even if the network is actually implemented using many different forwarding devices.

The controller uses the NBI to notify applications of events that occur in the network. Such an event could be a packet being received by the controller or a change in the network topology. Applications can also invoke methods on the controller through the NBI to effect change on the network. For example, a firewall application may modify which packets should be dropped by forwarding devices after detecting a potential denial of service attack.

2.2 SDN Applications

SDN network applications can be classified into two different types: internal or external. Internal applications are reactive – a packet arrives at a forwarding device and the device does not know what to do with it, so it contacts the controller. The controller then notifies the application, which determines how the packet should be handled and informs the controller to implement the new policy. External applications, on the other hand, are proactive and modify network policy without requiring a packet to arrive first.

Further, internal applications typically create resources that are added to the network and can be accessed by other applications. For example, a load balancer may expose its resources so other applications can query it or modify its behaviour. Thus, an internal application becomes part of the programmable network, whereas an external application can only be controlled from the outside.

The controller's NBI consists of two parts: the Listener API and the Response API. The Listener API allows applications to register listeners for any relevant events. Registered listeners are then sent any relevant packets that arrive at the controller. The Response API allows applications to modify the network managed by the controller. The generic design of an internal application is presented in Figure 2. An internal application processes packets obtained through the Listener API and then makes calls to the Response API based on the packets it receives. External applications are similar, except no listeners are required, so the Listener API is not used, and the Response API is called without first requiring a packet to arrive.

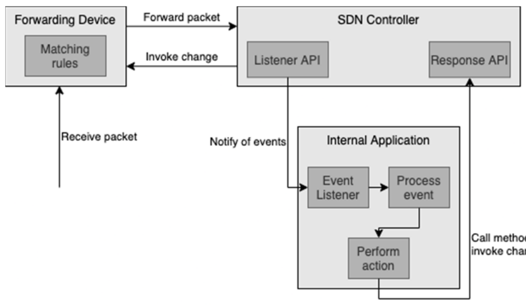

Figure 2: Generic design of an internal SDN application.

A RESTful interface is often used for the Response API because it offers the following advantages:

- Simplicity: REST utilises simple HTTP methods to access data and resources.
- Flexibility: All data and resources are represented as URIs, meaning there are no complicated schemas.
- Extensibility: New resources can be accessed by simply using the appropriate URI.
- Security: Communications can be secured by using HTTPS.

While the Response API is typically provided as a RESTful API, the Listener API is required to provide asynchronous notifications of incoming packets (Goransson et al., 2016). Since RESTful APIs operate on a request-response basis, which does not allow for such asynchronous notification, the Listener API is typically implemented as a native API on the controller (Banse & Rangarajan, 2015).

Unfortunately, in current SDN implementations, even the Response API is not standardised. The situation is even worse for the Listener API, where the interface provided typically depends on the language in which the controller is implemented (Goransson et al., 2016).

2.3 Northbound Interfaces

There is currently no standard NBI API defined for SDN. Instead, each SDN controller defines its own specific definition. Unfortunately, this means that applications that are written for one controller typically cannot be used in a network using a different controller without substantial redevelopment. This is despite the fact that the NBI is often considered the most important interface in an SDN architecture (Tijare & Vasudevan, 2016) because it is what allows the network to truly be programmable.

The typical approaches to overcome the issue of incompatibility between different SDN controllers are to either create applications on an ad-hoc basis, just for the controllers they need to interact with, or to use an SDN programming language that implements translators to convert application requirements into service requests for supported SDN controllers (Tijare & Vasudevan, 2016). While translators for different SDN programming languages are useful, this approach does require new translators to be developed any time a new controller or SDN programming language is developed. Thus, both approaches are quite inefficient.

The task of standardising an NBI is difficult because different SDN applications can have very different requirements. For example, a load balancer is likely to have significantly different needs than a security application. Because of this, many different NBIs have been proposed (Tijare & Vasudevan, 2016), though they typically only cover a limited set of operations or technologies.

