Tightening the net: security issues in software defined networks

Frans Ojala

ABSTRACT

In networking, decoupling the control layer from the data plane has gained tremendous interest in the past few years. Termed 'Software Defined Networking' (SDN) this separation promises to bring unprecedented control over the dynamics of the network configuration and data flows therein. To this end there has been significant research into creating an open protocol into network forwarding elements to enable their configuration and subsequent proliferation of network devices supporting SDN capabilities. One such protocol that has gained attention is OpenFlow. Despite the amount of interest, each new technology and protocol – OpenFlow not withstanding – is not without its flaws in its early stages. As the network becomes programmable new questions arise for instance about the correctness, verification, robustness, real time state and operation of the network and its controlling entity. While some of these questions have been answered, many still stand without answers. In this seminar paper we look into some of these problem domains primarily in the context of OpenFlow systems.

1. INTRODUCTION

The rise of cloud based computing has put more pressure on network infrastructure connecting the data centre. With the continually increasing number of computing components orchestration of the data centre network has become increasingly difficult. Cloud computing is not solely to blame. As the world develops there are more and more personal computers and servers being connected to the internet [1]. The internet of things (IOT) will bring even more pressure as it is estimated to grow by possibly orders of magnitude in connected devices by 2030 [2]. Managing the large amount of connections is a non-trivial task and it will take on even more responsibilities as time passes [1]. The current trend of pressure increasing toward our network infrastructure has been recognized in scientific communities and development of the second generation network has begun.

In current networks the control of the data flow path and the data flow itself are implicitly tied together. Many of the routing protocols employed today are restricted to a narrow local view of the network that is necessarily old because the protocols are at best reactive [3]. Often current routing protocols are time triggered further decreasing reactivity to anomalies within the network. Management of the network is done via tedious manual configuration of a single switch or router at a time via vendor specific API. Often verification of the network state is incomplete due to the limited nature of tools available to test network operation and the tremendous amount of work needed. Thus networks are often left misconfigured which results in time consuming debugging [4].

To make orchestration of increasingly large and complex networks easier the networking community has arrived at a simple, yet elegant, solution of separating the control of the data forwarding decision making (i.e. routing) from the data flows (i.e. packet forwarding). In essence the SDN paradigm centralizes the forwarding decision making and information distribution thereof into the control layer, while keeping the data flows themselves in the switches and routers. Often separation of the various monitoring, intrusion detection systems and firewalls – the application layer – is seen as another major part of SDN capabilities. To support such separation the entire network model needs to be redesigned so that the control and application layers are managed by a centralized authority and the packet forwarding (data plane) is the single responsibility of the network forwarding elements. Figure 1 illustrates this separation. [1]

In order for the network to be programmable from a centralized management entity the forwarding elements themselves need to be dynamically programmable. Just as there are protocols in place for switch-to-switch and router-to-router communication, there needs to be a protocol to enable the insertion of flow rules into the forwarding elements at run time from the aforementioned centralized entity and the export of flow metadata and statistics from the forwarding elements to the controlling entity. Export of flow metadata is crucial so that the controlling entity is able to make informed forwarding decisions. To enable intelligent real-time reaction to new flows there needs to be a mechanism to query the controller for new flow rules based on metadata. [3]

As noted, there has been significant research on SDN in the last few years. Numerous protocols have been developed and some are in active development [5]. Currently the OpenFlow protocol has received greatest attention and it has become today's de-facto standard that chip makers have elected to support in their network hardware. OpenFlow is being developed primarily by the Open Networking Foundation (ONF) that is dedicated to further the adoption of SDN. Although OpenFlow is not a new standard there are significant design challenges the protocol still faces. Securing the position of the authoritative controlling entity and subsequently the intended functions of the network is a major issue. [6] [7]

