State of the Art in Software Defined Networking – Lessons Learned and the Way Forward

Bilal Karim Mughal
Hyderabad, Pakistan
bilalkmughal@gmail.com

Abstract—While there have been multiple technologies, both hardware and software, that enabled programmable or software-defined networking (SDN) but while others have come and gone, Openflow is the one SDN-enabler technology that has stayed for good and has seen wide acceptance and deployment by major vendors and network operators alike. In this paper we provide an overview of the current Openflow research and point out the shortcomings or avenues for future research. We also discuss the novel applications enabled by this new technology and our experience with implementing some of them in our testbed network. We argue that openflow is the way to go and the technology of the future and provide arguments and evidence to support our views.

Index Terms—Openflow, software defined networking

I. Introduction

All hardware and software in a traditional network switch or router (switch and router are used interchangeably in this paper from here onwards) can be roughly divided into two planes: data and control. The data plane of a switch is the brawns: it encompasses mostly dumb hardware and software responsible for simple data forwarding. While the control plane is the brains: it encompasses software doing routing logic, QoS, rate limiting etc. So, control plane has the logic that runs traditional routing algorithms and based on their result decides on which incoming packets go where and through what port by installing forwarding rules in the data plane's FIBs (forwarding information base).

Openflow [8], the de facto realization of software defined networking (SDN), is essentially an architecture and protocol that cleanly separates the control plane and data plane of a switch extracting all control plane software from the device and putting it instead on a separate centralized server called the controller. The controller is completely programmable and thus enables software-defined networking to be an easy possibility because a network operator can now program the controller any way they want: they can handle each incoming packet of a switch separately or in groups based on certain criteria such as packets coming from a distinct IP or MAC address can be handled in groups. They can observe the packets for purposes of authentication by checking if the packet's IP/MAC belongs to an authorized user or not, they can do deep packet inspection (DPI) for intrusion detection, they can drop packets based on any rules they want thus essentially creating a custom firewall and completely preventing DDoS attacks by writing only a few lines of code in the controller. Once the controller code has been written and programming has been done by the operator, the controller then follows the Openflow protocol and talks to the data plane in the switch to fulfill the programmed tasks. The simplicity of the Openflow protocol is that all these complex and sophisticated tasks mentioned above like intrusion detection, firewall, or rate limiting are fulfilled by the switch by supporting only four kinds of functions for any incoming packet where the fourth is optional: (i) forward incoming packet through a certain switch port, (ii) drop incoming packet (iii) send incoming packet to controller, and (iv) optionally, collect statistics for the incoming packet.

This is the reason why openflow has been so widely accepted and deployed because it does not require any complex modifications in the switches, the required three functions can easily be provided by all switches and an additional benefit is that the switches can remain completely backward-compatible. This basically means that an operator can easily replace any traditional switch within their networks with an openflow-enabled one without changing or affecting any other component of their network. Also, openflow allows incremental deployment: you do not need to make each and every switch in the network openflow-enabled, even one openflow switch can be used with traditional non-openflow switches to gain the benefits provide by openflow.

Figure 1 show a typical openflow-enabled network with two openflow-enabled switches while there is one traditional non-openflow switch. The controller connects to only the openflow switches while the traditional switch works on its own without any controller involvement.

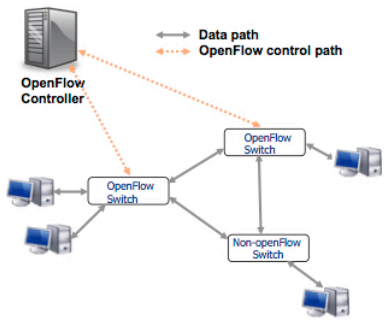

Fig. 1. An openflow-enabled network

To summarize, essentially, openflow turns black-box networking where all control logic and routing is within

various network devices and not accessible to operators only through diverse and difficult to use command line utilies into white-box networking where all control logic and routing is within a centralized piece of software called the controller from where the operator is able to see a global view of the entire topology, run traditional routing algorithms or write their own custom ones specialized for their network, and do a countless number of other functions such as QoS, firewalling, deep packet inspections, security or authentication checks, modify packet headers or even complete packets and forward them through any switch port they want, etc. Further, controller software is easily programmable and is available in a variety of languages ranging from C, C++, Java to Python and Ruby.

