Comparative Analysis of Control Plane Security of SDN and Conventional Networks

AbdelRahman Abdou, Paul C. van Oorschot, and Tao Wan

***Abstract*—Software defined networking implements the network control plane in an external entity, rather than in each individual device as in conventional networks. This architectural difference implies a different design for control functions necessary for essential network properties, e.g., loop prevention and link redundancy. We explore how such differences redefine the security weaknesses in the SDN control plane and provide a framework for comparative analysis which focuses on essential network properties required by typical production networks. This enables analysis of how these properties are delivered by the control planes of SDN and conventional networks, and to compare security threats and mitigations. Despite the architectural difference, we find similar, but not identical, exposures in control plane security if both network paradigms provide the same network properties and are analyzed under the same threat model. However, defenses vary; SDN cannot depend on edge based filtering to protect its control plane, while this is arguably the primary defense in conventional networks. Our concrete security analysis suggests that a distributed SDN architecture that supports fault tolerance and consistency checks is important for SDN control plane security. Our analysis methodology may be of independent interest for future security analysis of SDN and conventional networks.**

***Index Terms*—Network security, SDN security, control plane security, OpenFlow security.**

I. Introduction

SOFTWARE-DEFINED Networking is a network architecture in which the control plane is separated from each individual network device and instead implemented in an external software entity. The external entity has complete knowledge of the topology of a network under its control, and programs the forwarding tables of each individual device in the network. In contrast, *conventional networks* (CNs) have the control plane, *i.e.,* network control functions such as routing protocol implementations (*e.g.,* Open Shortest Path First (OSPF) [1]), running inside each network device to learn forwarding tables in a distributed fashion. SDN architectures have two distinguishing properties of direct interest herein [2].

1) *Control and data plane separation:* Removing the control plane from network devices and implementing it in an external SDN controller significantly reduces the complexity of network devices, making them simpler and cheaper than CN devices whose distributed control plane functionality is implemented across millions of lines of code, and defined across hundreds of RFCs.
2) *Network programmability:* An SDN controller, with complete knowledge of a network's topology, controls a multitude of network devices within its administrative domain. By providing application programming interfaces (APIs), SDN makes it possible to develop networking applications, *e.g.,* traffic engineering [3], thus enabling network innovation. In contrast, CN devices are proprietary and closed, making it hard or impossible to develop innovative network applications.

The concept of SDN has evolved since the term was originally coined in 2009 [4]. Network devices in practice can be pure SDN, pure CN, or hybrid. A pure SDN device implements no control function and is fully controlled by an external SDN controller. A CN device implements all of its own control functions and is not controlled by any SDN controller. A hybrid SDN device both implements control functions, and can be controlled by an SDN controller. Accordingly, a network can be one of the three types. A pure SDN network consists of at least one SDN controller and network devices all of which are fully controlled by the controller. A CN consists of devices all of which implement and run their own control functions with no external controlling entity. A hybrid network consists of hybrid devices and at least one SDN controller.

In academic work, "SDN" often implies a pure SDN network, such as an OpenFlow network, and many academic SDN security research papers (*e.g.,* [5]) focus primarily on the security of OpenFlow networks. SDN controllers originating from academic work, such as FloodLight and NOX, also primarily support OpenFlow and control OpenFlow switches which implement no control functionality (*i.e.,* are pure SDN, rather than hybrid).

In contrast, SDN in the industry commonly refers to hybrid networks consisting primarily of CN devices, augmented with open interfaces to allow external control by an SDN controller. For example Broadcom, a leading provider of switch chips, published OpenFlow Data Plane Abstraction (OF-DPA) software [6] to allow switches based on Broadcom

Manuscript received November 7, 2017; revised February 27, 2018 and April 27, 2018; accepted May 17, 2018. Date of publication May 22, 2018; date of current version November 19, 2018. This work was supported by the Natural Sciences and Engineering Research Council of Canada through Canada Research Chair Grant and the Discovery Grant. *(Corresponding author: AbdelRahman Abdou.)*

A. Abdou is with the School of Computer Science, Carleton University, Ottawa, ON K1S 5B6, Canada, and also with the Institute of Information Security, ETH Zurich, 8092 Zürich, Switzerland (e-mail: abdou@scs.carleton.ca).

P. C. van Oorschot is with the School of Computer Science, Carleton University, Ottawa, ON K1S 5B6, Canada (e-mail: paulv@scs.carleton.ca).

T. Wan is with the Cyber Security and Privacy Protection Laboratory, Huawei Canada, Ottawa, ON K2K 3J1, Canada (e-mail: tao.wan@huawei.com).

Digital Object Identifier 10.1109/COMST.2018.2839348

chips to be controlled by OpenFlow. Note those CN devices, although often claimed to support OpenFlow and which can be controlled using the OpenFlow protocol, do not actually implement OpenFlow tables and are not true OpenFlow switches. Rather, they use conventional tables such as L3 tables and Access Control Lists (ACLs) to emulate the behavior of OpenFlow tables, which allows packets to be processed beyond destination addresses. As another example, OpenDaylight [7] and ONOS [8], two leading open source SDN controllers, can control not only OpenFlow switches but also conventional devices, *e.g.,* using NETCONF [9]. It is clear that industrial network practitioners focus more on network programmability than on the separation of control and data planes. We refrain from speculating on which type of SDN is better, or is the future.

We study and compare the control plane security of a pure SDN (hereafter referred to as SDN) and a CN. While hybrid networks are more popular in the field, there is no clear consensus on how to best divide control functions locally inside a device and externally into a controller. Further, by studying the security of both SDN and CNs, we hope that security threats identified in each can be selectively applied to a given hybrid network when its local and external controls are well defined. This may also apply for stateful SDNs, which is another approach to allow some intelligence in an SDN switches (see Section IX-B5 for details).

Motivation: Research on the security of SDN and CNs is in two distinct states. On one hand, the security of CNs has received less academic attention but is well understood by network security practitioners. Aside from the area of routing (*e.g.,* BGP security [10]), there are relatively few academic papers on the control plane security of a CN. However, security threats are well understood by equipment vendors and many security mitigations are built into CN products (*e.g.,* switches, routers). In contrast, SDN security has received considerable academic attention (*e.g.,* [11]–[13]), but its progress is considered slow (at best) by industrial measures. For example, neither of the two leading open source SDN controllers, OpenDaylight and ONOS, has implemented significant security mitigation.

These different states of SDN and CN security research have attracted little attention. We observe that many papers on SDN security assume a simple network, ignoring practical properties such as redundancy and scalability essential to realistic networks—thus excluding security threats faced by important network control functionality. Further, security threats identified for SDN are not properly compared with those in CN. For example, previous literature [11] positions Host Location Hijacking attacks as a new attack in SDN, but inaccurately compares these with ARP cache poisoning attacks in CN, whereas Section VII-A herein shows them to be comparable to MAC table poisoning. Such misunderstandings contribute to why considerable academic research on SDN security have little impact on SDN in industry, highlighting the value of systematizing literature relating SDN and CN security, while focusing on practical issues.

Objective: We aim to address this gap by a comparative security assessment of conventional and SDN networks. Rather than a security analysis of all aspects, we focus on control plane security, since (1) it is in their control plane architecture that CNs and SDN differ primarily, and (2) attacks against control plane aim to affect data plane functionalities.

We provide a framework consisting of essential network properties required by production networks. Using this, we study how those properties are achieved by SDN and CN respectively, and analyze the security attacks and mitigations accordingly. Our finding is that the security threats faced by SDN and CN are comparable in an apples-to-apples comparison, *i.e.,* if they are tasked to provide the same network properties under the same threat model (despite the architectural differences between them). However, defenses vary in that filtering in the network edge is effective in CN, but less so in SDN. Further, consistency checks, which are required by both networks to defeat inside attacks, can be implemented inside each CN device, but require a highly modularized SDN software architecture to facilitate implementation there. Our finding is supported by detailed security analysis. Our framework and comparative methodology may also be of independent interest, to guide future SDN security analysis in both academia and by practitioners.

Contributions: To summarize, this paper:

- Identifies five control functions required by a realistic production network to accomplish essential network services.
- Analyzes threats and defense mechanisms pertaining to these five functions when implemented by L2 networks, L3 networks, and SDNs.
- Provides a new evaluation framework to objectively compare the security of both network paradigms, using two threat models defining the attacker's position in the network.

The sequel is organized as follows. Section II provides background information on CN and SDN architecture. Section III outlines fundamental network properties required by typical production networks, as well as the threat model used for our analysis. Sections IV and V analyze the security threats of the control plane of conventional Layer-2 (L2) and Layer-3 (L3) networks respectively. Section VI analyzes security threats in SDN. We compare the threats and mitigations of SDN with CN in Section VII, and provide insights based on this analysis in Section VIII. Section IX reviews related work. Section XI concludes.

II. Background

Here we provide background on CN and SDN for consistent terminology and later reference. Networking experts may advance to Section III.

A. Conventional Networks

A CN can be L2 or L3. A network consisting of only L2 switches, as its intermediate systems, is called an L2 network. Two (or more) L2 networks can be connected, *e.g.,* using an L3 router. A network of L3 routers is called an L3 network. Other than using different types of destination addresses for forwarding, L2 and L3 networks differ mainly in two aspects:

1) They use different mechanisms in constructing their forwarding tables. L2 devices learn their forwarding

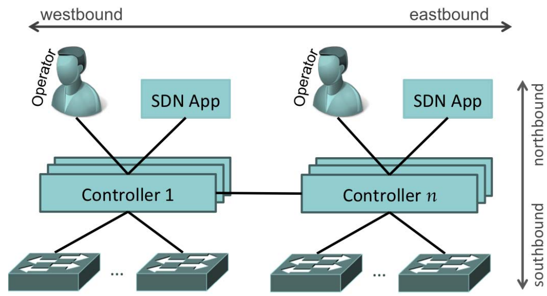


Fig. 1. Generic SDN architecture.

tables (*i.e.,* MAC tables) from the data plane. L3 routers build routing tables from the control plane using routing protocols. Note: MAC tables map MAC addresses to switch ports, not to be confused with ARP tables which map IP addresses to MAC addresses.