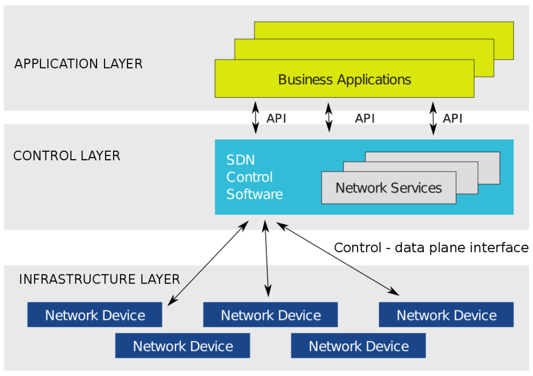

Figure 1: The SDN architecture

In the following, section 2 covers a vulnerability analysis the SDN faces in general. Section 3 will outline OpenFlow as a control layer protocol and SDN architecture. Section 3 covers also a vulnerability analysis of the OpenFlow protocol. Lastly section 4 discusses the mechanisms thus far developed in securing the network control logic within an OpenFlow enabled SDN.

2. VULNERABILITIES IN SDN

Interestingly, it seems that the benefits achieved are foreshadowed in threats by the same mechanisms that allow SDNs: centralising the control of the network. As underlined by [8] the main threat vectors consist of similar threats seen in current networks and computer systems of centralised nature:

1: Disruptive traffic flows can result from a faulty or misbehaving network element but can also be purposefully crafted by a malicious party. If not accounted for, may exhaust the flow table (or TCAM) of one or more switches or overload the controller in a manner similar to DoS attacks on web services seen today.

2: Switch implementation faults could be used to attack single or multiple switches to gain control of them. Once a switch is compromised all traffic flowing through it can be monitored and routed in ways favourable to the attacker.

3: Control plane communication can be attacked if it is not properly secured. Even a properly secured control channel is vulnerable to attacks if, for example, the underlying implementations for channel encryption and authentication are flawed, resulting in compromised switches. SSL/TLS implementations are known to contain bugs that have been exploited in the past.

4: Attacks on controllers can result in total loss of control over the network. If a perpetrator is able to exploit weaknesses in controller software they may be able to gain unbounded access and cause major disruptions in the network or network segment that is under the said controller.

5: Lack of application certification could result in an attacker gaining access to the controller via their own management application that was not certified by the controller. Lack of certification on the application layer would likely result at the very least to information disclosure or unauthorised access to the network. Worse, an attacker could trick the controller into injecting new flow rules undermining network security.

6: Gaining remote or physical access to the computers used to manage the controller is a threat vector in SDNs as much as it is in current networks. Any machine connected to the internet is vulnerable to attacks exploiting flaws in the programming of applications running on it.

7: Information disclosure vulnerability may reveal information about the internal state of a system by interacting with it. While SDN proposes to overcome the possibility of an attacker learning the state or presence of services on an internal network by masking network actions, there exist strategies which may in fact provide information to the attacker [9].

The authors also provide solutions that provide general protection against the aforementioned threats. **Replication** of the controller and applications render DoS attacks increasingly difficult to execute successfully. Faults in a minority of controllers could thus be mitigated by having controllers reach a consensus. Having the multiple different controllers increases **diversity**, that has been established to improve dependability. Network forwarding elements should be associated with

Threat	SDN specific	Consequence	Solution
Vector 1	no	can be a door for DoS attacks	Replication, Trust between controllers and devices
Vector 2	no	but now the impact is potentially augmented	Self-healing, Trust between controllers and devices, Fast and reliable update and patching
Vector 3	yes	communication with logically centralized controllers can be explored	Diversity, Dynamic switch association, Trust between controllers and devices
Vector 4	yes	controlling the controller may compromise the entire network	Replication, Diversity, Self-healing, Dynamic switch association, Trust between controllers and apps, Security domains, Secure components, Fast and reliable update and patching
Vector 5	yes	malicious applications can now be easily developed and deployed on controllers	Replication, Trust between controllers and apps, Security domains, Secure components
Vector 6	no	but now the impact is potentially augmented	Diversity, Self-healing, Fast and reliable update and patching
Vector 7	no	host discovery and learning attack vectors	Path randomisation, Diversity

