A Survey of SDN Security Research

Michael Coughlin
University of Colorado Boulder
michael.coughlin@colorado.edu

ABSTRACT

Software defined networking (SDN) has established a new method for creating and administering networks, but has also changed the attack surface that is presented by networks. SDN provides several features that allow for easy mitigation of certain types of attacks, such as DoS, and allows for mitigation of other attacks with more work. However, SDN also introduces new vulnerabilities that are not present in traditional networks, such as a communication bottleneck between the control-plane and the data-plane. Many new technologies and techniques have been proposed to solve SDN security vulnerabilities and some additional work can be applied address them as well.

Current research in SDN follows several identifiable trends that are related to the state of deployment of SDN technologies. As OpenFlow is the most popular implementation of SDN and is currently used in production settings, much research has been performed to utilize and improve the protocol. However, there is another research trend that has produced work that is applicable to SDN in general, including architectures that provide more flexibility than OpenFlow. Future research is likely to follow these trends by improving the OpenFlow protocol and proposing more general alternatives, and this research will include the further development of tools for the testing of network designs and the research of optimizations for OpenFlow when it is used in production environments. In this paper, I present a survey of current research on SDN security and other work in the field of SDNs that is applicable to security and a prediction of the directions of future research in SDN security.

1. INTRODUCTION

Software defined networking (SDN) is a networking design philosophy that advocates the separation of the network data-plane from the network control-plane. The data-plane represents all of the data that is being forwarded through the network, such as packets and the hardware that is used to forward it, such as switches. The control-plane represents all logic and devices that are responsible for deciding how and to where data in the data-plane is to be sent. Traditional networks combine these two planes on the same devices, forcing each device to make its own forwarding decisions based on distributed routing protocols. SDN, on the other hand, allows for the control-plane to have a global view of the network, allowing for policies to be applied that take into account all of the network state, rather than what is exposed to a single device.

This design philosophy evolved from the idea of active networks [19], which advocated the ability to embed programming inside of packets, so that computation could occur in the network along a packet's path. Active networks allowed for a large amount of flexibility in how packets are processed and how they traverse the network, but the system had issues with motivation and deployment, which eventually led researchers to investigate other fields. Little research is currently performed directly on active networks, but the active network concept led to the idea of SDNs. SDNs require the support of specialized hardware (as do active networks), but they present clearer motivating issues that exist in traditional networks, especially the issue of complex and inconsistent management [11].

The most popular deployment of SDNs, OpenFlow [15], implements the SDN design by providing a communication protocol that allows a centralized controller to communicate with and program specialized data-plane hardware. In essence, an OpenFlow controller writes forwarding rules to switches that specify actions to be performed on packets whose headers match a specific pattern associated with the forwarding rule. OpenFlow switches apply any action specified by a rule to a packet whose header is matched against the rule, and forward any packets that do not match an existing rule to the controller. The controller performs all necessary forwarding decisions, and the switches only apply actions that have been specified by the controller. Applications can also be written to interface with the control to add extended network-wide functionality, and the use of the terms *controller* and *application* can oftentimes be used interchangeably. This design methodology provides the ability to mitigate several classes of network attacks, but also opens the risk of new vulnerabilities, and potentially increases the risk of existing vulnerabilities in traditional networks [13].

Much of the current research into SDN security had focused on the OpenFlow protocol, but the vulnerabilities that exist in OpenFlow are likely to be generalizable to any SDN system that uses a centralized controller. The most common vulnerability that is noted by recent research [8, 13, 18] is the communication bottleneck that is between the data-plane and the controller in an OpenFlow network. Due to the centralized location of a controller in an OpenFlow network, it is simple to overload the communication path that is used by switches to communicate with the controller, even during normal operation of the network. However, as mentioned by Shine et. al.[18], it is possible for a knowledgeable attacker

to craft packets that will cause a standard OpenFlow network to quickly saturate the control communication channel, leading to some form of Denial of Service (DoS) attack in the network, possibly by preventing access to the controller, and at least preventing the creation of new flow entries. This issue can also manifest as a failure of the controller, and there has been much research on how to make the controller more fault tolerant [20, 9]. This work generally attempts to create a distributed but logically centralized controller, and the fault tolerant properties of these solutions would make them more resistant to a DoS attack.

Other recent work in SDN security research has sought to leverage the programmability and visibility that is provided by SDN in order to increase security against traditional attacks [17, 16, 14]. These solutions are not solely restricted to OpenFlow, but they do take advantage of SDN tenets. These technologies allow for the implementation of common mitigation strategies, including the use of middleboxes, detection algorithms and classification algorithms that are used to secure traditional networks. These solutions are useful for ensuring that SDNs are not vulnerable to known network susceptible, especially attacks targeted at hosts.

In this paper, I present a review of the current research into SDN security technologies, as well as some technologies that are directly applicable to SDN security. I also present a short overview of the evolution of SDN and a prediction of future SDN security research. Current security research in the field of SDN can be separated into OpenFlow-specific and general solutions, and specific to OpenFlow, can be split into research on protocol-specific and traditional network vulnerabilities.

Finally, I present an analysis of the trends in SDN security research and a prediction of the paths that future research will follow. Current research has placed a large focus on addressing the shortcomings of OpenFlow and there has also been an emphasis on the creation of network designs that incorporate flexibility that OpenFlow is not capable of. In terms of security, research trends show that the benefits provided by new network designs oftentimes introduce new vulnerabilities in the very features that differentiate them from the existing systems, as is illustrated by the OpenFlow control-plane communication bottleneck. To summarize my predictions that are detailed in Section 6, I predict that research will continue to be performed that attempts to address the shortcomings of OpenFlow due to its adoption by industry, including the creation of simpler development tools and the optimization and change of several of the protocol's features. I also predict that tools used to test SDN alternatives to OpenFlow, such as software switches and field programmable gate arrays (FPGAs), will be improved so that OpenFlow alternatives and hardware-side improvements to OpenFlow can be tested.

This paper continues in Section 2 by first presenting a brief overview of the evolution of SDN. In Section 3, I review several recent works in SDN security and how these specific works relate to different aspects of SDNs. In Section 4, I present a more detailed discussion of OpenFlow specific security technologies, and in Section 5, I detail the remaining technologies that are applicable to SDN in general. In Section 6, I present an analysis of trends in SDN security research and predixt how SDN research will progress in the future, and I conclude the paper in Section 7.

2. EVOLUTION OF SDN

Software defined networking has evolved from several different research tracks, starting with research into active networks. Although some of these tracks were unsuccessful, they were all motivated by the challenges faced by managing a growing Internet and the desire to have more flexible and programmable networks.