2) They handle unknown packets differently. An unknown packet is a packet without any corresponding forwarding rules. An L2 device floods an unknown packet to all ports except the receiving one to learn the forwarding rule, while an L3 router drops an unknown packet (and may also notify the packet source, *e.g.,* using ICMP).

Due to these differences, L2 and L3 networks face different sets of security threats. Thus, we divide CN into L2 and L3, and discuss separately in Sections IV and V.

B. Software Defined Networking

SDN involves one or more SDN controllers, each controlling a number of network elements within its domain via standard protocols (such as OpenFlow). Each controller may run in multiple instances, each further managing a subset of network elements and backing-up other instances to provide both scalability and high availability. SDN controller instances also communicate with each other within the same domain, or may be federated with controllers in other domains, *e.g.,* to form a complete view of the network (see Fig. 1). Further, there may be a hierarchy of SDN controllers for scalability or multiple layer control. The separation of the control plane from the data plane has enabled numerous novel network applications and usages, *e.g.,* Software-Defined Optical Networks (SDONs) [14] and SDN-based intrusion prevention [15].

Scope of Our Analysis: An SDN controller is an entity that does not exist in a CN, thus its security requires special attention. As noted earlier, we focus on control plane security herein. Security analysis of the mechanisms implementing fundamental network properties (see Section III) has not received much attention from the SDN community (see Hong *et al.* [11] for security analysis of forwarding mechanisms in SDN). This motivates us to consider it and herein give a framework for directly comparing control plane security issues with those facing CNs. We see analysis of control plane security as an important step contributing to a broad security analysis of SDN, which should include all SDN components (see Fig. 1).

III. A Framework for Comparative Analysis

Our framework consists of a set of five *network properties*, *i.e.,* functional requirements for typical services made available by production networks, and two threat models (see below) to be applied to both SDN and CNs.

A. Network Properties

Production networks must provide properties, such as loop free forwarding, to allow entities attached to the network to communicate. As outlined by ISO/IEC Standard 7498-1 [16], such properties can be provided by various layers of the protocol stack, hence the modularized layering architecture in CNs. In contrast in SDNs, the responsibility falls primarily on the controller to configure the network to provide such properties. We outline five primary such properties that, we argue, are among the most critical to allow production networks to operate properly in practice. While these are not specifically related to security, they are important in security analysis as each may require its own control functions and introduce unique security threats. Since multiple properties may be provided by a common control protocol, each property does not necessarily introduce new security threats.

A. *Basic Forwarding:* A network consisting of a single switch must establish forwarding information to allow attached entities to communicate. For example, a simple network consisting of one switch must allow hosts connecting via the switch to communicate.

B. *Loop Free Forwarding:* A network consisting of multiple devices and links which form physical loops must ensure there is no forwarding loops among network device forwarding tables.

C. *Link Redundancy:* If there are multiple links between a pair of network devices, the network topology should remain unchanged in the event that one or several of such links go down as long as there is one functioning link between the pair. Further, it should be possible to use all links to transmit data (for higher throughput), instead of only one.

D. *Device Redundancy:* This property is often referred to as *high availability*. A network consisting of two or more devices should remain fully available in the event of the failure of any single device.

E. *Scalability:* As a network grows and becomes large, it should remain functional and manageable. Network design should allow growth without significant management overhead.

Note that in this framework, we separate *Basic Forwarding* and *Loop Free Forwarding* because the protocols implementing them are different, the threats targeting each of their protocols are different, and defenses to harden each are different as well.

To demonstrate the importance of these properties, consider as an example a large Internet Service Provider (ISP) network. The first two properties are essential for its basic operation. The ISP aims to increase its network efficiency and minimize the down time for competitive advantage. Thus, the ISP needs to leverage protocols that utilize all the physical connections between switches (third property), and enable the network to continue operation if routers or switches fail (fourth property). This should be auto-configured, otherwise

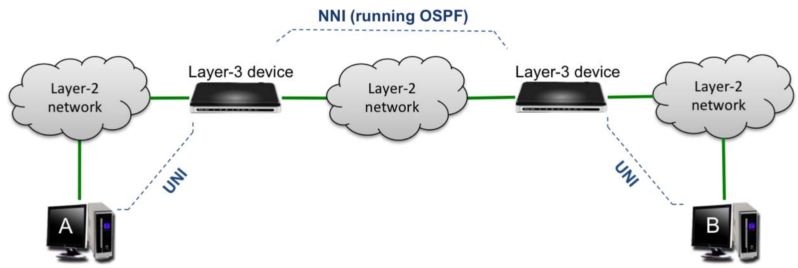


Fig. 2. L3 Network running OSPF in Network-to-Network Interfaces (NNIs).

network administrators must manually reconfigure the network to avoid the faulty device. Finally, the ISP is interested to allow the network to operate smoothly as clients and their traffic increase dramatically, which requires the ISP to run protocols and mechanisms implementing the fifth property.

Besides these properties, there are several others that a network may provide, *e.g.,* multicast routing or Quality of Service (QoS). A carrier network may also need to support virtual private networks (VPN) [17] to its customers, *e.g.,* using MPLS. The importance of these properties vary by the applications using the network, whereas the five properties we considered above in our framework are essentially fundamental as they reflect any network's ability to operate efficiently, be highly available (upon link or device failure, see [18]), and scale properly with increased loads and connected end-points [19].

B. Threat Models

ITU-T X.800 [20] indicated five fundamental security services to be provided by different layers of CNs: authenticity, confidentiality, integrity, access control and non-repudiation. Using our framework, we identify how an adversary can exploit the lack of these services to mount attacks. However, the network position from which the adversary mounts these attacks can change its ability to succeed. For example, if an adversary only has access to the network from its edge (*i.e.,* compromised an end host), it will not be able to easily mount a Switch Blackhole attack [13] (where traffic flows are routed through a dead-end path).

A consistent threat model is thus required to objectively compare security threats between CNs and SDN. CNs often assume that network devices (*e.g.,* switches and routers) are trusted but entities attached to the network are not. We call this the END-HOST threat model. Based on this model, CNs often take two defensive approaches in practice:

1) Run control protocols only in ports facing other network devices, *i.e.,* Network-to-Network Interfaces (NNIs), but not on ports facing user equipment, *i.e.,* User-to-Network Interfaces (UNIs). For example, Fig. 2 represents a L3 network using OSPF to automatically learn network topology and update forwarding tables; here OSPF will run in NNIs but not UNIs (the ports connecting A or B).
2) Defense mechanisms are usually deployed in UNIs to prevent attacks against the network from user equipment. For example, assume VLANs are used in Fig. 2. VLAN tags in the frames from user equipment (*e.g.,* A, B) are untrusted, and thus will be stripped upon arriving at a switch port. However, VLAN tags are not stripped by NNIs, assuming they are added by trusted switch ports.

While the END-HOST threat model has its merits, the assumption that network devices are trusted may be too strong, especially as virtualization [21] is becoming increasingly popular, because the boundary of a network can expand from within conventional proprietary hardware-based devices into user-lands, where virtual network devices run alongside user applications inside common commodity servers.

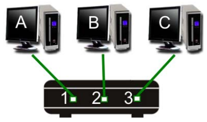
Thus we also consider the ALL-ELEMENT threat model, where all network elements are assumed vulnerable. That is, neither SDN controllers nor network devices are considered trusted. As such, attacks could be from end hosts, as well as network devices and SDN controllers. We use these two threat models in Section VII to compare the attacks and defenses of SDN and CN.

IV. L2 NETWORKS

We now discuss how L2 networks satisfy the network properties of our framework, and analyze security threats and mitigations associated with each property. We focus on Ethernet networks, as they are among the most widely used L2 technologies in practice. This analysis will then be used to objectively compare security threats stemming from the protocols satisfying these properties in L2 networks with their counterpart in SDNs (see Sections VII and VIII for comparisons and insights).

A. Basic Forwarding

To provide network connectivity, an L2 device uses MAC learning to build its MAC table, which maps switch ports to MAC addresses, and possibly other information such as VLAN tags. When a switch receives a frame from one of its physical ports, it adds a new (or updates an existing) entry in the MAC table mapping the frame's source MAC address to the receiving port. Multiple MAC addresses can be associated with a single port. To forward a frame, a switch looks up its destination MAC address in the table, and forwards the frame through the corresponding port. If no entry is found, the switch floods the frame to all ports except the receiving one. The intended destination, upon receiving the flooded frame, sends a response frame enabling the switch to learn the mapping between receiving port and responding source MAC address.



Port #	MAC
1	A
2	B
3	C

Fig. 3. MAC table for switch with three devices connected.

The missing mapping entry is added into the table for future use. Figure 3 illustrates a MAC table for a single switch.

1) Attacks: An L2 MAC table is learned from data plane (including end-user) packets. Thus, it is subject to *MAC attacks* [22]. A malicious host can send a packet with a falsified source MAC address to poison a switch's MAC table. Two known attack strategies are as follows. In *MAC spoofing* [23], an attacker sends frames with spoofed source MAC addresses matching those of target (victim) hosts, thereby hijacking traffic destined to those victims. If a victim is actively sending packets to switches, the poisoned MAC table will alternate between correct and falsified states. A more effective attack, *MAC Flooding*, sends a large number of garbage frames with randomly generated source and destination MAC addresses to fill up the MAC table. Once the table is full, legitimate frames will not match any forwarding entry, resulting in flooding of frames to switch ports including those connecting to the attacker who can thus eavesdrop or even hijack virtually all traffic. Any device, including end-user devices (outsiders) and network devices (insiders), can similarly manipulate a MAC table.