Although many networking paradigms and technologies to enable programmable or software-defined networks (SDN) keep getting proposed by researchers, they gain hype for a while but very few ever reach the actual implementation and deployment phase. Openflow (which we will be using as synonym for SDN in this paper) is different in that it is very much a reality now, the Openflow protocol itself is in its fourth version, major vendors like Cisco, HP, etc already have many fully functional Openflow-enabled switch models in the market, the controller software has evolved and is user friendly, Openflow hardware has been installed in major Internet backbones, research networks, many university campuses support it, and developers are increasingly creating applications that span a variety of domains from security and network management to optimized routing and advanced traffic engineering. Openflow has inarguably revolutionized the way we do networking by allowing the development and easy deployment of these novel applications.

This paper aims to capture a picture of current openflow research and applications or changes enabled by it in various domains as well as predictions and proposals for future work by providing a critique or pointing out weak areas in current research work. For every domain, we try to give an overview of the theoretical research and then discuss some practical research applications in relative detail. We also mention details from our testbed experiments of openflow that we did for some applications that are easy to program and install in a network. Thus this paper can serve as a starting point for both a network operator and researcher to gain a big picture of the novel applications enabled by Openflow, the feasibility of deployment in a given network, the ease of implementation, and also the drawbacks or avenues for improvement in current openflow research and suggestions on how to deal with them in future.

The rest of this paper is organized as follows. Section II provides a technical background for the openflow architecture and also a brief comparison of it with the traditional switch architecture. In Section III, we mention and discuss possible applications that can be enabled by openflow, our implementation experiences for some of the common ones, and some current research in openflow in various domains such as network security or network management. In our discussion we try to point out the advantages as well as give a critique for possible avenues for improvement and future directions that need to be taken. Finally, in Section V, we conclude the paper.

II. Architecture Background

This section discusses the openflow architecture [8] to provide the reader with a technical background to understand the applications discussed in later sections.

Figure 2 shows the typical setup for and distinction between a traditional switch and an openflow switch.

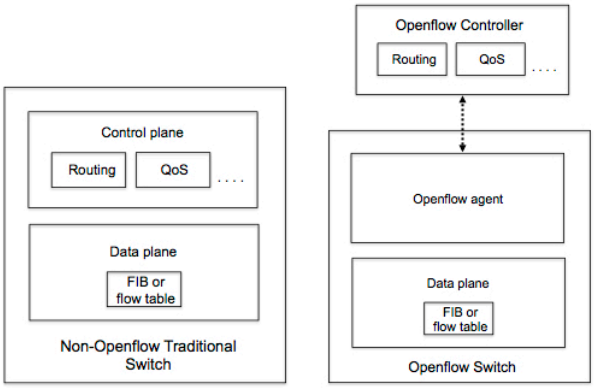

Fig. 2. A traditional non-openflow switch (left) and an openflow switch setup with controller (right).

In an openflow switch, the control plane logic is placed inside a separate controller running on a server. The controller runs all the routing and other logic software and sends commands and forwarding or dropping instructions to the switch. The openflow agent is a lightweight piece of software that runs on the switch and receives these controller commands and communicates them with the data plane by installing the information contained in the commands into the *flow table* of the data plane as *flow entries*. A *flow* in a network is a group of one or more packets that fit a certain criteria, such as, all packets that have the same sender IP address form a flow, or all TCP packets form a TCP flow, etc. In other words, a flow is defined in a customized way by the operator based on values in a packet's header. The values in a header may include things like the transport layer protocol of the packet or the TCP source or destination port, or the sender or receiver IP or MAC address of the packet, the higher level protocol type of packet such as whether it is an ICMP packet or a DNS packet, the VLAN ID of the packet, or the switch port the packet came in from (ingress port), etc.