Table 1: Threat vectors in SDNs vs traditional networks, possible solutions [8].

multiple controller instances if they don't support **dynamic controller discovery**. **Trust relationships** should be built between devices, applications and controllers to thwart injection of malicious variants. A common concept used in isolating outward facing systems is **security domains**. For example operating systems and browsers sandbox differing functions, web service providers use virtualisation to isolate products and services. Systems that are **self-healing** can automatically isolate malfunctioning components while the remaining components continue normal operations. Lastly, frequent **software updates** are the key to mitigating discovered vulnerabilities in any software system. Table 1 summarises consequences on the network and solutions that can be employed for mitigation.

3. OPENFLOW PROTOCOL

The core ideology of OpenFlow is based on the observation that most all switches and routers implement Ternary Contend-Addressable Memory (TCAM) for line-rate processing of data. More specifically that these TCAMs contain many similarities even across vendors. The TCAMs in use employ matching on incoming packet metadata and performing an action i.e. matching and acting on network flows. OpenFlow thus exploits this common set of functions to achieve a generalised view of a network forwarding element. At the core the OpenFlow (OF) protocol allows network administrators to program every switch and router in the same manner, to insert match-action rules into the forwarding elements' flow table through an open interface which enables a global view of the network at any given time. Essentially OF enables the network to be programmed according to high-level policies, rather than having to manually translate policies into vendor specific TCAM instructions. [1]

The flow table in an OF network element consists of header fields, counters and actions to be performed on a match. The headers supported for matching are [10]:

- Ingress Ports
- Ethernet source
- Ethernet destination
- Ethernet type
- VLAN ID
- VLAN priority
- IP source
- IP destination
- IP protocol
- IP ToS bits
- TCP/ UDP source port
- TCP/ UDP destination port

Each entry in the flow table contains an action to be performed on a matching packet and an associated counter. Match actions required include forwarding and dropping. A packet may be forwarded to all interfaces, the controller, to the local network or even back to the input port. Additionally an OpenFlow-switch may also support queueing and field modification. Field modification increases the abilities of the switch and thus provides even fine control over the network flows. Matches are also prioritized. Matches containing fewer wildcards are always higher in priority than those with more. Matches containing wildcards are also associated with a priority amongst themselves.

OpenFlow protocol features, in addition to the flow table, a specification of a SSL/TLS secured channel between the controller and the forwarding element. The channel is used to manage the flow tables from the controller and for sending statistics and forwarding unmatched packets to the controller from the forwarding element. The three message types – controller-to-switch, asynchronous, symmetric – make the core of the communication protocol [10].

Another important aspect of OpenFlow is the standardisation of the northbound API that is needed to develop *OpenFlow applications*. The applications – that serve as the rule creation logic – may act as firewalls, IDS, honeypots, load balancers and so on. As a result the applications can be intricately designed and utilised in manners not achievable in traditional networks. For further information regarding the OpenFlow protocol the reader should refer to its specification in [10].

3.1 Vulnerabilities in OpenFlow

There has been some research committed particularly to the security of the OpenFlow protocol and implementations [11]. Authors in [9] conducted a security analysis and empirical evaluation of Open vSwitch, a popular OpenFlow enabled virtual switch, specifically in the context of denial-of-service (DoS) and information disclosure attacks. In their analysis, the authors concluded from a Data Flow Diagram analysis that OpenFlow systems are vulnerable to at least the aforementioned DoS and information disclosure and additionally to tampering in the context of cache poisoning at the forwarding element.