Note that this differs from ARP spoofing (Section V-A1). MAC attacks poison the MAC table of a switch using any packet with a spoofed source MAC address. ARP spoofing poisons the ARP table of a host or router using only ARP-related packets (*e.g.,* ARP response or gratuitous ARP [24]).

Other attacks have also been previously studied. For example, wormhole attacks have received considerable attention in the academic literature (*e.g.,* [25] and [26]), but appear lower on the list of overall concerns than other security problems which we give more attention to herein.

2) Defenses: MAC attacks can be mitigated by preventing untrusted devices such as hosts from sending packets with spoofed MAC addresses. One such mitigation mechanism, *port security* [27], allows a switch to bind a port to one or several MAC addresses (*MAC binding*). Port security usually also limits the number of MAC addresses to be associated with a switch port (*MAC limiting*). MAC binding can prevent MAC spoofing, but is static and typically requires manual configuration—possibly introducing configuration overhead and misconfigurations. MAC limiting appears more practical as it can mitigate MAC flooding attacks and requires only simple configuration, but alone, does not prevent MAC spoofing.

B. Loop Free Forwarding

In L2 networks with multiple switches and looping links (see Fig. 4), forwarding loops can occur when unknown L2 frames are flooded and no Ethernet frame TTL field limits how many times a frame is forwarded. Thus for loop prevention, an L2 control protocol like the Spanning Tree Protocol (STP) [28] is needed.

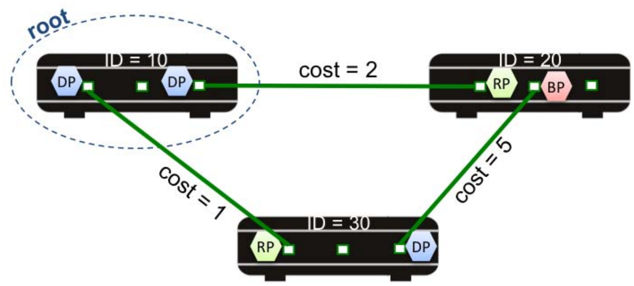


Fig. 4. Operation of the spanning tree protocol.

In STP, switches exchange Bridge Protocol Data Units (BPDUs) carrying information about switch identifiers and path costs, and accordingly compute a spanning tree. A root switch is first elected, typically that with the smallest identifier. Each non-root switch then determines the *root port* as the port with least-cost path leading to the root switch. Similarly, for each link in the network, the end port closer to the least-cost path is called the *designated port*. All remaining ports are called *blocked ports*. A spanning tree then consists of all the network switches (one as root) and some network links. The links not in the spanning tree are still used to exchange control plane traffic (*e.g.,* BPDU) but only in one direction to be loop-free.

1) Attacks: STP uses BPDU, with a multicast destination MAC address, to exchange topology information to elect a root bridge and to establish a spanning tree, assuming all BPDUs are trustworthy. Due to the lack of security protection (*e.g.,* no default robust authentication [29]), STP is subject to BPDU spoofing attacks [23], as well as *BDPU tampering* and *BDPU flooding*. For example, an attacker could send a spoofed BPDU packet with a low priority and small MAC address to result in the lowest bridge identifier among all switches, thus winning the root bridge election. Being the root bridge, the attacker receives virtually all network traffic within the STP domain. BDPU tampering could lead to the calculation of incorrect network topology. BDPU flooding could force switches to continuously re-calculate topology, resulting in service disruption.

2) Defenses: An administrator may intervene in root placement, *e.g.,* manually specifying the location of the root switch, thus eliminating the need for dynamic election. Cisco's *root guard* command [30] facilitates this. Likewise, *BPDU filter* prevents a host from participating in STP by filtering BPDUs in NNIs (see Fig. 2). As such, if a host sends a BPDU, the receiving switch discards it.

C. Link Redundancy

While STP can prevent forwarding loops, it uses a single link between a pair of switches even when redundant links exist, resulting in underutilized network bandwidth or even packet loss in the event of link failures. To improve bandwidth usage and redundancy, L2 protocols may support *link aggregation*, grouping multiple links into one virtual link.

A Link Aggregation Group (or LAG), viewed as a single link, can be included in a spanning tree, allowing their collective use for link protection and load balancing.

Link aggregation can be configured manually, or established dynamically by the Link Aggregation Control Protocol (LACP) [31]. LACP transmits LACP Data Unit (LACPDU) to inform the other end (*partner*) of its state and its understanding of partner state. Based on LACPDU, a LAG can be dynamically created and updated.

1) Attacks: A switch running LACP sends to, and receives from its partner, LACPDUs to maintain link aggregation. LACPDUs are typically sent over a point-to-point link, making Man-in-the-Middle (MitM) tampering difficult, but remaining vulnerable to *LACP spoofing* attacks because (1) LACP is usually implemented in the CPU (vs. data plane), allowing a switch to receive LACPDUs from remote entities; and (2) it has no security protection (*e.g.,* peer authentication) [31]. Thus, an external entity (*e.g.,* a host) may send forged LACPDUs to a switch to influence the state of its link aggregation, *e.g.,* to cause link instability or even Denial of Service (DoS).

2) Defenses: Implementing LACP in the data plane may ensure that LACPDUs are only received from a given port and never leave that port, mitigating forged LACPDU injection.

D. Device Redundancy

Device level redundancy ensures that the failure of one or more devices does not result in loss of network connectivity; *e.g.,* Cisco Switch Stacking (CSS) [32] allows a number of switches, usually of the same model, to form a redundancy group. Within a redundancy group, a master is elected dynamically, *e.g.,* based on bridge identifiers and/or priority values. If a master fails, a new master is elected to ensure ongoing network connectivity.

1) Attacks: Similar to attacks on STP, the process of dynamic master election is subject to spoofing attacks. Leveraging the lack of authentication, an adversary can send falsified messages to become the master upon device failure, thus control a device within the network itself.

2) Defenses: The election process should use cryptographic methods for origin authentication and message integrity to exclude unauthorized entities, *e.g.,* a host, from joining. Switches within a redundancy group should be connected via dedicated ports, and the election process should only run in those ports. Any election message received from other ports should be dropped. This prevents an adversary on the edge of the network from manipulating the process and itself become the master.

E. Scalability

Scalability is an important issue with STP and MAC learning; every switch must learn all MAC addresses and identifiers in the network. While easy in small networks, challenges arise in larger networks such as a large enterprise data center with many physical servers and virtual machines, each with several MAC addresses. The number of MAC addresses may exceed the MAC table capacity of a switch, and increase the delay of MAC table look ups.

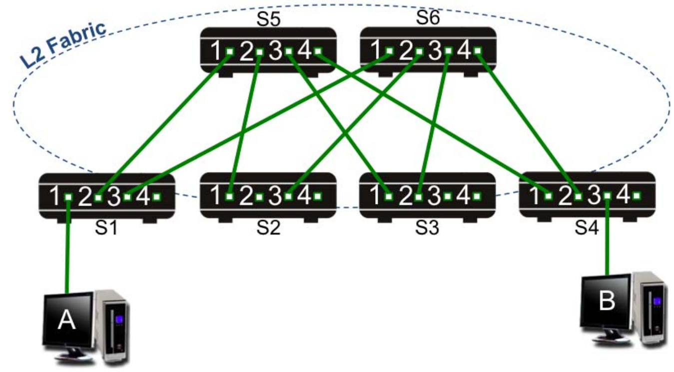


Fig. 5. L2 network with network fabric.

To improve scalability, VLANs may be used to divide a network into segments, each forming an isolated L2 broadcast domain. MAC learning then occurs within a VLAN, reducing MAC table size. L3 routing is used to connect VLANs.

Another method to improve scalability is to group core switches into domains called *network fabric*. In Fig. 5, switches S5, S6 and a subset of ports on switches S1–S4 (customer edge switches) form a fabric and run an L2 routing protocol, *e.g.,* Intermediate System to Intermediate System (IS-IS) [33], to learn the fabric network topology. In Fig. 5, a frame from A to B is encapsulated with an outer header with S1 as source address and S4 as destination address. The outer header usually introduces TTL to prevent indefinite forwarding within the fabric. Each customer edge switch learns all local MAC addresses and some remote MAC addresses; S5 and S6 learn no end-user MAC address. Example network fabric implementations are Transparent Interconnection of Lots of Links (TRILL) [34] and Shortest Path Bridging (SPB)—see IEEE 802.1aq [35].

1) Attacks: VLANs are subject to *VLAN hopping attacks*—traffic from one VLAN can be received by another, allowing L2 attacks against one VLAN to be launched from a different VLAN. One attack strategy is *VLAN double encapsulation* [36]. Another attack strategy is to exploit switch misconfiguration or VLAN auto negotiation protocols to impersonate another switch. In this way, a malicious host can pretend to be a switch and the link between the host and a switch would appear to be a trunk link, allowing the host to send and receive packets with any VLAN tag. Thus, the network isolation provided by VLAN is completely broken.

There are multiple means for implementing L2 network fabric, *e.g.,* TRILL and SPB, all of which use a routing protocol, typically IS-IS, to automatically discover and maintain the network topology inside the fabric. IS-IS is subject to several attacks, such as PDU spoofing, and DoS due to replaying hello messages [34], [37].

2) Defenses: VLAN-related vulnerabilities may be mitigated by disabling VLAN auto-negotiation, and configuring VLAN filtering in UNIs. Note that a packet may contain more than one VLAN tag; all such tags should be filtered if present in packets received from UNIs.

IS-IS vulnerabilities may be mitigated by enabling additional IS-IS cryptographic authentication [33], and ignoring unauthenticated PDUs. Further, IS-IS messages received from UNIs should be dropped.