Because each controller implements its own NBI API, there are numerous existing interfaces that can be studied. For example, NOSIX (Yu, Wundsam, & Raju, 2014) and SFNET (Yap, Huang, Dodson, Lam, & McKeown, 2010) provide ad-hoc APIs customised to each controller's needs. Others use SDN programming languages, such as Nettle (Voellmy & Hudak, 2011), Pyretic (Reich, Monsanto, Foster, Rexford, & Walker, 2013), Procera (Voellmy, Kim, & Feamster, 2012), or Frenetic (Foster et al., 2011), which can provide a variety of powerful abstractions but depend on control functions and data layer behaviour of particular controllers (Tijare & Vasudevan, 2016). Still others do provide a RESTful API, but these often change between different versions of a controller, leading to incompatibility (Li, Chou, Zhou, & Luo, 2016).

3 DISCUSSION

Our aim is to design an open, flexible, and independent NBI API for SDN. Most existing proposals suffer from having been designed in the early stages of SDN when there were few SDN controller implementations and limited practical experience writing applications for such systems, and have since been modified as systems have developed. We believe the time is now right to consider the lessons that can be learnt from the existing implementations, including their limitations, to design a complete new NBI API. We will design this API based on RESTful ideals, keeping it open to strengthen interoperability and portability of applications. Our design will follow the ONF guidelines (Open Networking Foundation, 2016) and contribute by recommending an NBI that covers a wide range of use cases.

As mentioned in Section 2, the NBI of an SDN controller can really be split into two different interfaces: the Listener API and the Response API. The purpose of the Listener API is to allow notification of events, while the Response API should allow applications to get information from the controller and to program the network.

For the Listener API, notifications should be available for at least the following events:

- Flow added
- Flow removed
- New device added to network
- Device removed from the network

The Response API can be divided into reading actions, which give details of the current state of the controller/network, and writing actions, which modify the network.

Reading actions should include:

- Read topology
- Read statistics
- Read flows
- Read controller information
- Read incoming packet

Writing actions should include:

- Insert flow
- Modify flow
- Delete flow
- Forward packet
- Set priority

3.1 Listening API

As mentioned in Section 2.2, internal applications register listeners with the SDN controller to be notified of relevant events. Since this event notification requires communication outside of a typical REST request-response, a RESTful service offered by the controller is not appropriate. Instead, internal applications are typically implemented using a native API (Goransson et al., 2016). For example, Floodlight (Project Floodlight)uses Java as its native language, so its modules are created as Java packages. These modules can then access underlying methods which may not be directly accessible via a REST call. To abstract over internal functions of a controller, programming languages such as Procera (Voellmy et al., 2012) or Frenetic (Foster et al., 2011) are often used, though this requires them to have support for the desired controller.

What we propose is having a REST-like service to allow registration of listeners with a controller. This service is only REST-like because the controller is required to maintain details of which applications are registered to particular events. Then, when a relevant event occurs, the controller calls a RESTful service implemented on the application side to notify it of the event, to which the application then responds appropriately.

3.2 Response API

In many regards, the Response API is the easiest to standardise into a RESTful service because it better fits the request-response pattern of REST (Goransson et al., 2016). Further, RESTful APIs are already offered by controllers such as ODL (OpenDaylight) and Floodlight (Project Floodlight). These APIs offer data about the network and methods to modify its behaviour, but are incompatible, even between different versions of the controller (Latif et al., 2020).

Thus, our aim is to provide a stable, extensible API that can be supported by multiple controllers. The API must support the entire lifecycle of SDN applications (Natanzi & Majma, 2017). This includes adding the application to the network, registering it with the controller, and conducting the required read and write actions.