The empirical analysis of Open vSwitch [12] was conducted in a virtual network setup. Their experiments show that, if a purely reactive strategy is used in the switch, then indeed there exists a great DoS risk on the control layer. The risk was evaluated to increase significantly with increasing rule timeout values, particularly at the 30 and 65 second timeout settings. Similarly the tests showed that an attacker may conclude from variations in the return trip times (RTT) the modification of switch flow tables on new purposefully crafted flows, making the Open vSwitch implementation of OpenFlow vulnerable to information disclosure. [9]

The aforementioned attacks, however, can be mitigated to some extent: DoS vulnerabilities by rate limiting, event filtering, packet dropping, timeout adjustment, flow aggregation and active attack detection are all mechanisms the authors propose for mitigation. Typically modern network forwarding elements already contain types of access control (lists) which may also be utilised in the event of DoS attacks. Information disclosure may be prevented by proactive strategies (the switch takes independent action until a specific rule is installed), rule timeout randomisation and again active attack detection and response. [9]

4. SECURING THE CONTROL LAYER

Another important security concern is that of network policies and their enforcement. While networks can be divided into virtual network segments and have different controllers managing them, policies exist even within segments. Policies can be used not just for network slicing, but also for access control, malicious flow detection and load balancing, among others. Moreover, the policies embodied within a network result from the total function of the currently standing flow rules in the network forwarding elements. Flow rule generation and insertion according to network policies is not an easy task. If careful measures are not taken the network may end in a state where a security policy is circumvented by rules that consecutively modify and route flows through virtual middleboxes, for example.[13]

The authors in [14] underline several challenges where the lack of policy mediation in OpenFlow systems hampers the adoption of SDN in production networks:

Multiple applications in the same SDN can generate a multitude of rules as the network evolves in time. In such a scenario it is not uncommon for applications controlling different aspects of the network to produce flow rules that would contradict each other. The basic question is, which application's rules are more important, and which rules should not be applied?

Flow modification allows complex virtual paths to be constructed. Flow rules generated by even a single OpenFlow application can be constricting on the flows or modifying their route. Multiple rules may form chains that are acted out the network forwarding elements. In effect it is thus possible to circumvent the dropping of packets to a destination by routing the flow through a middle point and modifying the headers of the flow.

Checking permissions to use a particular API call are not implemented in OpenFlow. The northbound API is very extensive, ranging from vendor specific commands to the switches to OpenFlow specific probes and statistics. Without a permission model any application can call on any API call. While this is not inherently a security risk it is inadvisable. If an application gets compromised it is possible for it to cause significant damage because its functioning is not restricted to a subset of API calls.

Origin of flow rules cannot be discovered in OpenFlow systems. This poses a challenge when considering for example the precedence of rules. Currently OpenFlow applications should handle rule conflicts among themselves. This hampers the modularity of applications as the arbitration becomes overly complex and results in monolithic applications.

Applications are thought of as libraries that are called upon from the controller. Applications are thus being launched in the process space of the controller and the controller is therefore vulnerable to all coding errors in all applications that are running within the controller. To mitigate for maliciously behaving applications or applications with coding errors the applications should be run in a completely separate process space, in a manner similar to most all modern desktop operating system.

4.1 SE-Floodlight

A way to enforce network policies, especially those concerning security, is to calculate for each new rule that is to be inserted the final resulting path that the flow will be directed through. To this end the authors of [14] have built a Security Enforcement Kernel, SEK, for the popular Floolight OpenFlow controller. The main task of the SEK is to accept or reject new rules based on network policies.

The SE-floodlight security enforcement kernel was designed to mediate all exchange that happens between the data plane and the application layer. The kernel employs a scheme according to which all operations are controlled. The scheme consists of a minimum authorisation that is associated with direction of control flow, operation and mediation policy. The three default authorisation roles along with a configurable scheme aim to provide a customisable permission model for any OpenFlow deployment. The major contribution in the kernel is the Rule-based Conflict Analysis (RCA), the purpose of which is to ensure that new flow rules don't conflict with existing ones and that the arising logic chains do not override any network policy who's authority is higher than that of the candidate logic chain. [14]

In SE-floodlight each application is assigned a group authorisation role and each application is authenticated via a multi-step procedure upon deployment. Each application is instantiated as a separate process and its messages to the SEK are accompanied by application credentials. All API calls are checked for each application and its role against the mediation scheme before they are allowed through. Upon submission of new flow rules the role, credentials and authorisation of the application are checked, after which the RCA algorithm checks if the rule would conflict with policies or other rules currently put in place by higher authorities. [14] Figure 2 illustrates the SE-floodlight architecture.