So, flow entries are created and put inside the flow table by the openflow agent on behalf of instructions from the controller, where the instructions inside the controller are written by the network operator. Each flow entry defines a flow and its associated action. As mentioned previously, the actions or functions are as follows:

(i) Action 1: forward incoming packet through a given output port.

(ii) Action 2: drop incoming packet

(iii) Action 3: send incoming packet for processing to the controller.

(iv) Action 4: store statistics about this packet (This action is optional for openflow switches and is used for collecting monitoring or accounting statistics such as how many number of packets were received from a certain IP or MAC or on a certain VLAN, etc).

By default, for every incoming packet that does not already belong to a flow whose entry is inside the flow table, the agent executes action 3 for that packet. This means every new packet without a flow entry is sent to the controller and the controller then processes the packet and installs a flow entry for it. Once a flow entry has been created it persists for a certain amount of time until expiring (expiring time is set by controller too). Until the entry does not expire, all new packets that fit the criteria of the installed flow entry do not need to be forwarded to the controller (so action 3 is typically a one-time thing).

An example flow entry that uses action 2 would be as follows.

Flow entry: srcIP = 192.168.1.4 ; action = drop

Once the above entry has been installed in the flow table, then all incoming packets that have a scrIP equal to 192.168.1.4 will be dropped by the switch.

Another example is routing all packets with a certain destIP through a certain egress port of switch, the flow entry for this would use action 1 and would look like this:

Flow entry: destIP = 192.168.1.5 ; action = outPort 2

This entry would send all packets belonging to the flow with destIP equal to 192.168.1.5 through switch output port 2. This is essentially how custom routing or traffic engineering is done with openflow.

III. Openflow Applications in Various Domains

A. A Network Operating System (NOS)

Just as a traditional operating system (OS) hides the computer hardware and provides clean and easy to use interfaces to the system and application software developers, similarly an Openflow controller can serve as a network operating system (NOS) that hides the complexity of heterogeneous network hardware and provide the network operator with a unified and easy to use interface to program their network however they want. This NOS model enabled by openflow is attractive because it hides the complexity of having to configure heterogeneous network devices which may come from various vendors which usually have different configuration languages or low-level hardware commands that the operator has to learn. Using a controller only requires use of one high-level programming language to configure and run all of the hardware within the network. The main drawback here is that operators are already used to managing their networks manually in the traditional way and most of them specially those managing small networks do not see enough incentives or motivation to move to an openflow network and learn a high level language like Java or C++. But for operators managing moderate to large sized networks where there are a lot of devices to manage, openflow has a lot of incentives and motivation for switching to this new paradigm. And with more and more device vendors now brining out openflow-enabled devices, it is only a matter of time that openflow would become the norm.

There has been a much research already in creating user friendly controller frameworks, some of the commonly available research products that are now in production stage include Beacon [11], POX [14], Trema [13], and Floodlight [12]. Future work in creating a complete NOS still remains, there has only been some word in that regard including Google's B4 network implementation [10] that accomplishes some of the clean separation of interfaces and layered architecture similar to an operating system and Pyretic [9] which was the initial proposal that tried to bring forward this NOS idea with modular architecture like a computer OS.

B. Firewall

The controller can be used instead of a dedicated firewall so that selected incoming or outgoing packets that match a certain criteria are blocked or dropped. There has been work in creating reliable firewalls on top of openflow [22, 23, 24].

We created a small testbed using machines from the Emulab testbed [1] to test the firewall application. We ran the Beacon controller software and only had to write a few lines of code to get the firewall working so that the social networking service Facebook was blocked on the network. We took the IP address for Facebook and had our controller put flow entry such that for all incoming packets, action 2 was applied if the destIP matched Facebook's IP essentially disallowing anyone within our network to access Facebook.