Summary and Insights: This section reviewed the main protocols implementing L2 Ethernet networks, namely: MAC learning that enables targeted switch forwarding, STP protocol for loop prevention, LACP protocol for grouping and utilizing network links, device redundancy protocols like Cisco's switch stacking, and the usage of VLANs and TRILL fabrics for virtual network isolation thus enabling L2 scalability. Insights upon analyzing the discussed security threats and their defenses against each protocol are given in Sections VII and VIII below.

V. L3 Networks

As with L2 networks, we analyze security threats and mitigations for L3 networks related to each property in our framework (Section III). See Sections VII and VIII for discussions and insights.

A. Basic Forwarding

L3 devices, namely routers, perform two main tasks: route learning and packet forwarding [38]. In simple networks where two or more subnets are connected by a single router, the routing table is usually manually configured without running any routing protocol. A packet sent from one subnet to another has a destination MAC address of the default gateway of the source host, thus always arrives at the router via one of its interfaces. A router, upon receiving a packet from one subnet destined to another, performs three actions: removes the packet's L2 header, looks up the routing table for the next hop, and encapsulates the packet with a new L2 header for forwarding. A next hop in a routing table could be a local interface or an IP address. Routing table lookups are recursive until a next hop is a local interface. In this case, it further looks up the ARP table associated with that interface for the MAC address of the packet's next hop IP address. If not found, the router uses ARP to obtain the MAC address.

1) Attacks: Since an L3 router uses ARP to resolve the MAC address of a packet's destination IP, it is subject to ARP cache poisoning attacks [39]. This enables the attacker to associate its IP address with another (victim) MAC address, and thus receive the traffic intended for the victim's device.

2) Defenses: There are several approaches to address ARP cache poisoning. Dynamic ARP Inspection (DAI) [40] is a mechanism by which ARP responses are checked against (1) a central DB that binds IP to MAC addresses (this may be populated by listening to DHCP requests and responses in the network); or (2) static pre-configured ARP entries.

Cryptographic measures can also be used to distribute prepopulated IP-to-MAC-address mapping attestations, such as Ticket-based ARP (TARP) [41]. A voting-based protocol, requiring network consensus before updating ARP entries, has also been proposed by Nam *et al.* [42].

B. Loop Free Forwarding

In networks with multiple routers and redundant physical paths, a routing protocol is often used to advertise and learn routing information. Routing protocols are either *link state* (*e.g.,* IS-IS, OSPF—Open Shortest Path First [1]) or *distance vector* (*e.g.,* RIP—Routing Information Protocol [43]).

1) Attacks: While attack methods vary among different routing protocols, a common attack objective is *routing table poisoning*—to pollute network topology information and derived forwarding tables by advertising or injecting false routes through announcements. For example, a malicious router could advertise a malicious Link State Advertisement (LSA) (*e.g.,* with a false link cost) to influence other routers' calculation of routing tables. Such attack is easy to launch but has limited impact since a neighboring router will eventually advertise a correct LSA with a fresher sequence number, resulting in the removal of the falsified LSA from being used for routing table calculation. More advanced attacks (see [44]) can be launched to increase the effectiveness of routing table poisoning.

2) Defenses: To mitigate routing table poisoning attacks, three levels of defenses should be considered. First, routing protocols should only run in NNIs (routing updates received from UNIs should be dropped). This is to prevent an outsider (*e.g.,* a host) from participating in routing protocol communication. Second, message origin authentication should be implemented to prevent a malicious (compromised, previously legitimate) router from impersonating another router. Third, routing updates should be corroborated when being used to calculate routing tables. For example, a link cost advertised by one router should be corroborated with the link cost advertised by the other router on the same link.

C. Link Redundancy

L3 networks usually provide link redundancy through routing strategies like Equal Cost Multiple Path (ECMP) routing [45], [46], which allow packets to a common (*i.e.,* the same) destination address to be routed to their next hops over multiple links of equal cost.

1) Attacks: Multipath routing is a local decision made within a single router, requiring no interaction with adjacent or remote routers. Thus, it neither requires control protocols nor appears to introduce new security threats, other than link DoSing [46].

D. Device Redundancy

If a gateway router goes down, traffic across different subnets will be unable to reach their destinations, resulting in service outage. To improve availability, two or more routers often share a common virtual IP address and run a control protocol such as Virtual Router Redundancy Protocol (VRRP) [47] to dynamically elect a master as the default gateway of a subnet. When a master fails, VRRP dynamically selects another router as the master. VRRP runs over IP with an IP multicast address as its destination.

1) Attacks: Protocols for high availability routing such as VRRP [47] are subject to spoofing attacks. For example, VRRPv3 [47] does not include authentication of VRRP messages, thus an attacker may send a spoofed VRRP message with the highest priority to become the master of the router cluster. Such an attacker will receive all traffic to and from a subnet. While the previous versions of VRRP [48], [49] do include message authentication, it was removed from version 3 because it could be exploited to result in (malicious) election of multiple masters [47]. Other attacks, such as ARP spoofing, exist which could result in the same attack effect (*e.g.,* becoming the gateway of end hosts) [47].

2) Defenses: Dropping VRRP messages that arrive from a host-connected port prevents an attacker sitting at the network edge from spoofing such messages [47]. Additionally, VRRP message includes a TTL set to 255 by default. Upon receiving a VRRP message, a router validates the TTL field and discards a VRRP message whose TTL is not equal to 255. This limits the ability of remote attackers (*e.g.,* outside of a network) from spoofing VRRP packets.

E. Scalability

Within an AS, scalability in L3 networks is provided using routing protocols, supported by hierarchical routing. For example, OSPF allows a large network to be divided into subdomains (OSPF areas). Routers within an OSPF area need only maintain network topology information of the area they belong to. A backbone OSPF area is used to connect all other areas. Thus routing advertisements are limited to within an area, reducing the size of routing databases. Between ASes, BGP is used to advertise network reachability information.

Security threats of routing protocols, and their mitigation, are discussed in Section V-B above.

Summary and Insights: In this section, we discussed common protocols implementing L3 network functionalities. Those include: ARP to allow proper L2 addressing and thus forwarding across multiple subnets, routing protocols such as RIP and OSPF for loop-free routing, ECMP for routing along multiple paths and thus better utilization, VRRP for router redundancy thus increasing availability in case of equipment failure, and hierarchical routing across separate network domains providing network scalability. See Sections VII and VIII for insights upon analyzing and comparing the attack surfaces and defenses of these protocols with their counterpart in SDNs.

VI. Software Defined Networks

In SDN, the control plane of a device is implemented in an external entity, as opposed to within the device in a CN. This architectural difference impacts how network properties from our framework are provided. In CNs, a property achieved in the data plane is also considered achieved in the control plane. This does not hold in SDN due to the separation of planes. Thus for SDN, we discuss separately how each network property is provided for the data and control planes. We use OpenFlow switches [50] as an example in our discussion.

A. Basic Forwarding

Here we consider how a single SDN controller, controlling a single OpenFlow switch connected with a number of hosts, learns forwarding information. We assume that the controller has a direct connection with the switch, thus no need to learn about this control connection.

To configure the switch to provide connectivity among connected hosts, the controller must learn the mapping between hosts (*e.g.,* their MAC addresses) and switch ports. To do so, the controller may configure the switch to forward ARP requests and unknown packets, to the controller. An OpenFlow switch forwards such a packet to a controller using the `PACKET_IN` message, which includes the switch port from which the packet is received. The `PACKET_IN` message provides the controller with information about which hosts connect to which switch ports. If a destination host is also unknown, the controller instructs the switch to flood this packet using a `PACKET_OUT` message. The response from the destination will also be sent to the controller, allowing the controller to learn the location of both hosts. As a result, the controller can set up flow rules for the pair to communicate.

This learning process by a controller, called *Host Tracking Service*, is equivalent in principle to MAC learning by an L2 switch. It demonstrates how a conventional L2 control function is taken out from a switch, and implemented inside a controller. One difference is that MAC learning, being L2, only learns MAC addresses (possibly VLAN IDs). OpenFlow can learn both L2 addresses (MAC address and possibly VLAN ID) and IP addresses. We call this process *Host Learning*, for better comparability with MAC learning.

1) Attacks: Host learning by a controller, being based on information provided by a switch and hosts, is thus subject to spoofing attacks (MAC spoofing, IP spoofing, VLAN tag spoofing), since a dishonest host or switch can forge such information inside a packet. Often called *Host Location Hijacking* [11], here we call it *Host Profile Poisoning* for better comparability with MAC table poisoning. Host learning is also subject to flooding attacks—an attacker may generate a large number of packets with arbitrary MAC and IP addresses, resulting in creating (1) a large number of host profiles inside the controller, (2) a large number of messages sent to the controller, and (3) a large number of flow tables inside a switch. Thus, it is possible to cause DoS or packet interception, as unknown packets are also flooded. For example, if the memory allocated for host profiles in a controller is full, an existing host profile (*e.g.,* the oldest) will be overwritten, resulting in the flooding of a new packet destined to that host. This resembles an L2 MAC flooding attack.

Host-learning messages between switches and the controller may be exploited to cause a message forwarding loop.

2) Defenses: MAC binding (discussed earlier) can mitigate MAC spoofing attacks, albeit requiring static configuration. MAC limiting (also discussed earlier) can mitigate MAC flooding attacks, but cannot prevent MAC spoofing.

To mitigate VLAN spoofing, an OpenFlow switch can designate its ports as UNIs and NNIs, and remove VLAN tags in packets received from UNIs. Note: since there is only one switch, NNIs will not receive any traffic but the defense still works in the case of multiple switches. If the port an SDN controller connects to cannot be determined, and the controller needs to tag traffic for some reason, this defense becomes problematic.

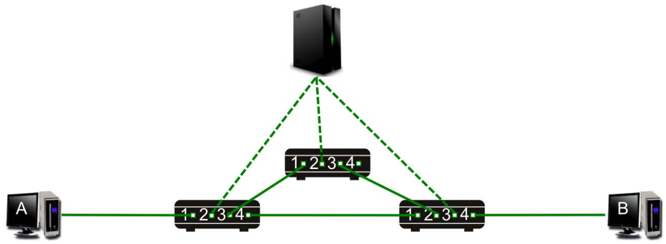


Fig. 6. An SDN with multiple switches.