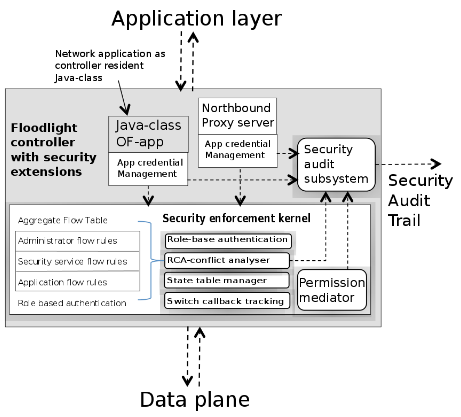

Figure 2: Overview of the SE-Floodlight architecture [14]

4.2 FRESCO

Policy enforcement is an important aspect that any venerable OpenFlow network needs in order to keep the network in its intended state. We have seen that the SE-floodlight SEK can be useful in this context. While the SEK can be used in policy enforcement, how should the logic that is needed to create the rules – the OpenFlow applications – be implemented? What design considerations should be tackled in order to make the northbound API safe as well as producing high quality security applications?

Once again, the authors in [15] underline some of the challenges that current OpenFlow systems face regarding the rule production logic, especially in terms of security:

Scarcity of information is one of the key problems today, as flow information is not captured equally by controllers. The flow information is important for the (security) applications to be able to make informed decisions about how to route data in the network.

Application linkage is currently poor in OpenFlow systems. As noted earlier, typical OpenFlow applications are built as monolithic structures that often have complex dependencies. To support modular development of applications the applications themselves need to be able to communicate and share information and trigger events across application boundaries.

Threat response is key to mitigating attacks to hosts in the network or to the network itself. Currently there are no frameworks for developing modular applications that are able to produce the often complex rule chains in order to efficiently mitigate attacks.

Enter *FRESCO*, an application development framework and SEK designed to combat the aforementioned problems. *FRESCO* consists of two major components: a development environment that sits on the application layer and a SEK that runs on the control layer. The applications are implemented as NOX Python modules and are enriched with calls to the *FRESCO* API. The development environment provides the applications with the API to access script-to-module translation, database management, event management and instance execution. The script-to-module translation takes a FRESCO script and creates a module that can be directly used as an instance in the environment. The database management module collects statistics and state information from the network flows and makes it available to module instances through an API. The event management module handles notifying instances about a specific event the instance has registered to get notified of so that it can take appropriate actions. The instance execution manages deploying of instances into the running environment including their authentication, authorisation and registration. The FRESCO resource controller monitors OpenFlow enabled switches to collect statistics and performs garbage collection on the least frequently used flow rules if the flow table is full beyond a certain threshold. [15]

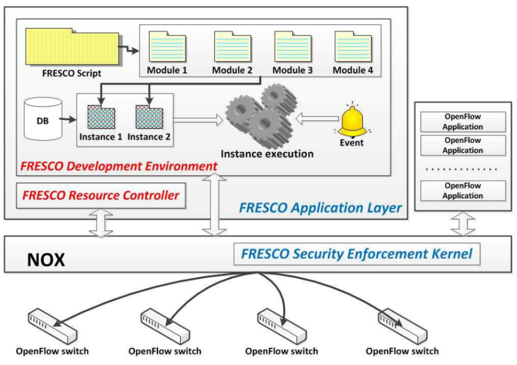

Figure 3: Overview of the FRESCO architecture [15]

On the other hand *FRESCO* presents the NOX OpenFlow controller with a Security Enforcement Kernel. The purpose of the SEK for NOX (FortNOX) is the same as with SE-floodlight: to ensure that network policies are adhered to on the level of forwarding elements. The SEK introduces a hierarchical authority model that along with policy management the network forwarding rule precedence is determined. Additionally the SEK incorporates rule conflict detection and resolution. Altogether the FRESCO framework aims to provide a comprehensive solution to network security policy management, deployment and enforcement. [13]