The difficult part is defining minimal requirements for compliance, while allowing extensibility for controllers that offer more than the minimum. For example, after considering existing implementations, we believe that resources that must be supported by the API include: hosts; switches; applications; messages; network topology; statistics; and events, though other resources might be needed, and the exact details available for each resource might be different between implementations.

3.3 Backwards Compatibility

One of the advantages of this suggested approach is that it should be possible to allow backwards compatibility for compliant controllers and applications. Provided a controller offers the minimal functionality required by the API we are proposing, a small program could be written that converts requests to the new API into the native requests of the controller. This small program could then be used as the controller, with all other parts of the system unaware that it is communicating behind the scenes with another controller.

Similarly, from the application side, a small program could convert requests made by the application to the existing controller's native interface into the REST calls of the new proposed API. This small piece of code could also implement the RESTful interface of the Listening API to allow compatibility with internal applications that respond to network events.

This backwards compatibility can also allow evaluation of the new NBI API: if these small converter applications allow an application to function correctly with a controller that it does not natively support, then the new interface could be considered a success.

4 CONCLUSION

Traditional networks are difficult to control because the control and data layers are both integrated inside individual network devices. Further, each device typically has its own configuration and management interface. SDN separates the control and data layers and offers a programmable interface to dynamically control the network.

In SDN, controllers communicate with forwarding devices through a southbound interface, typically using the OpenFlow standard. However, the utility of SDN is really because applications can communicate with the controller via a northbound interface to query and modify the state of the network programmatically.

Despite its importance, the northbound interface has not been standardised. This means that different controllers and applications are incompatible. While some partial solutions exist, such as SDN programming languages that are compatible with multiple controllers, the better solution would be to define a standard open and extensible API for communication between an SDN controller and any networking applications.

The position argued in this paper is that the time is right to study existing SDN implementations to design a new RESTful API for the northbound interface of SDN controllers. This API will provide all necessary functions to support both external (proactive) and internal (reactive) applications.

The success of such an API can be determined by using small shim applications to allow a network application to correctly work with a controller it does not natively support.

A standardised northbound interface really is one of the big missing pieces in SDN. By examining the lessons from existing systems, an open, future-proof northbound API can improve compatibility of existing SDN implementations, saving time and effort and making adoption of SDN even easier.

REFERENCES

Akcay, H., & Yiltas-Kaplan, D. (2017). Web-Based User Interface for the Floodlight SDN Controller. *International Journal of Advanced Networking and Applications, 8*(05), 3175-3180.

Banse, C., & Rangarajan, S. (2015). *A secure northbound interface for sdn applications.* Paper presented at the 2015 IEEE Trustcom/BigDataSE/ISPA.

Comer, D., & Rastegarnia, A. (2019). Toward Disaggregating the SDN Control Plane. *IEEE Communications Magazine, 57*(10), 70-75.

Coutinho, D. R. (2017). *Contributions for the Standardisation of a SDN Northbound Interface for Load Balancing Applications*.

Dang, V. T., Huong, T. T., Thanh, N. H., Nam, P. N., Thanh, N. N., & Marshall, A. (2019). Sdn-based syn proxy—a solution to enhance performance of attack mitigation under tcp syn flood. *The Computer Journal, 62*(4), 518-534.

Du, S. G., Lee, J. W., & Kim, K. (2018). *Proposal of grpc as a new northbound api for application layer communication efficiency in sdn.* Paper presented at the Proceedings of the 12th International Conference on Ubiquitous Information Management and Communication.

Foster, N., Harrison, R., Freedman, M. J., Monsanto, C., Rexford, J., Story, A., & Walker, D. (2011). Frenetic: A network programming language. *ACM Sigplan Notices, 46*(9), 279-291.

Goransson, P., Black, C., & Culver, T. (2016). *Software defined networks: a comprehensive approach*: Morgan Kaufmann.

Haji, S. H., Zeebaree, S. R., Saeed, R. H., Ameen, S. Y., Shukur, H. M., Omar, N., . . . Yasin, H. M. (2021). Comparison of software defined networking with traditional networking. *Asian Journal of Research in Computer Science*, 1-18.