2.1 Active Networks

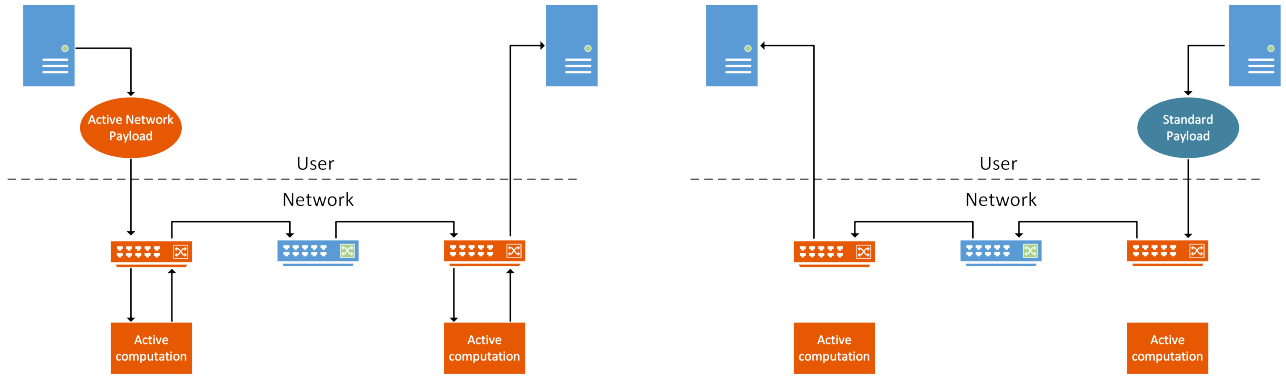
Research into SDN was first motivated by the field of active networks. Active networking provided an ability to embed computation into packets and network devices, allowing for the computation to occur inside the network as a packet traveled through the network [19]. An illustration of the operation of an active network is shown in Figure 1. This system provided a SDN-like interface to programmers and allowed for interesting classes of applications, such as the ability to modify packet headers at different points in a packet's flow or implement common network functions, including firewalls or proxies, inside of the network without the need for extra hardware. However, active networking introduced some fundamental challenges that proved to be difficult to solve.

The largest of these issues was a lack of a single clear motivation for deployment. Although many applications for the technology could be described, none of the them provided a compelling reason for deployment of the system [10]. Without this reason, network operators had no incentive to deploy the extensive hardware upgrades that would have been required to support the system.

2.2 Early SDN

Active networks, despite their limitations, represent a design thats attempts to provide the flexibility that current SDN strategies strive for. Learning from the motivational issues that prevented the adoption of active networks, researchers focused on narrower and more clearly defined problems, which led to a focus on a separation between the control plane and the data plane. This focus was prompted by the increase in traffic volumes as the Internet grew in size, leading to network administrators to search for a new control interface for their networks. Early technologies attempted different methods of creating a separation between the control and data planes [10]. However, many of these technologies proposed the use of standard APIs for control of the data-plane, while leaving the operation of the data-plane essentially unchanged. These designs left little incentive for adoption by hardware vendors [10], as they would have allowed new competitors access to their products.

New technologies were soon proposed that created clean-slate designs for centralized control of networks [11, 7] . Such technologies allowed for entirely new methods for control while still using existing protocols in the data-plane, such as IP, ARP, TCP and others. In addition, these designs allowed for easier deployment as they could be deployed alongside existing traditional networks. In particular, the work proposed



(a) An active network transmitting data with active contents. The data will cause computation to occur in the network based on its contents.

(b) An active network that is transmitting non-active data. The packet is transferred as if the network is not active.

Figure 1: Two examples of active network operation. Red elements represent network elements and data that are active network enabled.

by Ethane included a full-scale deployment of an SDN system that supported end hosts and existing network devices unmodified, which provided clear example of a functioning system. These works proved to be the first attempts at SDN designs, and led to the design of the OpenFlow protocol [10].

2.3 OpenFlow

Early work in SDN research encountered a basic issue of deployment scale, which led to different trade-offs between the level of programmability in the network and the ease of deployment. However, work such as Ethane proved that SDN networks were deployable, and campus networks were oftentimes used to implement new network designs, along with tools like Emulab, PlanetLab and the GENI project[10]. The OpenFlow protocol was designed to be deployed inside of these networks, with a trade-off designed to ease distribution while still providing network programmability.

The protocol allows for a centralized controller to control special OpenFlow-enabled devices using a specific communication protocol, but the messages inside of the data-plane are transmitted using existing protocols (IP, TCP, etc...). Furthermore, existing traditional network devices can be used alongside OpenFlow enabled hardware due to the support of existing network protocols. The OpenFlow controller controls OpenFlow-enabled devices by writing instructions directly to device forwarding tables, which act on information encoded in protocol headers (IP addresses, TCP ports) in order to determine the correct forwarding action. The protocol defines a number of actions that can be performed on a packet, and when a packet is received by an OpenFlow-enabled device, the device checks the flow table, which is populated with rules sent from the controller, for a match for the packet. If any of the defined rules match information in the packet header, then the rule's actions are applied to the packet. Some actions that are defined by the protocol include actions to forward, drop, or modify a packet, and the protocol allows for devices to match rules against the headers of packets for most existing network protocols.

The OpenFlow protocol was initially designed for deployment on campus networks, but as the utility of the protocol was demonstrated, hardware vendor interest increased. In addition, the protocol began to see deployment in datacenters, and along with the promotion of the protocol by its developers, hardware vendors began to create network devices that have hardware support for the protocol. Work on software switches, such as Open vSwitch [1] allowed for researchers to quickly create new applications, and then use vendor hardware to deploy them.

With the data-plane provided by both software and hardware switches, the door was opened for the development of control-plane architecture. There have been many controller architectures proposed, such as NOX, Floodlight, POX, Ryu and others, and all of these are capable of communicating to switches using the OpenFlow protocol. In addition, these architectures do not need to be run on specialized hardware, allowing for commodity hardware to be used to implement controllers.

Despite the combined support for OpenFlow by both the research and industry communities, it should be noted that OpenFlow is not synonymous with SDN. OpenFlow is currently the most popular implementation of SDN, but other implementations could exist, such as private designs from industry entities like Cisco, Microsoft or Google. It should also be noted that OpenFlow is not a perfect solution. Research has identified several security issues in the protocol, which are detailed in Section 4. The protocol itself has also seen difficulties in deployment of new updates to the protocol specification. Only the first protocol specification, version 1.0, is commonly supported by vendors, despite the large amount of community support, but the most recent specification has gone through several revisions [2]. Adoption rate is much slower than other hardware protocols, such as WiFi.