5. DISCUSSION

SE-floodlight, FortNOX and FRESCO provide solutions for enforcing security policies within OpenFlow enabled software defined networks. However, they themselves have similar issues as discussed in section 2 of this paper. While they can be powerful tools in separating malicious traffic flows and managing attacks they still suffer from potentially bottlenecking in the system. There are at least some components in each of the solutions that need to be centralised and there needs to be a single master instance that can be overwhelmed by modern botnet installations. To this end their performance should be put to the test in extreme environments. Additionally the systems are quite homogeneous, meaning that a weakness in one system may lead to the compromise of other systems built on the same components. In both cases of SE-floodlight and FRESCO the SEK will likely be the main target of DoS attacks and the OpenFlow applications attached to it will be the targets of DoS and infiltration attacks. Additionally, in FRESCO the database management module will be the target of DoS attacks as well because in the end the database will have to be either centralised or if it is distributed there needs to be a master instance. In the end, all the components of the systems should be considered to be potential targets of attackers solely for the reason that they are software. With that in mind any systems that are being implemented for use in connected environments should be constructed with care.

6. CONCLUSION

Software defined networking may be the next generation of controlling computer networks. Their power emerges from the centralised nature of the control layer, while only the data plane is kept at the switch. The standardisation of the southbound API is well on its way, as we have seen is the case with OpenFlow. The northbound API enables the attaching of applications that can provide virtualised network functions and therefore assist the control layer below in making decisions about network flows. We have also seen that the fine grained control that can be achieved through SDN does come at some cost, at least in the beginning.

The logically centralised nature of the controlling entity and open interface specifications enlarge the attack surface of SDN in comparison to traditional networks. Despite the risks, and particularly due to increasing interest in SDNs, they are being put to use and developed as our understanding increases. Several solutions to the most compelling problems are presented in the form of flow aggregation, rate limits, attack detection, specialised kernels and frameworks designed for the developing of SDN applications.

This paper has provided an overview of the current issues in securing control of the software defined network. While the area is still in development there have been significant contributions in the area and many vendors have chosen to start adapting their product line-up to the emerging technology. Finally many service providers have also begun the transition, and many are already there [16].

7. REFERENCES

[1] "ONF WHITE PAPER on software-defined networking: The new norm for networks," tech. rep., Open Networking Foundation, 2012. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[2] "Internet of things by the numbers: Market estimates and forecasts." http://www.forbes.com/sites/gilpress/2014/08/22/internet-of-things-by-the-numbers-market-estimates-and-forecasts/.

[3] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4d approach to network control and management," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 41–54, 2005.

[4] N. Feamster and H. Balakrishnan, "Detecting bgp configuration faults with static analysis," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pp. 43–56, USENIX Association, 2005.

[5] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[6] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "Sdn security: A survey," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN For*, pp. 1–7, IEEE, 2013.

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[8] D. Kreutz, F. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 55–60, ACM, 2013.

[9] R. Kloti, V. Kotronis, and P. Smith, "Openflow: A security analysis," in *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, pp. 1–6, IEEE, 2013.

[10] "Openflow specification 1.0.0." https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf.

[11] "Openflow sec," 2013. http://www.openflowsec.org/Publications.html.

[12] "Open vSwitch." http://openvswitch.org/.

[13] 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*, pp. 121–126, ACM, 2012.

[14] P. Porras, S. Cheung, M. Fong, K. Skinner, and V. Yegneswaran, "Securing the software-defined network control layer," in *Proceedings of the 2015 Network and Distributed System Security Symposium (NDSS), San Diego, California*, 2015.

[15] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson, "Fresco: Modular composable security services for software-defined networks.," in *NDSS*, 2013.

[16] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, *et al.*, "B4: Experience with a globally-deployed software defined wan," in *ACM SIGCOMM Computer Communication Review*, vol. 43, pp. 3–14, ACM, 2013.