Since dedicated firewalls are already typically deployed within networks so convincing operators to move to openflow specifically for firewalls is not very practical. But the good news is that openflow has many other benefits besides the firewall type protection and so motivating operators is not very unpractical anymore though research into performance of security using openflow controllers is still left for future work, security is a critical feature and so the controller security applications need to go through intensive testing phases before deployment especially for large networks.

C. Authentication Service

Openflow controller can also be used to create an authentication service so that only users with the right credentials or those who have paid for use of the network can have their traffic transferred. This can be implemented by having an authentication list maintained by the operator in the controller, such as, the list of users and their credentials. The code in controller can then say that traffic coming in from any IP address should first contain packets that have the login and password details, the packets are processed at the controller and the credentials extracted from them and checked against the list, if they match correctly then flow entries are installed in the switch flow table with action 1 which basically forwards this authenticated user's traffic. If the authentication failed then

the user's packets can be ignored and not let them pass through the switch any further.

D. Science DMZ

A Science DMZ [7] is a specialized portion of a network that is dedicated and optimized for high-performance, speed, security, and reliability for scientific data movement. Science DMZ are typically located on large campuses or research labs that generate massive amounts of scientific data. Campuses and labs often collaborate on experiments so there is a need for all this data to be moved from a DMZ on one campus to a DMZ on one or more other campuses or labs. An example of such collaborations that require such massive data movement include the Sloan Digital Sky Survey [2] that involves transfer of petabytes of data. Another major example is the large amounts of data generated by the Large Hadron Collider in Europe [4] that has to be transferred to various labs that are then responsible for the analysis of this data. All this data movement has traditionally required a lot of human involvement: operators at both the source and destination labs as well as those in the middle who control the network backbones have to be involved in the data transfer task. Further, since Science data requires speed, security and reliability so instead of normal routing, dedicated paths have to be set up by the collaborating network operators and firewalls that usually slow down the transfer speed have to be turned off specially for these transfers thus security is always a concern.

Openflow again comes to the rescue here. If there are even a few Openflow-enabled switches installed at the source, destination, and backbone then a centralized controller can be used to automate and optimize the routing task and there is no need for a separate firewall since the Openflow controller itself can secure the data as shown in the previous section. Thus only some controller code has to be written to automate all the collaboration: human involvement or use of dedicated physical paths is no longer needed.

There are already some efforts that have been made in this regard, various campuses and labs have started installing openflow switches in their networks and the major backbones such as Internet2 [5] and NLR [6] also include openflow switches within them to enable this large data movement but a lot more research is still needed that actually documents the performance of openflow-enabled data movements so that more and more Science SDN operators can be convinced to switch to Openflow. Also, only major campuses mainly in US and Europe have started deployment of openflow, so some more deployment efforts are needed before the openflow data transfers can become practical.

E. Security

The openflow controller can protect a network from many attacks since no packet traversing the network links can be hidden from the controller, the controller is always in a position to intercept any packets it desires. Openflow security work involves protection against DDoS using openflow [25], IP scanning and worm attacks [15], providing elastic IP and security groups [26], and general security monitoring [3], etc.

To show the practicality of security with openflow, in the rest of this sub-section we discuss an example of a very easy to implement and simple defense mechanism using openflow as provided in [15], they call it the OpenFlow Random Host Mutation (OF-RHM). This technique basically combines openflow and the concept of 'moving target defense' to provide network nodes with a defense mechanism against IP scanning-based attacks and worm propagation. Every network node or machine that needs to be protected is assigned a virtual IP by the controller, and the real IP is not accessible, as long as the real IP of the protected node is not available, attackers cannot mount attacks on this node and IP scanning attacks are out of the question.