3. SURVEY OF SDN SECURITY

For this survey of current SDN research, I discuss eleven distinct works that have a direct bearing on SDN security. Some of these works are historical, but they serve to illustrate the trend of security research during the development of SDN. Security research in active networks, in particular,

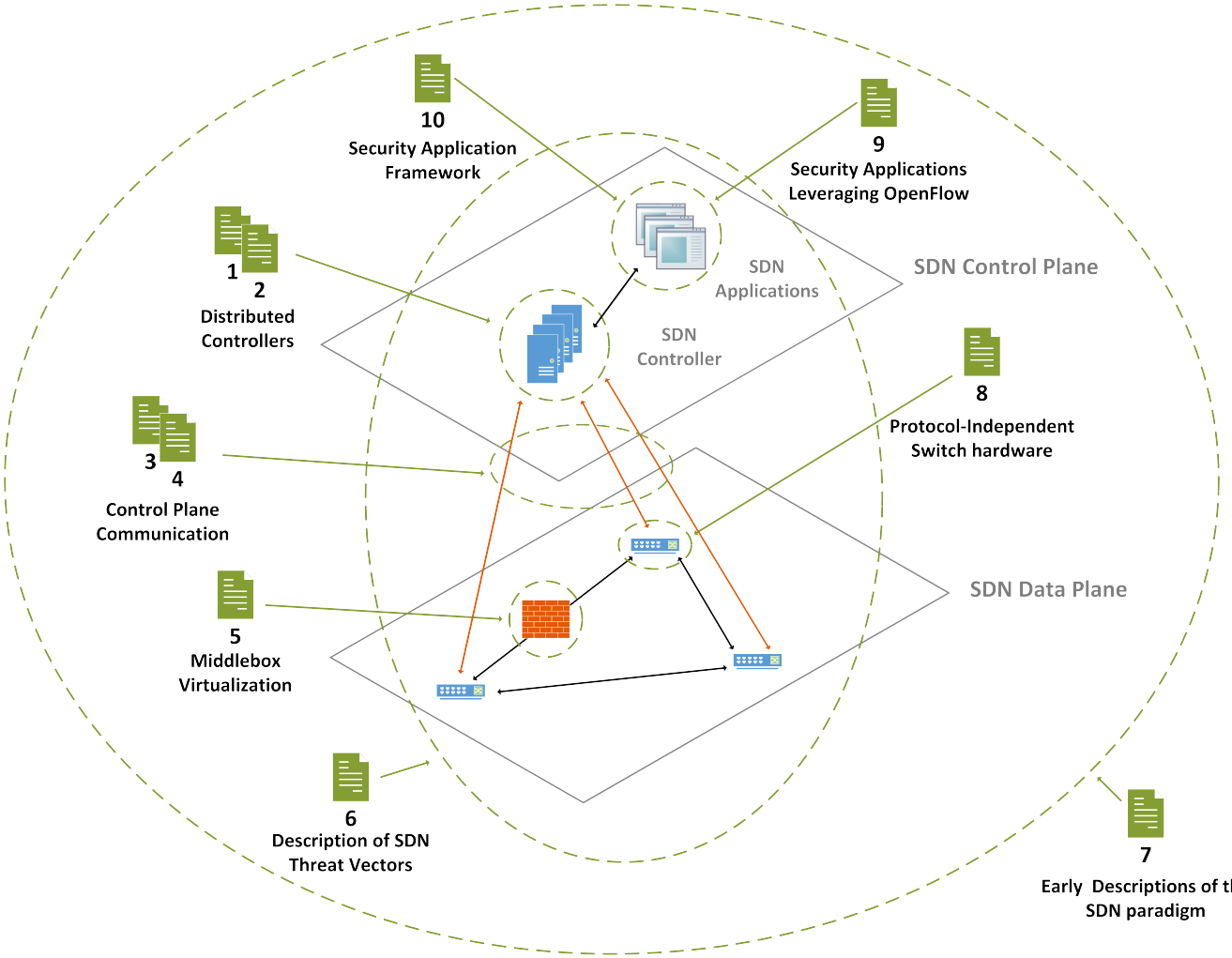


Figure 2: Relation between different papers that address SDN

was not a priority, which may help to explain some of the security issues that have been noted in modern SDN. An illustration of how the papers discussed in this review relate to different SDN elements is shown if Figure 2. Each numbered paper that is defined in the figure is listed below, as well as in the description of each paper in this section. The work presented by Tennenhouse, et. al. is not indicated on this figure, as this figure specifically list SDN technologies based of the design paradigm presented by Greenberg, et. al. and discussed in Section 3.1.2.

Papers indicated in Figure 2

1. *Towards an Elastic and Distributed SDN Controller* [9]
2. *Kandoo : A Framework for Efficient and Scalable Offloading of Control Applications* [20]
3. *DevoFlow: Scaling Flow Management for High-Performance Networks* [8]
4. *AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-Defined Networks* [18]
5. *Enabling Fast, Dynamic Network Processing with clickOS* [14]
6. *Towards Secure and Dependable Software-Defined Networks* [13]
7. *Public Review for A Clean Slate 4D Approach to Network Control and Management* [11]
8. *Programming Protocol-Independent Packet Processors* [6]
9. *Revisiting Traffic Anomaly Detection using Software Defined Networking* [16]
10. *FRESCO: Modular Composable Security Services for Software-Defined Networks* [17]

3.1 Early SDN Research

Software defined networking research can trace its roots to active networking and research into 4D architectures, as presented in [11, 19] and discussed above. These two works, in particular, are representative works of the early research in SDN-related fields. Although security was not the focus

of these two works, several security considerations are presented.

3.1.1 Active Networks

The work presented by Tennenhouse, et. al. is a survey of active network that existed at the time of publication, 1997. In this survey, two approaches to active network research are defined: a *discrete* approach, by using programmable network switches that act to data packets and packets containing code differently; and an *integrated* approach, by encapsulating all data inside of a program packet, which would require network hardware to perform computation on each packet. The paper also presents strategies for interoperability between these two design philosophies, including different programming language choices and use of common primitives and encoding schemes to ensure interoperability. The paper concludes with a list of current research topics in active networking from various institutions. It should be noted that there are many projects listed in this survey, but very little work was done on related projects by these institutions past the year 2000.