Latif, Z., Sharif, K., Li, F., Karim, M. M., Biswas, S., & Wang, Y. (2020). A comprehensive survey of interface protocols for software defined networks. *Journal of Network and Computer Applications, 156*, 102563.

Li, L., Chou, W., Zhou, W., & Luo, M. (2016). Design patterns and extensibility of REST API for networking applications. *IEEE Transactions on Network and Service Management, 13*(1), 154-167.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., . . . Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review, 38*(2), 69-74.

Moy, J. (1998). OSPF version 2.

Natanzi, S. B. H., & Majma, M. R. (2017). *Secure northbound interface for SDN applications with NTRU public key infrastructure.* Paper presented at the 2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI).

Open Networking Foundation. (2016). Intent NBI – Definition and Principles. Retrieved from http://opennetworking.wpengine.com/wp-content/uploads/2014/10/TR-523_Intent_Definition_Principles.pdf

Open Networking Foundation. (2021). Retrieved from https://opennetworking.org/

OpenDaylight. Retrieved from https://www.opendaylight.org/

Priya, A. V., & Radhika, N. (2019). Performance comparison of SDN OpenFlow controllers. *International Journal of Computer Aided Engineering and Technology, 11*(4-5), 467-479.

Project Floodlight Retrieved from https://www.projectfloodlight.org/floodlight/

Reich, J., Monsanto, C., Foster, N., Rexford, J., & Walker, D. (2013). Modular sdn programming with pyretic. *Technical Reprot of USENIX*, 30.

Rekhter, Y., Li, T., & Hares, S. (1994). A border gateway protocol 4 (BGP-4). In: ISI, USC Information Sciences Institute.

Rosen, E., Viswanathan, A., & Callon, R. (2001). RFC3031: Multiprotocol label switching architecture. In: RFC3031.

Scott-Hayward, S., Kane, C., & Sezer, S. (2014). *Operationcheckpoint: Sdn application control.* Paper presented at the 2014 IEEE 22nd International Conference on Network Protocols.

Shahid, A., Fiaidhi, J., & Mohammed, S. (2016). Implementing innovative routing using software defined networking (SDN). *Int J Multimedia Ubiquitous Eng, 11*(2), 159-172.

Singh, S., & Jha, R. K. (2017). A survey on software defined networking: Architecture for next generation network. *Journal of Network and Systems Management, 25*(2), 321-374.

Tijare, P., & Vasudevan, D. (2016). The northbound APIs of software defined networks. *International journal of engineering sciences and research technology, 5*(10), 501-513.

Voellmy, A., & Hudak, P. (2011). *Nettle: Taking the sting out of programming network routers.* Paper presented at the International Symposium on Practical Aspects of Declarative Languages.

Voellmy, A., Kim, H., & Feamster, N. (2012). *Procera: a language for high-level reactive network control.* Paper presented at the Proceedings of the first workshop on Hot topics in software defined networks.

Wang, H., Xu, H., Qian, C., Ge, J., Liu, J., & Huang, H. (2020). PrePass: Load balancing with data plane resource constraints using commodity SDN switches. *Computer Networks, 178*, 107339.

Yap, K.-K., Huang, T.-Y., Dodson, B., Lam, M. S., & McKeown, N. (2010). *Towards software-friendly networks.* Paper presented at the proceedings of the first ACM asia-pacific workshop on Workshop on systems.

Yu, M., Wundsam, A., & Raju, M. (2014). NOSIX: A lightweight portability layer for the SDN OS. *ACM SIGCOMM Computer Communication Review, 44*(2), 28-35.

Zhang, Y., Cui, L., Wang, W., & Zhang, Y. (2018). A survey on software defined networking with multiple controllers. *Journal of Network and Computer Applications, 103*, 101-118.