The technique hides the real IP as follows. Any hosts trying to access the protected node first try to access the DNS for hostname resolution of the protected node. The DNS receives the request and sends back a response. The controller already has a flow entry inserted in the switch that tells the switch to intercept all DNS response packets and rewrite the resolved real IP address within the DNS response to a fake or virtual IP. This translation from real to virtual IP or this mutation of real IP is done with unpredictability by choosing the virtual IPs randomly from a large address space and mutation is done frequently by setting a short DNS TTL value. This way every host that tries to resolve the name of the protected host using DNS gets a new virtual IP of the protected host and thus increases protection. Further, the controller also inserts one more flow entry in switch telling the switch to intercept all packets coming in the network whose destIP is equal to the virtual IP of the protected host, after interception the switch is made to rewrite the

Further, the controller also inserts two more flow entries in the switch. The first one telling the switch that all packets going towards the protected host (i.e., whose destIP is equal to any virtual IP of the protected host) should be intercepted and their destIP rewritten to the real IP of the host so that they can be routed properly now. The second entry tells the switch that all packets going away from the protected host (i.e., whose sourceIP is equal to the real IP of the protected host) should be intercepted and their sourceIP rewritten to a virtual IP of the protected host so that the receiver of this packet does not get to find out the real IP from the packet header.

So, the whole algorithm requires a minimum of three flow entries to be installed to achieve protection.

The advantage with using openflow here is that it is very transparent to end hosts. On the other hand, using the traditional approach (the one without openflow) requires modification of end hosts and involvement of DHCP server in order to accomplish this hiding of real IP addresses. Further, it was observed that OF-RHM reduces the accuracy of IP scanning attacks by up to 99%, and saving of hosts from scanning worms by 90%. These are really motivating numbers in favor of openflow.

The main criticism we have about the openflow approach is that the packet header rewriting mentioned above should ideally be done within the switch rather than the controller (because reaching the controller for every packet rewriting

would increase latency) but some openflow switches do not support the rewriting feature, it has only been introduced in the newer openflow versions so this technique would only be practical if the switch being used supports the newer openflow version otherwise the latencies would get unbearable. Another disadvantage of this approach that should be taken as indication of required future work is that the DNS server itself is still unprotected [15] so some mechanism for its security is also required.

We implemented this algorithm in our test network and tested against IP scanning attack, it showed 100% security for all our tests. We had four hosts h1, h2, h3, and h4. Host h4 was termed as the host we were trying to protect and its IP was set to 10.0.0.10, host h3 was used as DNS server while hosts h1 and h2 were being used as the attackers or the hosts trying to access h4. Fig. 3 shows hosts h1 and h2 simultaneously trying to ping h4, the result from the ping in the two windows shows how both of them got different virtual IP address resolutions for the same name 'h4' and none of them was able to know the real IP 10.0.0.10 from using the DNS. This showed that the algorithm had worked correctly in fooling the attackers and protecting the host h4 from any IP scanning attacks to be mounted.

Before moving forward, one more criticism we have related to security using openflow in general is that the openflow protocol itself also requires some security analysis just to make sure that there are no security holes within the architecture or the protocol itself. There has been some research [21] in this regard but an extensive analysis still remains for future work.

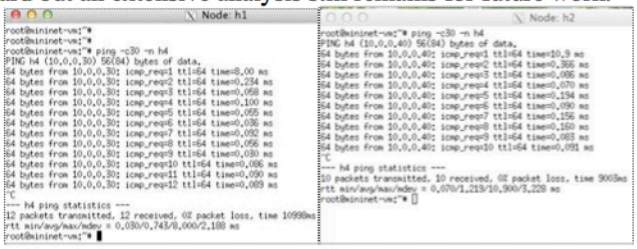

Fig. 3. Pinging host h4 from host h1 and h2 simultaneously: the name 'h4' resolves to IP 10.0.0.30 for host h1 and to IP 10.0.0.40 for host h2.