In this particular work, the researchers present a single security consideration of active networks for future research. The authors present active networks as a simpler platform to implement per-user authentication in the network, as compared to methods that existed at the time. As active networks allow for computation to be performed in the network, it was a logical extension to propose authentication in the network. However, this work does propose an implementation of such an authentication mechanism. The paper does highlight research performed at the University of Pennsylvania on the SwitchWare project [3], which had a direct focus on security. However, work on the SwitchWare project effectively ceased in 1999.

3.1.2 The 4D Project

The second work, from Greenberg, et. al., is a publication of a review of the original work, "A Clean Slate 4D Approach to Network Control and Management". The original work advocated the creation of networks based on four planes, the decision, dissemination, discovery and data planes. This work follows a trend at the time to create network designs that separate the control-plane from the data-plane, which was a more focused research problem then the architectures proposed by active networks. The separation of the control- and data-planes provided a set of well-defined benefits, included a much-simplified administration interface, which would be beneficial to network operators faced with the ever-increasing size of the Internet.

The four planes presented by the 4D approach perform this separation by placing the control functionality into the decision plane and using the dissemination plane to communicate between the decision and data planes. The discovery plane is used by the decision plane in order to create a network wide view of all connected devices. By providing a network wide view to a separate control plane, the authors present the 4D approach as having better potential for simple implementations of network-wide security policies, as the decision plane has the ability to place security measures wherever is necessary in the network.

However, the 4D system was a theoretical design that was not implemented for the paper. In addition, the authors note several challenges to the architecture, including a number of inherent security issues in the system that do not exist in traditional networks. These security issues, especially that of attacks on the centralized decision plane, are still a topic of SDN security research today.

3.2 Recent SDN Research

Recent research into SDN security follows two specific tracks, that of OpenFlow specific research, and general SDN research. As OpenFlow is the most prominent SDN deployment, there is great interest in mitigating security problems in the protocol, as it is used in many production settings. However, other security research in general SDN strategies allow for progress towards new protocols beyond OpenFlow.

3.2.1 General SDN Security

OpenFlow, despite its popularity, is not the only method of implementing SDNs. There are many limitations of OpenFlow, with the limitations relating to security detailed in Section 4. The work presented in this section lists research that has security applications towards SDNs in general, in contrast to those limited to the OpenFlow specification.

Programming Protocol-Independent Packet Processors. The work presented by Bosshart et. al. advocates the creation of switch hardware that can support any arbitrary protocol using a parser and a series of match-action processors [6]. This work is motivated by the ability of OpenFlow-enabled hardware to match rules to packets based on different header fields in existing protocols, which allows for rules to be to be enacted on any possible permutations of these fields. However, this work is also motivated by the fact that OpenFlow can only be perform match-actions on protocols that are supported by the specification, such TCP or IP, rather than any arbitrary protocol that a developer desires to support.

This work proposes a method to match rules to header fields that are defined by the programmer, which allows the system to support new protocols that have headers that are differently sized then existing protocols. In addition, the system advocates the use of a parser that reads programs that are written in the domain-specific language provided by the authors of the paper, which can be compiled to various hardware platforms. This allows for various different hardware implementations to be supported so long as a compiler can be written for each platform. This ability has several implications for creation of new security measures, which is explored further in Section 5.

ClickOS. The work presented by Martins, et. al. is a method for creating virtualized middleboxes that are capable of operating at line rate [14]. This work is motivated by the Click Modular Router [12], which allows for the creation of varied packet processing datapaths that can implement various network functions. The researchers note that Click can be used to implement middleboxes in hardware, but the Click software is only implemented inside Linux operating

systems, which have a very high system footprint compared to the functionality that is created by a middlebox implemented in Click. The researchers instead use MiniOS, a minimalistic operating system provided by the Xen hypervisor [4], and run the Click software inside of instances of this operating system. By using MiniOS, the researchers are able to create new virtual machine instances very rapidly, with each potentially implementing different middlebox functionality. This system presents software-defined middleboxes and presents several interesting security opportunities for different SDN technologies, as is discussed in Section 5.

3.2.2 OpenFlow Security

As previously stated, the OpenFlow protocol is the deployment of SDN that has received the largest amount of support from the research community and industry. Due to the popularity of the protocol, security issues that were not addressed by the original specification are now priorities of focus, as the protocol is being used in production systems. Kreutz et. al. present in [13] a list of security vulnerabilities that exist in SDNs, especially the OpenFlow standard. The authors of this paper recognize that many of these vulnerabilities are not unique to SDNs, but these vulnerabilities are oftentimes changes and sometimes exacerbated in SDNs. The authors also present several mitigation strategies for these vulnerabilities that the selected research, detailed below, can be seen attempting to implement. The specific threat vectors detailed in[13] are listed in Section 4.

Towards an Elastic Distributed SDN Controller. In this work, Dixit et. al. present a distributed SDN controller that is compatible with OpenFlow [9]. This work makes significant use of OpenFlow specific mechanisms, especially specific OpenFlow switch management functions. The architecture allows for switches to be migrated between different controllers, and for a pool of available controllers to be resized based on the demand seen in the network. This work is differentiated from others in its use of a distributed data store to store state about the network, and its safety properties that are ensured by the switch migration protocol. This work, along with Kandoo (detailed below), and other distributed controllers, can help to mitigate certain classes of attacks, such as DoS attacks on the control plane, as explained in Section 4. However, the intention of this technology, along with other distributed controllers, is to increase fault-tolerance and scalability of the control-plane, rather than to address security issues.

Kandoo. In this work, Yeganeh and Ganjali present another distributed controller compatible with OpenFlow, which differs from other work in its use of a controller hierarchy [20]. The authors present Kandoo as a system of controllers that establish a hierarchical structure of responsibly, with only the controllers at the top of the hierarchy having responsibly for the entire network. Controllers that exist lower in the hierarchy are responsible only for local decisions, and defer any decisions outside of their scope to the next level, just as the data-plane forwards all decisions to the control plane. The authors only describe a low-level hierarchy with a local and global layer, but they claim that the system can be extended to larger hierarchies if necessary. This system is also useful for defenses against certain types of attack on the control-plane, but its attack surface and response are different than those presented by [9]. It should be noted that security was not one of the primary goals of this work.

DevoFlow. In this work, Curtis et. al. detail several shortcomings of the OpenFlow protocol that degrade its performance in high-performance networks, and then present an architecture for a protocol similar to OpenFlow that addresses these shortcomings [8]. The problems noted in OpenFlow by the authors are related to the fact that the control- and data-planes must communicate in order to gather information about flows due to their separation. In smaller-scale networks, this communication is tolerable, but in higher-performance networks, the amount of communication can seriously degrade the performance of the network, which is not just a performance issue, but also a potential security risk. The architecture presented by the authors returns a degree of responsibility to network devices to make local decisions, such as making quick routing decisions, and allows for more efficient collection of statistics in order to reduce communication. In this architecture, the control-plane is only consulted for unknown flows and flows that have been designated for specific traffic engineering goals, such as quality of service (QoS). This is another attempt at reducing the load on the control plane, similar to the previous two works. However, security considerations were made by the authors during the design of the system.