To mitigate IP spoofing, OpenFlow controllers could avoid the learning of IP based forwarding rules from the data plane by planning IP address assignment and configuring flow tables with IP prefixes, acknowledging that prefix matching might be slower than precise matching.

B. Loop Free Forwarding

Here we consider a single controller controlling a number of OpenFlow switches (Fig. 6).

To configure forwarding tables on OpenFlow switches, the controller must first learn the network topology using a control protocol such as OpenFlow Discovery Protocol (OFDP). There are two scenarios to consider. (1) There exists a dedicated control network (*e.g.,* a direct link between each switch and the controller) such that each OpenFlow switch can communicate with the controller, *e.g.,* by establishing a TLS connection with the controller. (2) No such dedicated control network exists, and the controller must discover all switches and set up proper flow tables so they can begin communicating with the controller to receive further flows.

In case (1), the switch initiates communication with the controller during a boot-up process. The controller thus obtains information about individual switches under its control without running any control protocol. However, the controller does not know the connectivity between switches, *i.e.,* the network topology. In case (2), the controller does not know which switches are under its control and must use a protocol to discover them (see [51, Sec. 3.3]). Here we consider case (1), which is also commonly studied in other papers (*e.g.,* [11]).

OFDP works as follows. First, the OpenFlow controller sends a Link Layer Discovery Protocol (LLDP) [52] packet (inside a `PACKET_OUT` message with output port set to ALL) to every switch under its control. A switch receiving such a message floods to all of its ports. A switch receiving an LLDP packet from a neighbor switch must forward it to the controller, *e.g.,* due to the absence of flow rules for processing such a packet, triggering a default rule for forwarding unknown packets to the controller, or the existence of an explicit flow rule for forwarding LLDP packets to the controller. This `PACKET_IN` message to the controller also includes the port number that receives the LLDP packet. Thus, the controller discovers a link between two switches, and subsequently all links between all switches, allowing completion of a complete network topology.

1) Attacks: An OpenFlow controller, discovering network topology by learning from LLDP packets [52] sent by switches, is thus subject to LLDP spoofing attacks. An attacker, a switch or a host, may send falsified LLDP packets to a controller to contaminate computation of network topology. For example, a host may send falsified LLDP packets to insert itself and create non-existent links into the topology. This is a *Link Fabrication Attack* [11]. LLDP flooding may also force a controller to continuously re-calculate network topology, disrupting service.

2) Defenses: To mitigate LLDP spoofing from hosts, an OpenFlow switch can designate its ports as NNIs and UNIs, and reject LLDP packets received from UNIs. This defense would work if the port an SDN controller connects to is prior known. Otherwise, it will be problematic, since LLDP packets to and from the controller might also be filtered. Message authenticity and integrity, if implemented by LLDP, can effectively mitigate LLDP spoofing by hosts. Consistency checks of LLDP packets by a controller can mitigate LLDP spoofing by a host or switch. See Section VII-B for discussion on how to mitigate attacks from an SDN controller.

C. Link Redundancy

We note two options for implementing link aggregation for two OpenFlow switches with multiple physical links between them. (1) The OpenFlow switches, just as an L2 switch, run LACP between them to create a virtual link presented to the OpenFlow controller during link discovery. (2) An OpenFlow controller discovers shared links between two switches and uses a group table (available in OpenFlow v1.3 and later) to distribute traffic among a set of switch ports within a group. Here the OpenFlow controller must monitor the status of all links and update a corresponding group table upon detection of a link state change. Note: if an OpenFlow switch shares multiple links with a conventional switch or host running LACP, the OpenFlow switch can be configured to pass LACPDUs from the conventional switch to the controller to be processed. We consider the second case, since a switch implementing LACP is considered hybrid rather than pure SDN switch.

1) Attacks: An SDN controller is subject to message spoofing attacks. For example, an attacker may send false link up and down events to a controller, to manipulate the state of a link group. Such spoofing may be from a switch or host.

2) Defenses: Link state events should only be sent from OpenFlow switches, not hosts. If the port an SDN controller connects to is prior known, such events if received from other UNIs should be dropped. Message authenticity and integrity, if implemented by OpenFlow events, can effectively mitigate spoofing by hosts or switches. Consistency checks of OpenFlow events, if implemented by a controller, can detect falsified events by a host or switch.

D. Device Redundancy

For redundancy among a group of OpenFlow switches, a controller could monitor the status of each switch, and update switch configurations and flow tables accordingly to allow traffic go through a new switch. For redundancy among a group of OpenFlow controllers, OpenFlow switches are configured with multiple controllers. When the current controller goes down, a switch establishes a connection with the next controller. A distributed election protocol can also be used to elect a master controller, often for SDN applications. When the master fails, another slave replica becomes the master. The result of such master election can also be synchronized into the switch's configuration of controller preference to ensure all switches are also controlled by the same master. For example, ONOS uses ZooKeeper [53], a tool implementing an election protocol for distributed coordination and election. OpenDaylight uses Akka and Raft [54] for master election.

Further, controller states must also remain synchronized among controllers. For example, ONOS implements an eventual consistency model [55], in which a background process updates written objects in all replicas periodically.

1) Attacks: Beside spoofing attacks against device redundancy (see Section VI-C1), an election protocol used by controllers to achieve redundancy is subject to spoofing attacks. First, a non-controller entity (*e.g.,* a host) may join the election process to cause undesirable results. Second, a misbehaving controller may be able to manipulate the master election process (*e.g.,* by manually picking the smallest allowable time before candidacy election) to become the next master.

For example, Akka used by OpenDaylight for controller clustering employs no default security mechanisms [56], *e.g.,* for integrity protection of inter-cluster messages. In ONOS, we note no security mechanisms are used to ensure the integrity of update information. A timestamp may be forged, and if replicas are not properly authenticated, an attacker may impersonate one of them and manipulate stored objects.

2) Defense: If the ports SDN controllers connect to are prior known, election messages if received from other UNIs can be dropped. Otherwise, message origin authentication and integrity need to be implemented for the election protocol and state replication, *e.g.,* using mutually authenticated TLS among controllers, to prevent an outsider from participating in or tampering with the election process. Additional mechanisms, *e.g.,* information corroboration, are also needed to mitigate attacks by a misbehaving controller (*e.g.,* compromised by an attacker) participating in election with forged information.

E. Scalability

Unlike CNs, which run control protocols such as IS-IS to implement a network fabric for scalability and other benefits, OpenFlow controllers can configure flow rules in a way to improve scalability. For example, an OpenFlow controller may configure: (1) encapsulation rules at ingress switches, (2) forwarding rules based on outer headers at intermediate switches, and (3) decapsulation rules at egress switches. In this way, the controller creates tunnels shared by many individual flows and the intermediate switches only need to be configured with the rules related to tunnels, not individual flow, significantly reducing the number of rules required. Edge switches (ingress and egress) also only need to be configured with the rules relevant to the end hosts connected to them. In this way, OpenFlow controllers may create a network fabric without need of running additional control protocols.

Scalability in the SDN control plane can be provided by dividing a network into areas or domains, each controlled by one or more controllers. Controllers could be peer-to-peer or hierarchical. In a peer-to-peer model, area controllers synchronize states among themselves so that each maintains a consistent global view of the network. In a hierarchical model, lower level controllers maintain a subset of the global view; only a top level controller has the global view.

Distributed controllers need to communicate with each other to exchange reachability and state information. There is currently no standard defined for inter controller communication. A distributed protocol is usually needed. For example, BGP is suggested to be the message exchange protocol among SDN controllers [57]. ONOS relies on a distributed databases, *e.g.,* Cassandra [58] and Distributed Hash Tables (DHTs), for distributing network topology and state information among controllers.

1) Attacks: Vulnerabilities may arise from distributed communication in a peer-to-peer model, or from controller-to-controller communication in a hierarchical model. For example, if BGP is used to exchange information among controllers, vulnerabilities in BGP could be exploited to attack SDN. Vulnerabilities could also arise from the distributed database if it is used for synchronization among controllers. For example, without proper configuration, Cassandra may be vulnerable to query injection attacks [59].

2) Defense: As with any other distributed protocol such as an election protocol, communication among controllers, either peer-to-peer or hierarchical, must provide data origin authentication and message integrity to prevent outsiders from participating in or tampering with the communication. Further, an additional mechanism such as information corroboration is needed to mitigate misbehavior by legitimate controllers. This may also mitigate the effect of a compromised controller. Note, unlike in a simple network where the ports to which controllers connect to can be prior known, it is hard to define a communication boundary for distributed controllers to prevent outsiders from participating in the election process.

Summary and Insights: Upon discussing how each network property in our framework (Section III) is implemented in OpenFlow networks, we found that the mechanisms implementing them are similar in nature to their counterparts in L2 and L3 networks. For example, the host tracking service employed by OpenFlow controllers allows the controller to automatically learn about host locations in the network after these hosts initiate traffic, *i.e.,* similar to MAC learning. For the controller to set up loop-free flows, it has to first learn about the network topology using LLDP. OpenFlow switches run LACP for link aggregation, similar to L2. Controller redundancy is achieved by configuring OpenFlow switches with multiple controllers, and synchronously electing a master controller, *i.e.,* similar to VRRP router election on L3. In addition to controller redundancy, scalability in SDNs also requires hierarchical clustering of the network (see L3 scalability). In the following two sections, we likewise raise the question of whether the nature of threats and defenses is also similar for each of these mechanisms, and whether the threat model necessitates different handling of threats in SDNs and CNs (L2 and L3).