F. Network Management

Network management is defined by different people in different ways in the literature and in practical scenarios. It encompasses various tasks including capacity planning, network event correlation, root cause analysis of problems, auditing, monitoring and logging, planned maintenance, traffic engineering, enforcing business constraints or other policies, etc. Traditionally, all these tasks have been done manually by operators with very little automation, even when automation systems have been developed, they were standalone systems with no collaboration and so the operator has to spend hours and even days to solve a certain problem or find data they need by sifting through extensive logs generated by heterogeneous devices.

Recently, there has been interest shown by operators and research into coming up with a unified network management system that can automate all management tasks and that provides the operator with a unified centralized view of their network. Some related work in this direction includes systems such as Coolaid [16], Knowops [17], Darkstar [18], G-RCA [19], and Statesman [20], etc.

We believe that using openflow to support these systems can help with the goal of getting a complete unified view of the network since that is exactly what a controller is suitable for. Also, management tasks such as event correlation or root cause analysis or even configuration of devices can be made very fast and easy with openflow, since the operator doesn't have to learn complicated device configuration languages specially in large networks. Also, monitoring and gathering logs from devices also gets simplified since openflow already has a monitoring and auditing feature in its specification. Finally, we also believe that auditing or authentication or payment system involving checking which users have paid for services they are trying to access or not, all these tasks are very simple to implement using openflow as mentioned in the previous sections. Traffic engineering (TE) is also a very attractive feature relatively easily implementable with openflow than using the non-openflow approach, in fact, Google's B4 WAN incorporated it within their network and reported very positive results [10].

Thus, openflow seems like a very useful technology that can help the task of creating a unified network management system. The future work now involves incorporating its support within the above mentioned systems and recording the results as compared to the traditional approach by letting operators test it within real networks. We think a starting point for developing a complete openflow-enabled system would be to first work on the network 'read' and 'write' interfaces because these are the two components that provide input (by 'reading') to the entire network management system, the system then does computations or processing based on this input and finally produces output that is then converted into a set of configuration commands and written back to the network devices using the 'write' interface. Writing would basically involve using flow entries to configure network devices with the configuration setup produced by the system. The architecture of the system itself should be layered similar to Google's B4 WAN example [10] since a modular layered architecture is very flexible to use. We believe such a architecture is also attractive because some of the layers can be created and provided as part of the management system package to operators so that they do not have to write all the software layers themselves. They will only have to provide the system with their policies and information about the network, etc. and the system would take care of the other stuff. This obviously is a long-term goal that requires much effort before such a complete system can become a reality and be deployed. As the previously mentioned related work shows there are substantial attempts being made by researchers and interest being shown by major operators for the creation of the unified management system so using openflow or SDN to support this research will we think make it a the system a reality much faster than is possible with traditional non-openflow methods.

IV. Conclusion

This paper provided an overview of software defined networking enabled by the new Openflow standard. It discussed various applications enabled by Openflow implementation of the simple ones in our testbed. We discussed the changes openflow has brought about in how we do networking, the simplification and efficiency it has brought. We also mentioned current research in this field and pointed out directions for future work by providing a critique for the current research and applications. We believe openflow has a bright future, it is already considered the de facto protocol and architecture for enabling SDN and has seen acceptance from both manufacturers as well as operators as indicated by the large number of openflow enabled models now in the market as well as by the large number of deployments. While openflow definitely has great potential but it is still a relatively new technology and so its applications have only just begun to mature, we still need more research and analysis to prove its worth and use it up to its full potential.

References

[1] Emulab. http://www.emulab.net

[2] Sloan Digital Sky Survey http://www.sdss3.org/

[3] Ballard, Jeffrey R., Ian Rae, and Aditya Akella. "Extensible and scalable network monitoring using opensafe." *Proc. INM/WREN* (2010).