AVANT-GUARD. In this work, Shin et. al. present a system that is directly focused on security vulnerabilities that exist in OpenFlow [18]. The system is motivated by two specific vulnerabilities in the protocol, namely the bottleneck in control-plane and data-plane communication and the slow response rate of the controller to changes in the data-plane. These vulnerabilities are an inherent flaw in the separation of the data- and control-planes, as the controller has to initiate communication to interact with the data-plane, and therefore experiences an unavoidable delay. The researchers propose two mechanisms for the OpenFlow protocol in order to address these issues, which are implemented in the data-plane. The first of these mechanisms is the addition of an ability for network devices to directly respond to TCP connection requests and to filter out illicit connections using SYN cookies [5] and a special TCP handshake procedure. In this method, the network switch acts as a proxy for the destination of a TCP request and filters connection requests until it is determined whether these connections will be accepted by both endpoints. Only when it is determined that a connection will be accepted by both endpoints is the control-plane informed of the connection request. The second mechanism is an extension to the flow table rule specification to allow for rules to be triggered by events, such as a threshold number of packets/second. This mechanism also provides the ability for packet payloads and event notifications to be sent to the controller, which is not supported by current OpenFlow specifications. Further discussion of these mechanisms is presented in Section 4.

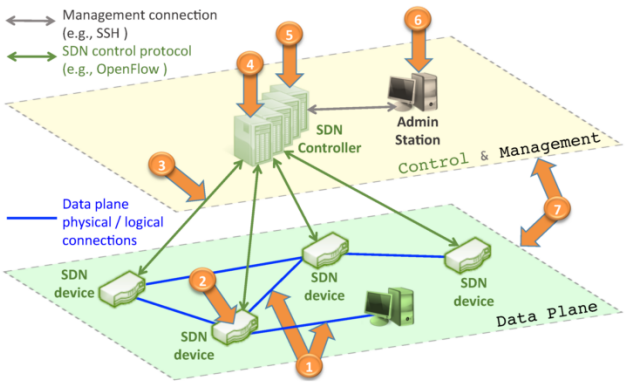


Figure 3: SDN Threat Vectors. Figure originally appeared in work from Kreutz et. al. [13].

FRESCO. In this work, Shin et. al. present a framework specifically for the development of security applications for OpenFlow [18]. The researchers note the useful features of the protocol, especially the visible that is given to the controller, but also note that creating applications that run on top of controllers entails use of various low-level functions, including the direct writing of flow rules to switches. In order to promote the creation of more OpenFlow-enable security applications, the authors created a framework that provides an abstraction of the OpenFlow protocol that allows for applications to focus on security policies, rather than low-level OpenFlow operations. In order to create this abstraction, the system provides a scripting language and interpreter that allows for applications to compose elements in a fashion similar to Click, and provides an API so that new modules can be written in Python to implement new functionality. This API also allows for non-OpenFlow applications to access the FRESCO abstraction layer. The system also implements a *Security Enforcement Kernel* that is used to resolve conflicts between different applications based on user privileges, and to ensure that non-security applications do not override or circumvent security applications. The implications of this framework are discussed further in Section 4.

Revisiting Traffic Anomaly Detection using Software Defined Networking. In this work, Mehdi et. al. present a security application that utilizes OpenFlow to detect anomalies in home and small office networks [16]. This work represents a use of OpenFlow to implement a security solution that exists in traditional networks, one easier to address using SDN rather than previous detailed works that address specific security vulnerabilities introduced by the OpenFlow protocol. The researchers implement four different anomaly detection algorithms, and using several features of the OpenFlow specification, are able to achieve the effectiveness of these algorithms using SDN while still operating at line rates. Further discussion of the OpenFlow features that enable this performance improvement are detailed in Section 4.

4. OPENFLOW SECURITY

Many of the recent works in SDN security research utilize or are concerned with the OpenFlow protocol. Of the research presented here, there are three categories of research that relate to OpenFlow security. The first of these is research that attempts to solve the scalability and fault-tolerance issues that exist in OpenFlow controller design. These issues are not directly motivated by security concerns, but are directly applicable as they improve the durability of the network under load, as may be seen during a DoS attack. The second category is research that directly addresses security vulnerabilities that exist in the OpenFlow specification. The chief of these issues is the communication bottleneck between the data- and control-planes that can be easily be inundated with control traffic in many situations. The third category is research that uses OpenFlow to solve existing security vulnerabilities. Due to the visibility of the network that is proved to the controller, applications are able to utilize the protocol to create network-wide policies that are more effective than what is available in traditional networks.

As previously stated in Section 3.2.2, Kreutz et. al. have outlined a list of vulnerabilities that exist in SDNs [13]. The researchers note that the main benefit of SDNs, that of the separation of the control-plane and the data-plane, creates many of the vulnerabilities that they identify. The researchers also note that there has been little work done to address these vulnerabilities, and as such, they propose several possible solutions to these vulnerabilities. The different threat vectors are identified by the researchers are summarized below, and are depicted graphically in Figure 3.

1. **Falsified traffic flows**: Flows created by faulty or duplicitous devices in the network, that can be used to deny resources from other devices, in either the control plane or the data plane.
2. **Switch vulnerabilities**: Attempts to exploit vulnerabilities in switches in order to compromise these devices. Such attacks can lead to the exploit of other weaknesses in the network.
3. **Control plane communications**: Any attack that can compromise the security of the communication channel between the data plane and the control plane can disrupt or even halt network operations. Other attacks can simple attempt to overwhelm the communication channel in order to prevent the network from functioning.
4. **Controller vulnerabilities**: Similar to attacks on switch vulnerabilities, but much more severe. Once a controller is compromised, an attacker potentially has complete control over the network.
5. **Trust between controllers and applications**: Most controllers do not establish rules of trust for applications, and mechanisms do not exist in order to establish trust. Applications run on controllers must be trusted, as a controller application has the same view of the network as the controller.
6. **Vulnerabilities in administration stations**: Just as the controller must be protected from attacks by

the network, so must the means of controller programming. If this system is compromised, then an attacker can reprogram a controller, rather than attempt to compromise it.