VII. SDN Versus Conventional Networks: Lessons Learned

Table I summarizes the control functions, attacks and defenses noted above. We also cite references in the table, where we are aware of suitable references, pertaining to the respective network property. We now discuss lessons learned upon comparing attacks and defenses of CN and SDN control planes following each of the two threat models in Section III, namely END-HOST and ALL-ELEMENT.

A. Basic Forwarding

MAC table poisoning (against a CN switch) and host profile poisoning (against an SDN controller), the two major threats respectively, are similar in nature but differ in details. For example, the attack vector of MAC table poisoning is MAC address spoofing, while both MAC and IP addresses could be spoofed in host profile poisoning. Since an L3 router with manually configured routing table does not learn forwarding information from the data plane, it is not subject to IP spoofing attacks. Another subtle difference is related to the size limit of the MAC table and memory allocated to host profiles when flooding is employed. MAC table size could vary from a few thousand to a few million entries, depending on the vendor and model of a switch; an SDN controller usually has larger memory and is thus less vulnerable to such flooding.

A CN is also less vulnerable to DoS attack (than an SDN network) because the SDN controller itself is a new attack surface, as is the link between a switch and an SDN controller, which could become a new bottleneck [60].

Defenses for MAC and host profile poisoning are similar. For example, port security could be used in a relatively static network to bind switch ports with MAC addresses for both CN and SDN. In a dynamic network where MAC addresses often change (*e.g.,* in a data center with server virtualization), static binding is problematic for both paradigms.

We do not discuss the ALL-ELEMENT threat model here since this simple network consists of only one switch and one controller. If the switch or the controller is malicious, the network would be completely compromised.

B. Loop Free Forwarding

Conventional L2 and L3 networks use STP and routing protocols (such as OSPF) for loop free forwarding. SDN uses LLDP for topology discovery.

In the END-HOST threat model, an end host can attack both CNs and SDN, resulting in incorrect forwarding tables by exploiting protocols' vulnerabilities. While the impact from such attacks appears comparable, attack techniques will differ since the protocols exploited differ.

Defenses for CNs can rely on UNI filtering. However, that works in SDN only if the attach points of the SDN controller are prior known and remain static. Otherwise, UNI filtering is ineffective and cryptographic mechanisms are required in SDN to prevent outsiders from participating in topology discovery.

In the ALL-ELEMENT threat model, the network (*e.g.,* L2 or L3 device and SDN controller itself) could be malicious; attacks and defenses then appear similar for CNs and SDN, albeit with subtle differences. From a threat model perspective, a malicious CN device or an SDN controller may be able to compromise the entire network (*e.g.,* influencing the routing table of any device within the network), acknowledging that a malicious SDN controller appears capable of causing more damage.

From a defense perspective, data origin authentication, message integrity, and consistency checks are all required by both CNs and the SDN control plane to counter insider attacks (*e.g.,* to detect and discard false information received from other legitimate nodes). In CNs, consistency checks can be done inside individual devices. In SDN, consistency checks should be done by both switches and controller. First, an OpenFlow switch should validate LLDP messages to ensure that it does not contain false information (*e.g.,* a faked link between the sender and the receiver). Second, an SDN controller should validate LLDP messages to rule out faked nodes and faked links. However, new defenses are needed to mitigate or reduce threats of a controller misbehaving in doing network topology discovery and route calculation. It appears difficult to contain damage from a misbehaving controller if it is monolithic. Thus, a controller is better divided into small, independent units to minimize the threat from a misbehaving control unit and facilitate cross-checking the behavior of each unit.

If adopted for SDN controllers, a micro service architecture [61] can serve this purpose. As an example of such an architecture, the control function providing loop free forwarding can be implemented in three micro services; the first is collecting and validating LLDP messages; the second is performing topology and route calculation; the third is updating flow rules in switches. Each micro service runs multiple instances, each of which cross-checks requests and responses from multiple other service instances. To cross-check behavior of flow rule updating services, other types of services such as real time flow validation (*e.g.,* VeriFlow [62]) can be implemented.

C. Link Redundancy

A CN uses LACP for link aggregation; an SDN controller can monitor link state changes, and update link groups in a switch accordingly. Both are vulnerable to message spoofing and tampering attacks.

In the END-HOST threat model, rules can be configured on UNIs to filter LACPDU packets for CNs. If the attach points of the SDN controller are prior known and static, rules can also be configured to filter link up/down events for SDN. Otherwise, cryptographic mechanisms are required by SDN to prevent outsiders from sending link up/down events to the SDN controller.

In the ALL-ELEMENT threat model, message origin authentication and message integrity can be used to address a legitimate switch spoofing a link group member. If a link group member itself misbehaves, it falls short for the other member to maintain a correct link group state since the misbehaving end can manipulate packets (*e.g.,* selectively dropping them) to achieve the same end.

In SDN, message origin authentication and integrity, *e.g.,* by mutually authenticated TLS, can mitigate a legitimate switch spoofing another switch by sending the controller faked link up and down events. Further, the function controlling link

TABLE I
A SUMMARY COMPARISON BETWEEN CONVENTIONAL NETWORKS (CNs) AND SDN

NETWORK PROPERTIES	**CONVENTIONAL NETWORKS** **LAYER-2**	**LAYER-3**	**SDN**
		Control Functions	
Basic Forwarding	MAC Learning	Static routes, Address Resolution Protocol (ARP)	Host Location Learning
Loop Free Forwarding	Spanning Tree Protocol (STP)	Routing Protocols (OSPF, RIP)	Link Layer Discovery Protocol (LLDP) and OpenFlow Discovery Protocol (OFDP)
Link Redundancy	Link Aggregation (LACP)	Equal Cost Multiple Path (ECMP)	Controller
Device Redundancy	Switch Stacking	Virtual Router Redundancy Protocol (VRRP)	Election Protocol
Scalability	VLAN, Network fabric (TRILL, VxLAN)	Routing Protocols (OSPF, IS-IS)	BGP, distributed DB
		Attacks	
Basic Forwarding	MAC table poisoning (MAC spoofing and MAC flooding) [22]	ARP table poisoning [39]	Host profile poisoning [11]
Loop Free Forwarding	BPDU spoofing, tampering and flooding [29], [23]	Routing advertisement spoofing [44]	Link fabrication [11]
Link Redundancy	LACPDU spoofing [31]	Link DoSing [45]	Spoofed link-manipulation messages (*e.g.,* [31])
Device Redundancy	Stacking spoofing	VRRP message spoofing [47]	Master election manipulation [56]
Scalability	VLAN hopping [36]; Switch impersonation [36]; Routing advertisement spoofing	Routing advertisement spoofing [44]	BGP attacks, distributed DB attacks
		Defences	
Basic Forwarding	Port Security [27]; MAC binding and limiting	Dynamic ARP inspection (DAI) [40]; Ticket-based ARP (TARP) [41]; Voting-based protocols [42]	MAC binding, host location validation [11]
Loop Free Forwarding	Root Guard [30]; BPDU filtering (prevent a host from masquerading as a switch)	UNI filtering and consistency check of routing advertisements	UNI filtering of LLDP packets [11], Authentication of LLDP and OFDP packets
Link Redundancy	LACP source port authentication (data plane implementation)	N/A	UNI filtering of control messages, mutual authentication of control channel
Device Redundancy	Run master election process on dedicated ports, authenticate devices involved in the process	UNI filtering of VRRP messages, TTL checks	Authenticity and integrity in master election
Scalability	VLAN filtering on UNIs, and disabling VLAN auto negotiation	UNI filtering and consistency check of routing advertisements	Authenticity and integrity in communication among SDN controllers

redundancy can be implemented in micro services, several running simultaneously to cross-check each others' behavior.

D. Device Redundancy

Both L2 and L3 use an election protocol to exchange messages among a device group to elect a master, thus being subject to spoofing attacks. In SDN, an election need not be implemented in switches for data plane redundancy, but is required for controller redundancy.

In the END-HOST threat model, rules can be configured on NNIs to filter control messages from end hosts. If the attach points of the SDN controller are prior known and static, such an approach can also be employed for SDN. Otherwise SDN requires message origin authentication and message integrity to counter outsider attacks.

In the ALL-ELEMENT threat model, message origin authentication and message integrity are required to prevent one legitimate device from impersonating another. An additional mechanism appears required to detect a legitimate device from participating in an election using false information. A similar mechanism appears required in SDN.

E. Scalability

For scalability in CN and SDN, respectively, routing protocols such as OSPF and BGP can be used. They are subject to similar attacks. Regarding defenses, the network boundary can be defined for CN to discard control messages from end hosts in the END-HOST threat model. In SDN this is less effective since SDN controllers often run insider servers that connect to the edge of a network; SDN controllers require cryptographic mechanisms to prevent outsiders from participating in the routing protocols.

In the ALL-ELEMENT threat model, to detect inside attacks, CNs and SDNs require message origin authentication and message integrity, as well as consistency checks.

VIII. Discussions and Further Insights

Our work shows that conventional networks and SDNs face similar threats on the control plane, *i.e.,* the mechanism for conducting attacks under the same threat model is comparable. Despite that, the two network paradigms necessitate different defenses. We observe the following from our analysis in Section VII. In the END-HOST threat model, conventional networks can define a network boundary to filter control messages from end hosts (see the effectiveness of "filtering-based" defenses for conventional networks in Table I). This approach is less effective in SDN—its control plane is implemented in SDN controllers, which are usually connected to the edge of the network; their attaching points, similar to end host locations, may change unless there is a dedicated control network separate from user networks. Thus, SDN largely requires cryptographic protection to prevent outsiders from participating in the control plane.

In the ALL-ELEMENT threat model, both conventional networks and SDN require cryptographic mechanisms, as well as consistency checks to mitigate insider attacks. This can be deduced from the *Defences* part in Table I, where data origin authentication (*e.g.,* LACP source port authentication, LLDP and OFDP packet authentication), mutual peer entity authentication (*e.g.,* between switches in CNs, along the SDN southbound control channel, and among SDN controllers in a physically distributed control plane), and integrity protection mechanisms (*e.g.,* in the master election process in STP and among distributed SDN controllers, and of LLDP messages for topology discovery in both SDNs and CNs) are cryptography-based techniques. Likewise, DAI, host location validation [11], and Port Security [27] are examples of consistency checking.