[4] Large Hadron Collider. http://home.web.cern.ch/about/accelerators/large-hadron-collider

[5] Internet 2. http://www.internet2.edu/

[6] NLR. http://www.nlr.net/

[7] Science DMZ. https://fasterdata.es.net/science-dmz/

[8] McKeown, Nick, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. "OpenFlow: enabling innovation in campus networks." *ACM SIGCOMM Computer Communication Review* 38, no. 2 (2008): 69-74.

[9] Monsanto, Christopher, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. "Composing Software Defined Networks." In *NSDI*, pp. 1-13. 2013.

[10] Jain, Sushant, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata et al. "B4: Experience with a globally-deployed software defined WAN." In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pp. 3-14. ACM, 2013.

[11] Beacon. openflow.stanford.edu/display/Beacon/Home

[12] Floodlight. www.projectfloodlight.org/floodlight/

[13] Trema. trema.github.io/trema/

[14] POX. openflow.stanford.edu/display/ONL/POX+Wiki

[15] Jafarian, Jafar Haadi, Ehab Al-Shaer, and Qi Duan. "Openflow random host mutation: transparent moving target defense using software defined networking." In *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 127-132. ACM, 2012.

[16] Chen, Xu, Yun Mao, Z. Morley Mao, and Jacobus Van der Merwe. "Declarative configuration management for complex and dynamic networks." In *Proceedings of the 6th International COnference*, p. 6. ACM, 2010.

[17] Chen, Xu, Yun Mao, Z. Morley Mao, and Jacobus Van der Merwe. "KnowOps: towards an embedded knowledge base for network management and operations." In *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services (Hot-ICE'11), Berkeley, CA, USA*, pp. 7-7. 2007.

[18] Kalmanek, Charles R., Ihui Ge, Seungjoon Lee, Carsten Lund, Dan Pei, Joseph Seidel, Jacobus Van der Merwe, and J. Ates. "Darkstar: Using exploratory data mining to raise the bar on network reliability and performance." In *Design of Reliable Communication Networks, 2009. DRCN 2009. 7th International Workshop on*, pp. 1-10. IEEE, 2009.

[19] Yan, He, Lee Breslau, Zihui Ge, Daniel Massey, Dan Pei, and Jennifer Yates. "G-rca: a generic root cause analysis platform for service quality management in large ip networks." *Networking, IEEE/ACM Transactions on* 20, no. 6 (2012): 1734-1747.

[20] Sun, Peng, Ratul Mahajan, Jennifer Rexford, Lihua Yuan, Ming Zhang, and Ahsan Arefin. "A network-state management service." In *Proceedings of the 2014 ACM conference on SIGCOMM*, pp. 563-574. ACM, 2014.

[21] Klöti, Rowan. "Openflow: A security analysis." *Proc. Wkshp on Secure Network Protocols (NPSec). IEEE* (2013).

[22] Qazi, Zafar Ayyub, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. "SIMPLE-fying middlebox policy enforcement using SDN." In*Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pp. 27-38. ACM, 2013.

[23] Hu, Hongxin, Gail-Joon Ahn, Wonkyu Han, and Ziming Zhao. "Towards a Reliable SDN Firewall." *Presented as part of the Open Networking Summit 2014 (ONS 2014)}* (2014).

[24] Suh, Michelle, Sae Hyong Park, Byungjoon Lee, and Sunhee Yang. "Building firewall over the software-defined network controller." In *Advanced Communication Technology (ICACT), 2014 16th International Conference on*, pp. 744-748. IEEE, 2014.

[25] Braga, Rodrigo, Edjard Mota, and Alexandre Passito. "Lightweight DDoS flooding attack detection using NOX/OpenFlow." In *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pp. 408-415. IEEE, 2010.

[26] Stabler, Greg, Aaron Rosen, Sebastien Goasguen, and Kuang-Ching Wang. "Elastic IP and security groups implementation using OpenFlow." In *Proceedings of the 6th international workshop on Virtualization Technologies in Distributed Computing Date*, pp. 53-60. ACM, 2012.