7. **Lack of trusted forensics resources**: OpenFlow does not provide resources to understand many of the problems that may occur in the network and it does not provide any authentication mechanisms to verify the source of the statistics that are provided by the specification.

Items 1, 2, 6 and 7 are not unique to SDN, but these problems manifest in different ways in SDN and OpenFlow. Several of these issues are addressed by some of the work reviewed in this paper.

The work presented in Section 4.1 addresses subclasses of attacks that are related to item 3, where a DoS attack attempts to overload the communication path between OpenFlow switches and controllers. By making the controller more fault-tolerant and distributed, it is significantly more difficult to create sufficient load to prevent the controller from being able to respond to information from the data plane. In addition, this work also helps to mitigate attacks related to item 4, as anomaly detection could be applied to the controller layer in order to detect when a controller is potentially compromised, and a new controller can be tasked with the misbehaving controller's responsibilities.

Work in Section 4.2 is mostly applicable to the first three threat vectors due their focus on two OpenFlow-specific limitations. The only work that is applicable to vulnerability 5 is presented in Section 4.3, and no presented work is applicable to threat vectors 6 and 7. However, these two vectors are not exclusive to SDNs, and there exist several possible existing solutions to these issues [13].

4.1 Performance and Fault Tolerance

By enabling a more fault-tolerant network, the system becomes resistant to brute force attacks such as DoS. By increasing performance, the controller is better able to quickly response to events that occur in the data plane. The work presented by Dixit and Yeganeh create distributed controllers that allow for both increased performance by load balancing over a set controllers, and controller fault-tolerance, as the pool of controllers is capable of recovery from the loss of controllers. In terms of the above defined threat vectors, a distributed controller helps to mitigate the third vector, as control communication can be balanced across the pool of distributed controllers in order to reduce the overall load. A distributed controller can also be adapted to address the fourth threat vector, since an anomaly detection algorithm can be deployed in the controller to detect any misbehaving controller instances.

In terms of the two works that are discussed in this review, both define valid distributed architectures that can solve these issues. However, the architecture presented by Dixit, et. al. provides some very clear safety and liveness guarantees , whereas the architecture presented by Kandoo provides an interesting hierarchical design that allows for the distribution of decision responsibility, allowing lower layers to make local decisions, and higher layers to rule on global policy. Both of these aspects are desirable in an SDN, so a hybrid approach of these two designs should be implemented. However, it should be noted that these two works specifically define distributed OpenFlow controllers, and particularly the liveness and safety guarantees provided by [9] are made possible by OpenFlow mechanisms.

4.2 Vulnerabilities in OpenFlow

The work presented by Curtis and Shin specifically address different limitations of OpenFlow. These two works independently identify the same two limitations in the OpenFlow protocol.

- **Controller communication bottleneck**: Due to the inherent imbalance between the number of network devices in the data-plane and the number of controllers, even in a logically centralized controller, there is the potential for a bottleneck when devices send control messages to the controller. An attacker could attempt to exploit this vulnerability by inducing a large number of control messages to be sent to the controller by sending a large number of packets to the network that do not correspond to existing flow rules. Under normal OpenFlow operations, each of these new messages must be stored by the switch and the controller must be informed of the messages existence with the dispatch of a PACKET_IN message. If enough of these messages are generated in a short time, the communication path to the controller, and possibly the controller itself, could be overloaded, leading to the dropping of some messages that could correspond to legitimate traffic and a general slowing of the network. If this issues persists for long enough, then it is even possible to create a DoS in the data-plane, as switch buffers are filled with traffic that does not match flow rules, which will prevent any new legitimate traffic that does not match a flow rule from using the network.

- **Lack of timely data-plane monitoring functions**: The OpenFlow specification provides several mechanisms for accessing statistics about flows from switches, but these statistics must be requested by the controller. Furthermore, there is no defined method for aggregating these statistics or batching them to reduce link usage. As the request for statistics and the response from switches must traverse the control communication link, the communication link bottleneck issue is worsened and any timely response by the controller to events in the data-plane is limited by the frequency of requests for information and the delay in their delivery.

4.2.1 DevoFlow

The first work that specifically addresses these issues, DevoFlow, proposes the return, or *devolvement*, of some decision functionality from the control plane to the data plane in order to reduce the number of messages that are sent to the controller. The functions that are returned to the data plane allow switches to make local decisions on routing for similar flows, based on rules defined by the controller, and allow for multi-path calculations and quick re-routing in the

face of failures. This allows the controller to essentially "pre-register" flows with the data-plane, and requires a smaller number of flow entries to support multiple routes. In addition, DevoFlow also defines additional statistics options, including sampling, trigger-based reporting and approximation, in order to decrease the amount of statistics information that needs to be reported to the controller and allows for timely updates to be provided to the controller based on events that occur in the data plane.

DevoFlow addresses the third threat vector defined above by decreasing the use of the communication path between the data plane and the controller. However, this issue is not totally mitigated, as an attacker can still initiate a DoS and overwhelm the controller communication link by crafting packets that do not match flow entries. DevoFlow also addresses the first and second threat vectors by allowing the controller to register for event-triggered statistics updates that allow it to perform timely anomaly detection in the data plane.

4.2.2 AVANT-GUARD

The second work that addresses OpenFlow vulnerabilities, AVANT-GUARD, proposes a similar return of functionality to the data plane as is proposed by DevoFlow, but in a more effective manner.

AVANT-GUARD first proposes a method for switches to intercept TCP connection attempts and implement a handshake protocol using SYN cookies [5] before involving the destination host or the controller. This handshake is intended to filter illicit requests that would not be accepted by the destination host or that would not be maintained by the source host. The system only informs the controller of the connection request once the handshake is complete, allowing the controller to establish a flow rule, if required. This method significantly decreases the control-plane traffic that would be seen by an average port scan attack, and also prevents the propagation of control messages for connects that do not match flow rules in terms of TCP packets. As such, this method makes great strides in addressing threat vector three.

The second method proposed by AVANT-GUARD is similar to the statistics options that are introduced by DevoFlow, with several additional features. AVANT-GUARD introduces the concept of event-triggers that can initiate a number of actions to be performed by a device in the data-plane. Such actions can be the activation of a flow rule based on some condition such as a traffic rate threshold, or the delivery of statistics to the controller. The system also allows the controller to register for the delivery of packet payloads in addition to header information, which is not currently supported by OpenFlow. These capabilities allow AVANT-GUARD to address the first and second vulnerabilities defined above in a much more effective manner than is proposed by DevoFlow, as the controller is not only able to register for event-based statistics, but can proactively install event-based flow rules. The controller can also perform deep packet inspection on packet payloads by registering for their delivery, allowing many host-directed attacks to be mitigated.