On the other hand, it can be noticed that where and how to implement cryptography and consistency checking differ in conventional networks and SDNs. While straightforward consistency checks might be implemented within individual CN devices, it is less obvious where and how to do this in SDN. Our analysis suggests that a highly modularized and distributed SDN software architecture may facilitate consistency checks, improving SDN control plane security (see Section VII-B, including for modularization based on micro service architecture). As noted earlier, current SDN controllers (*e.g.,* ONOS, OpenDaylight) lack mechanisms to mitigate insider threats.

IX. Related Work

To complement the preceding comparative analysis, we discuss relevant literature for security in SDN and counterparts in conventional networks (CNs). We also summarize surveys in the literature that focus specifically on SDN security.

A. Control Plane

1) Security-Oriented Controllers: As one of the first network operating systems, NOX [63] provides greater flexibility to the management plane. Despite lacking the ability to undertake most network functionalities by itself, NOX aims to provide sufficient APIs to ease the fulfillment of such functions. Porras *et al.* [12] proposed SE-Floodlight, a security enhanced system based on Floodlight [64]. Network administrators manually assign roles to applications, while SE-Floodlight mediates all OpenFlow operations to enforce a role-based permission model. SE-Floodlight also provides authentication and flow conflict resolution services (based on FortNOX [65]), both occurring on the system level independent of the applications, to enforce privilege separation (see also PermOF [66]). Shin *et al.* [67] identified several reasons for controller weaknesses, including lack of (1) resource control, (2) application separation, (3) application authentication and authorship, and (4) access control, and presented Rosemary as a (non-monolithic) micro-NOS to address these shortcomings.

2) Control Plane Security Extensions and APIs: Many security extensions to SDN controllers have been proposed to monitor and detect suspicious network behavior. VeriFlow [62] and FlowVisor [68] are two examples. The former provides real time checking and verification of forwarding behavior, whereas the latter enables network slicing such that each slice is typically under a different control domain, thus providing logical separation of multiple controller instances (see [69]). For such multi-slice networks, FlowChecker [70] is an example of a tool that checks consistency across multiple slices.

In contrast to VeriFlow, FlowGuard [71] is a firewall for SDNs that specifically focuses on conflict resolution. Note that SE-Floodlight resolves conflicts only between flows, whereas FlowGuard resolves conflicting network policies in general. To address the source-binding problem within the network, FlowTags [72] enables switches to tag packets for appropriate source binding, avoiding conflict with middleboxes in the network. Kim *et al.* [73] proposed Kinetic to not only monitor network properties, but also enable administrators to take appropriate control actions in response to network changes, and to analyze source of errors in control programs leading to the undesired network behavior. Similarly, Flover [74] and NetPlumber [75] are systems to verify that flow policies do not contradict with desired network security policies.

The proper extent of privileges that should be granted to an SDN application is unclear. Excessive privileges may constitute a significant weak point if an application becomes compromised or is malicious [76]; too few may not allow sufficient flexibility to run security applications. Fresco [5] is a framework for developing security SDN applications, providing APIs for developers to access sensitive network resources securely. Similarly, OperationCheckpoint [77] aims to secure the network against third party applications by ensuring that critical operations can be executed by trusted applications only.

3) Attack Mitigation: Benton *et al.* [78] highlight the importance of isolating applications running on top of the controller and the importance of verifying flow tables, to avoid erroneous controllers (including errors introduced without a malicious intent). Braga *et al.* [79] showed how machine learning can be used to identify traffic involved in a DoS attack. BASE [80] was proposed as an anti-spoofing mechanism, aiming to mitigate DoS. SD-Anti-DDoS [81] is another tool used to *clean* bloated flow tables after a DoS attack.

4) Proposals for Control Plane Scalability: To ensure scalability and fault tolerance, numerous proposals advocate replication and distribution of the control plane [82]–[84]. Onix [85] is a prominent example that abstracts the network distribution state to the control plane running on top. HyperFlow [86] allows multiple separate SDN domains to be consolidated and controlled from a single point.

B. Data Plane

Despite SDN's promise to ease management and service deployment, it is becoming clear that not all services can simply be implemented as applications on the controller. Depending on the required levels of network support, some security solutions in the literature propose either low-level modules running in controller kernel space, or software running on switches. These create challenges in managing/updating network elements, as is the case with CNs. The OpenFlow Extension Framework (OFX) [87] leverages control plane centralization to allow dynamic installation of software on switches.

1) Security Services: Distinct from CNs, the centralization of the control plane in SDNs challenges conventional means by which gateway services, *e.g.,* firewalls and IDSs, are set up. For example, instructing edge switches to forward a copy of the traffic to a separate IDS box would consume substantial delay and bandwidth [88]. FleXam [89] is an OpenFlow extension to enable a switch to send sample packets (including payload) from a specific flow to the controller.

2) Handling Compromised Switches: Compromising an SDN switch enables a wide range of MitM and impersonation attacks [90]; detection in SDN however differs from CNs. Chi *et al.* [91] implemented applications that periodically sample flow rules, and check if a random subset of switches are behaving as instructed.

3) DoS Attacks: While SDN is considered more vulnerable to DoS attacks [60] than CNs, mitigation techniques appear to follow non-conventional solutions. For example, FloodGuard [92] rate-limits packets sent to the controller to protect controller bandwidth from being consumed by intentional `PACKET_IN` requests from clueless switches. SDNsec [93] allows edge switches to encode whole routes on each ingress packet, thus mitigating DoS by state exhaustion. Flow aggregation [94] and flow time-out [95] are also widely regarded as good practices to mitigate switch memory exhaustion.

4) MitM Attacks: Sphinx [13] aims to counter security threats from within an SDN network (*e.g.,* from network switches and end hosts) by building flow graphs to represent a closed form of the network topology, and using these to detect anomalous switch behavior. Another tool, TopoGuard [11], aims to detect LLDP hijacking and MitM attacks, by validating the network view seen by the controller, to mitigate host location hijacking and link fabrication vulnerabilities.

5) Stateful SDN: FAST [96] and OpenState [97] are among the first proposals to suggest the need to re-include some intelligence back in switches. For the goal of increasing efficiency by reducing the southbound communication overhead, switches can retain some network state that enables them to, *e.g.,* identify and change flows without consulting the controller. This can be performed programmatically, where the controller enables the switch to know how to react to packet changes. Stateful SDNs were found vulnerable to several attacks [98], including exhaustion of the switch's state memory, and state inconsistency across multiple switches for the same flow. The latter can be mounted from an end host, by injecting specially tailored packets into the network.

C. Security Surveys

Kreutz *et al.* [99] made the first attempt to provide recommendations on improving SDN security by discussing seven threats against SDN and possible countermeasures. The threats address each stage of communication between the switches, controller, and applications, in addition to the interfaces within these stages. The authors highlight the limitations of TLS to secure communications to/from the controller, and the consequences of a hijacked such session. The authors also point out the issue of trusting networking applications, whereby a compromised application could jeopardize the security of the whole network.

Scott-Hayward *et al.* [100] surveyed the literature for potential control and data plane attacks and mitigation techniques, listing security issues discussed so far. The authors categorize research in SDN security as Analysis, Enhancement, or Solution, providing information as to which bound (north or south) is the research conducted on, and whether protocols other than OpenFlow are analyzed. A set of security threats, such as unauthorized access and data leakage, are enumerated and mapped against each of the five SDN layers and interfaces.

Klöti *et al.* [94] used the Microsoft STRIDE model and attack trees with the assumption that the controller, network elements, and the control channel between the two are all secure. They identify several possible threats including DoS by overloading the flow table, information disclosure, and tampering the flow table through cache poisoning. Information disclosure might allow a host to deduce the flow states in an OpenFlow switch by measuring the time difference in TCP connection setup. If an OpenFlow switch to which both the client and a TCP server connect implements flow aggregation, a fast TCP setup could imply that an aggregated flow rule has already been installed in the switch, which was likely triggered by the communication to the TCP server from other clients connecting to the same OpenFlow switch. While interesting, it is not clear if such a side-channel attack will be effective in a network with large number of switches. In general, some flow states could be obtained by scanning the whole network to reconstruct the network topology.

Alsmadi and Xu [101] similarly used the STRIDE model. In a brief comparison with CNs, Alsmadi and Xu [101] submit that the DoS attack surface is larger in SDN, but should more easily be mitigated through the controller. On the other hand, spoofing is likely more controllable in SDN due to reduction in stale information (such as ARP entries) that aid in spoofing. Finally, the authors concluded that while the standard attacks are not expected to differ substantially between SDN and CNs, SDN appears to allow for a larger attack surface.

Ahmad *et al.* [102] similarly conduct a security survey, summarizing efforts in the literature under the taxonomy: *Application*, *Control*, and *Data* planes. Such a taxonomy helps the community identify persistent weak spots in the paradigm, and where future research efforts could and should be directed.

Ali *et al.* [103] similarly presented a survey of security research in SDN, classifying the literature into either proposals for protection against attacks, or further enhancing network security. Many of the proposals falling under the former category work towards threat detection and remediation (*e.g.,* DoS and traffic anomalies), and verifying that the networking is behaving the way it should; proposals of the latter category utilize the SDN paradigm to provide security as a service, such as enhancing online anonymity and outsourcing network security management.