4.3 Security Applications in OpenFlow

Previous research discussed in terms of OpenFlow security has focused on providing solutions for known vulnerabilities in the protocol, or using protocol-specific mechanisms to protect the network infrastructure. However, little work has been presented on the implementation of known network threat-mitigation strategies, such as intrusion detection and firewalls, which protect end host systems from attack. Previous presented research has defined the target as the network, and the OpenFlow protocol specifically. However, the work presented by FRESCO and Mehdi et. al. represents different attempts to implement know security applications inside OpenFlow.

4.3.1 Revisiting Anomaly Detection using Software Defined Networking

Mehdi, et. al. present an analysis of the effectiveness of four different anomaly detection algorithms that exist for traditional networks[16]. The researchers note that these algorithms, at some level, operate on the number of packets passed through the network per flow. By utilizing standard OpenFlow network operation, the controller is informed of the first packet for each flow. By intercepting this first packet notification, the algorithm can know the associated flow, and then statistics can be pulled from switches to determine the number of associated packets. By controlling what type of flow entries are written, different algorithms can be supported that need to have different levels of flow granularity, e.g. raw number of TCP connections versus connections per TCP port number, etc. By utilizing these OpenFlow features, algorithms that ostentatiously require every packet can instead only be proved with the first packet, and allow all other packets to be processed at line rates.

This application is a prime example of the use of OpenFlow capabilities to implement a network security solution. The researchers also present the OpenFlow implementation as a more efficient implementation than the traditional network implementation, as the algorithms implemented do not need to see every packet, but only the number of packets processed.

4.3.2 FRESCO

The system that is presented by FRESCO is a development framework for the creation of OpenFlow enabled security applications. As detailed in Section 3.2.2, FRESCO provides scripting language to implement a Click-like modular compositional architecture, an API for the definition of new Python modules, and a kernel responsible for the enforcement of rules. In addition, FRESCO also allows for the the use of public-key cryptography in order to sign an verify security modules, which allows for a trust system to be implemented for security applications. This is the only presented research that is able to address the above-defined threat vector five, as this is the only proposed authentication mechanism between the controller and applications. In addition, this work promotes the creation of new security applications, as the modular nature of the system and the abstraction interface provided, hide many of the low-level details of OpenFlow. Application developers are able to focus on defining high-level policies, and the FRESCO system is responsible for pushing those rules to switches.

5. GENERAL SDN SECURITY

As described in Section 3.2.1, two works presented in this review are applicable to SDN security as a whole: ClickOS [14] and Programming Protocol-Independent Packet Processors [6]. The first of the works, as is previously described, is a system that allows for the creation of modular network functions, such as the functions that are implemented in middleboxes (firewalls, proxies, intrusion detection systems, etc.), and encapsulating them in an extremely lightweight virtual machine. These virtual machines can be quickly instantiated, and therefore can potentially be created and destroyed based on demand. The second work presents an architecture for a more general SDN implementation than OpenFlow, allowing for rules to be matched on programmer-defined header fields, and allows for a compilations process to support varied hardware implementations. Both of these technologies have direct implications towards SDN security.

5.1 ClickOS

The implications of ClickOS are applicable even to OpenFlow enabled networks, though the focus of the system is on middlebox functions, which generally address host-based security issues. However, due to the speed with which the middlebox VMs can be instantiated, it is possible for new on-demand middlebox security applications to be developed. As such, potential load-balancing of network traffic can be performed that allows for greater flexibility for network operators in how they implement network-wide security policies.

5.2 Programmable Protocol-Independent Packet Processors

The implications of this work are more related to how programmers attempt to detect and classify attacks. In a system where programmers are able to define the header fields that they wish to match to flow entries, it provides greater flexibility for programmers to both detect and control how attack traffic passes through the network. For example, any TCP option can be used to match on, which allows for programmers to potentially tag packets and have them redirected based on these tags, or header fields can be defined for higher level protocols than are supported by OpenFlow, such as the application layer (HTTP, FTP, etc.), which allow the programmer to have an even higher level of granularity that can be applied to traffic dissection.

6. FUTURE SECURITY RESEARCH

After reviewing the different works that are summarized in Section 3, I was able to identify several trends that exist in current research. The importance of OpenFlow to the research community must be noted, as the protocol is focused upon by a majority of the recent security research. This is logical due to the industry adoption of the protocol, which motivates the research community to attempt to improve the specification and utilize it to implement security solutions. However, other research proposes technology that provide greater flexibility than what is proved by OpenFlow. Such technologies provide powerful tools to network operators and application developers, but unfortunately, some of these are difficult to implement as they entail changes to hardware. The different trends that I identified are detailed below.

1. After examining the origins of active networks and SDN, it can be seen that each of these technologies attempts to solve some specific issues from previous technology, e.g. the Internet. Furthermore, new SDN designs attempt to resolve specific issues that are observed in OpenFlow. However, new designs face the issue of deployment, as can be seen by the issues encountered by active networks and new SDN designs, and even new OpenFlow specifications. New designs also seem to introduce new vulnerabilities that are directly related to the new features that separate them from previous work. This can be seen clearly in the control-plane communication bottleneck that exists in SDN. This trend shows that even though many new solutions to protocol issues are presented, few of them are adopted due to the difficulty of achieving industry support and the existence of vulnerabilities that are created by the technology's new features.

2. Continuing from the previous trend, the new SDN designs that are introduced present more flexible capabilities than what is provided by OpenFlow, such as the support of arbitrary network protocols. Due to the static set of features that are provided by OpenFlow, the motivations for these applications are clear, though there is still the issue of deployment that is discussed previously. Most of these new designs require changes to data-plane devices. Such changes can be implemented in software implementations, but any widespread hardware deployments are more difficult to achieve.

3. Due to the popularity of OpenFlow, research is also being performed in order to optimize different aspects of the specification and address its shortcomings. These technologies are oftentimes easier to deploy, as many of them can be applied at the controller layer and take advantage of existing OpenFlow capabilities. These technologies are also more likely to be deployed, as industry entities that use the protocol are more invested in implementing improvements.

Based on these trends, I predict several future research movements. These trends are based on the fact that OpenFlow is currently in use in datacenter environments, which demand a higher degree of scalability of the protocol than it was designed for. In addition, there is a continued push to achieve the network programmability that motivated OpenFlow and SDNs in the first place. As such, there is a greater motivation to provide support for developers to easily create applications. Current application development is difficult, as applications are tied to a specific controller architecture and are not portable. Also, new SDN alternatives are difficult to test, especially at scale, due to the difficulty in achieving hardware support. Due to these two issues, future research will have a focus on development platforms and frameworks.

In terms of SDN security, easier and more portable application development allows for easier integration of security and non-security applications that can be deployed on any controller platform. New development platforms allow for the testing, and potential deployment, of SDN designs that incorporate new security features that are not present in

OpenFlow. Specific predictions for research in this area is detailed below.

- **Tools for adoption and testing**: Virtual switches, e.g. Open vSwitch, are often used to test new SDN protocols, but Open vSwitch and other switches are limited by their existence as software components. I believe research into increasing performance of software switches is essential, and work has already been proposed that beats the performance of Open vSwitch. However, a lower-level virtualization solution, such as is provided by ClickOS, would be beneficial. Also, better solution would be to use FPGA technologies, however little development has been performed using FPGAs, perhaps due to time and monetary constraints. However, FPGAs provide the flexibility that is needed to test new protocols.
- **OpenFlow application frameworks**: FRESCO provides a framework for the development of security applications. A natural next step is to extend this system to be used with any OpenFlow application, and to extend it to a general interface for all controllers. Such a framework would allow for a creation of a trust system between applications and controllers, similar to what is proposed by FRESCO.

In addition to tools for application development and deployment, research will also continue to address OpenFlow limitations and present new optimizations. This is motivated by OpenFlow's deployment in datacenters, which puts a higher responsibility on OpenFlow and makes securing the protocol more important. Future research will therefore attempt to implement more pragmatic features in the protocol that are suited for the network environments that are seen in datacenters. These features will serve the dual role of increasing the protocol's performance and making the protocol more resilient to failure and attack. In addition, further work will be performed to address specific security vulnerabilities in OpenFlow. I have already presented work that addresses some of these issues, and work on these vulnerabilities will continue, but some of these vulnerabilities need to be addressed by future work.

- **Systems for trust in the network**: Previous work has called for methods to assign trust to different aspects of SDNs, especially OpenFlow [13]. Public-key cryptography can be used to implement trust, if implemented with care. However, designs must include a system for establishing trust. This can be performed by human administrators or by merit-building trust systems. Such a system could be combined with a development framework in order to ease development, such as a system that leverages capabilities that are presented by FRESCO.
- **Further OpenFlow devolution**: As OpenFlow is currently deployed in datacenters and further adoption of OpenFlow specifications has stagnated [2], greater designs such as DevoFlow will be proposed that give more decision power to data-plane devices. Combined with software switches in hypervisors, these designs could be implemented. Other powers that could be returned to the data-plane include forwarding lookups and flow entry optimizations. Controllers would be responsible for populating devices with the correct information and pushing changes, but devices would be able to make routing decisions or combine flow entries locally. This trend helps to reduce the impact of the data-plane to control-plane communication bottleneck.
- **Optimization of OpenFlow flow tables**: Many issues in OpenFlow scalability are related to the number of flow entries created in flow tables. A step to resolve this issue should be taken by refactoring individual flows into general flows using the wildcard rules provided by OpenFlow. Previous work has hinted at this possibility [20], but it was not fully explored. Such optimization would decrease the impact of brute-force attacks on the network that attempt to generate large numbers of flow entries in the data-plane.

7. CONCLUSION

In this paper, I presented a review of current research into SDN security along with a prediction for the paths that future research will take. I have identified several categories of research that are split between the most popular SDN deployment, OpenFlow, and general SDN research. This split is natural due to the wide community and industry support for OpenFlow, which motivates research to help improve the protocol as it is used in production environments.

This research marks the first concerted attempts at SDN security, which until recently has been somewhat lacking, even in OpenFlow. As OpenFlow was not originally designed for security and it supports existing protocols, new security research does not seem necessary. However, the benefits provided by SDN also introduce new security challenges. Research discussed here represents valid solutions to some of these security issues, but further work must be done in order to satisfactorily secure these vulnerabilities.

8. REFERENCES

[1] Open vSwitch. http://openvswitch.org/.

[2] Openflow - enabling innovation in your network. http://archive.openflow.org/. Open Networking Foundation.

[3] The SwitchWare Project. http://www.cis.upenn.edu/ switchware/l. University of Pennsylvania.

[4] The xen project. http://www.xenproject.org/.

[5] D. J. Bernstein. SYN Cookies. http://cr.yp.to/syncookies.html.

[6] P. Bosshart, D. Daly, and M. Izzard. Programming Protocol-Independent Packet Processors. *arXiv preprint arXiv:*, pages 0–6, 2013.

[7] M. Casado, M. Freedman, and J. Pettit. Ethane: Taking control of the enterprise. *ACM SIGCOMM*, 2007.

[8] A. Curtis and J. Mogul. DevoFlow: scaling flow management for high-performance networks. *ACM SIGCOMM*, 2011.

[9] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella. Towards an elastic distributed SDN

controller. *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*, page 7, 2013.

[10] N. Feamster, J. Rexford, and E. Zegura. The Road to SDN. *ACM Queue*, 11(12):20–40, Dec. 2013.

[11] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. Public Review for A Clean Slate 4D Approach to Network Control and Management. *ACM SIGCOMM*, 35(5):41–54, 2005.

[12] E. Kohler, R. Morris, and B. Chen. The Click modular router. *ACM Symposium on Operating Systems Principles*, 34(December):217–231, 1999.

[13] D. Kreutz, F. M. Ramos, and P. Verissimo. Towards secure and dependable software-defined networks. *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*, page 55, 2013.

[14] J. Martins, M. Ahmed, C. Raiciu, and F. Huici. Enabling fast, dynamic network processing with clickOS. *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*, page 67, 2013.

[15] N. McKeown and T. Anderson. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM*, 2008.

[16] S. A. Mehdi, J. Khalid, and S. A. Khayam. Revisiting Traffic Anomaly Detection using Software Defined Networking. *Recent Advances in Intrusion Detection (RAID)*, pages 1–20, 2011.

[17] S. Shin, P. Porras, V. Yegneswaran, and M. Fong. Fresco: Modular composable security services for software-defined networks. *Network and Distributed Systems Security Symposium*, 2(1), 2013.

[18] S. Shin, V. Yegneswaran, P. Porras, and G. Gu. AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks. *Proceedings of the 2013 ACM Conference on Computer and Communications Security*, 2013.

[19] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, and G. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, Jan. 1997.

[20] S. H. Yeganeh and Y. Ganjali. Kandoo : A Framework for Efficient and Scalable Offloading of Control Applications. *Proceedings of the first workshop on Hot topics in software defined networks HotSDN' 12*, pages 19–24, 2012.