Schehlmann *et al.* [104] counted the number of security aspects where SDN is better than CNs, and reached the conclusion that SDN is generally better, after considering their advantages over CNs. Schehlmann *et al.* [104] argue that the newly added requirements of application authenticity and controller availability count towards the *negative* factors (*i.e.,* negatively affecting SDN security). However, they concluded that overall the security benefits offered by SDN outweigh the risk, and that SDN does not introduce new types of threats nor require new countermeasures—the paradigm rather needs to leverage existing well-known security mitigation and technologies like PKI for protection. While this evaluation is interesting, it does not provide specific guidelines to SDN development community on how to design and develop secure SDN.

Rawat and Reddy [105] summarized cases where an SDN can act as a security solution for classical attacks like network intrusions (*e.g.,* SDNIPS [15]), anomaly detection, and DoS detection (*e.g.,* [106]). They also covered cases whereby the three planes of an SDN can be attacked. Attacks were classified as either spoofing, intrusion, causing network anomalies, or DoS attacks.

Summary of Surveys: The above surveys are useful in summarizing the SDN security literature, showing how various attacks on SDN can be mounted on different planes, and how SDNs can be used to enhance network security. However, with the exception of Klöti *et al.* [94], none of the above surveys directly uses formal threat modeling to analyze how different assumptions made about attackers can affect the feasibility of these attacks. Klöti *et al.* [94] assumed an attacker model with access to the data plane, and anlayzed two attacks following the END-HOST model. In addition, the consequences of the reviewed attacks are not always assessed in terms of the affected network functionality. Our work complements existing literature as it (1) filters out threats against specific network properties, thus providing the ability of objective comparison with CNs; and (2) clearly defines two threat models, enabling an objective assessment to the difficulty of conducting the respective attacks, and the measures required by practitioners to protect against them.

X. FUTURE CHALLENGES AND RESEARCH DIRECTIONS

In this section, we discuss future research directions towards understanding the control plane security of SDNs.

A. *Other Threat Models*

We provided an objective comparison between CNs and SDNs across two threat models indicating the attacker's position in the network. New and possibly finer-grained threat models could be explored, such as an attacker compromising a single (possibly unprivileged) application on the controller; or compromising one switch versus many, and whether that switch is edge (ingress or egress) or not. Such models can help better understand situations where certain defenses would be more effective than others, and possibly which paradigm would be better for certain applications. In general, the five properties provided in our framework provide a baseline for further threat models to be analyzed against these properties.

B. *Expanding Network Properties*

The properties we analyzed herein are required by any production network to operate successfully. It would therefore be useful to also analyze threats introduced by mechanisms implementing other network services. Examples include multicast and anycast routing, Quality of Service (QoS), VPNs [17], and other traffic engineering techniques. Those can be investigated on both network paradigms, CNs and SDNs, which would allow for the comparison of threats and defenses across both paradigms, analogous to the analysis conducted herein for fundamental network services.

C. *Analyzing the Effect of Network Technologies*

We have explored attacks and defenses when Ethernet protocols implement the five properties in L2 networks. Other technologies can also be evaluated as they implement these five network properties, particularly for wireless communication. Examples include the IEEE 802 family, including WiFi networks [107], and other wireless technologies showing potential deployment for SDNs [108].

D. Distributed Robust Controllers

From use of our framework, it becomes clear that it is crucial to design defenses withstanding the ALL-ELEMENT threat model. That is especially true with the increasing trends of virtualizing the network infrastructure, which is facilitated by the proliferation of the SDN paradigm. Such a virtualization makes network devices, *e.g.,* switches, more prone to compromise due to the fact that conventional network boundary is disappearing thus it is difficult to enforce edge based filtering in SDN. Future research in that direction should thus aim to design modularized SDN controllers for partition tolerance of control domains, with the aim of improving SDN control plane security under the ALL-ELEMENT threat model.

E. Evaluating SDN Controllers in the Wild

An interesting research direction would be to use the new framework for conducting a real-world comprehensive analysis of current SDN controllers (*e.g.,* OpenDaylight [7]) for the security vulnerabilities and defenses discussed herein. Attacks could particularly follow the two threat models defined herein, or others as suitably required by target applications. Such experimental testing helps researchers and practitioners grasp the extent by which proposed solutions can withstand true attacks in real-world environments.

XI. Concluding Remarks

This paper provides a framework allowing an objective security comparison between SDNs and conventional networks. The framework consists of five network properties and two threat models. The identified properties are critical for successful operation of networks in practice, as they enable networks to function efficiently, scale, and ensure the networks' high availability. The two threat models provide varying degrees of adversarial capabilities, enabling the analysis of differences and similarities in the attack surface of protocols implementing each of the five properties in both network paradigms. To the best of our knowledge, this is the first framework that allows for such an apples-to-apples comparison of both paradigms, exploring both attacks and defenses.

Previous literature compared the security of SDNs and conventional networks either by focusing on less common functionalities, or by enumerating the number of points/devices (in the network) that are vulnerable to specific attacks, *e.g.,* DoS. Albeit useful to the academic research community, this does not provide answers to practitioners of large production networks who are interested in migrating their data centers from conventional networks to SDNs, but unable to grasp the nature of their new threats. In contrast, by reducing the focus of the analysis to the subset of properties that are likely implemented by all production networks in practice, our framework helps security researchers and practitioners understand and re-adapt their currently implemented defenses in conventional networks to suit the new SDN paradigm. The framework is useful to both network administrators and security researchers; we thus believe this work will help guide further SDN research, and aid practitioners in the design, development, and deployment of SDNs with stronger robustness and security properties.

Acknowledgment

The authors thank Yapeng Wu and Xingjun Chu for their constructive discussion and comments on this paper.

References

[1] J. Moy, "OSPF version 2," IETF, Fremont, CA, USA, RFC 2328, Apr. 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2328.txt
[2] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: An intellectual history of programmable networks," *ACM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, 2014.
[3] S. Giacalone, D. Ward, J. Drake, A. Atlas, and S. Previdi, "OSPF traffic engineering (TE) metric extensions," IETF, Fremont, CA, USA, RFC 7471, Mar. 2015. [Online]. Available: http://www.ietf.org/rfc/rfc7471.txt
[4] K. Greene. (Feb. 2009). *TR10: Software-Defined Networking*. [Online]. Available: http://www2.technologyreview.com/news/412194/tr10-software-defined-networking/
[5] S. Shin *et al.*, "Fresco: Modular composable security services for software-defined networks," in *Proc. NDSS*, Feb. 2013, pp. 1–15.
[6] Broadcom. (2015). *OpenFlow-Data Plane Abstraction Networking Software*. [Online]. Available: http://www.broadcom.com/products/ethernet-communication-and-switching/switching/of-dpa-software
[7] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a model-driven SDN controller architecture," in *Proc. IEEE World Wireless Mobile Multimedia Netw.*, Sydney, NSW, Australia, 2014, pp. 1–6.
[8] P. Berde *et al.*, "ONOS: Towards an open, distributed SDN OS," in *Proc. ACM HotSDN*, Chicago, IL, USA, 2014, pp. 1–6.
[9] R. Enns, M. Bjorklund, and J. Schoenwaelder, "Network configuration protocol (NETCONF)," IETF, Fremont, CA, USA, RFC 6241, Jun. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6241.txt
[10] P. C. van Oorschot, T. Wan, and E. Kranakis, "On interdomain routing security and pretty secure BGP (psBGP)," *ACM Trans. Inf. Syst. Security*, vol. 10, no. 3, 2007, Art. no. 11.
[11] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning network visibility in software-defined networks: New attacks and countermeasures," in *Proc. NDSS*, San Diego, CA, USA, Feb. 2015, pp. 1–15.
[12] P. Porras, S. Cheung, M. Fong, K. Skinner, and V. Yegneswaran, "Securing the software defined network control layer," in *Proc. NDSS*, San Diego, CA, USA, Feb. 2015, pp. 1–15.
[13] M. Dhawan, R. Poddar, K. Mahaf, and V. Mann, "SPHINX: Detecting security attacks in software-defined networks," in *Proc. NDSS*, Feb. 2015, pp. 1–15.
[14] A. S. Thyagaturu, A. Mercian, M. P. McGarry, M. Reisslein, and W. Kellerer, "Software defined optical networks (SDONs): A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2738–2786, 4th Quart., 2016.
[15] T. Xing, Z. Xiong, D. Huang, and D. Medhi, "SDNIPS: Enabling software-defined networking based intrusion prevention system in clouds," in *Proc. IEEE CNSM*, Rio de Janeiro, Brazil, 2014, pp. 308–311.
[16] *Information Technology—Open Systems Interconnection—Basic Reference Model: The Basic Model*, ISO/IEC Standard 7498-1: 1994, 1994.
[17] A. R. Sharafat, S. Das, G. Parulkar, and N. McKeown, "MPLS-TE and MPLS VPNs with OpenFlow," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 452–453, 2011.
[18] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "OpenFlow: Meeting carrier-grade recovery requirements," *Comput. Commun.*, vol. 36, no. 6, pp. 656–665, 2013.
[19] B. C. Neuman, *Scale in Distributed Systems*, Inf. Sci. Inst., Univ. Southern California, Los Angeles, CA, USA, 1994.
[20] International Telegraph and Telephone Consultative Committee, "ITU-T X.800: Security architecture for open systems interconnection for CCITT applications," Data Communication Networks: OSI; Security, Structure and Applications, 1991.
[21] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for software defined networking," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 655–685, 1st Quart., 2016.

[22] S. Convery. (2002). *Hacking Layer 2: Fun With Ethernet Switches*. [Online]. Available: ftp://n.manet.nu/pub/doc/Hacking_Layer_2_Fun_with_Ethernet_Switches.pdf
[23] T. Kiravuo, M. Sarela, and J. Manner, "A survey of Ethernet LAN security," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 3, pp. 1477–1491, 3rd Quart., 2013.
[24] K. R. Fall and W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Boston, MA, USA: Addison-Wesley, 2011.
[25] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Wormhole attacks in wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 2, pp. 370–380, Feb. 2006.
[26] N. Tsitsiroudi, P. Sarigiannidis, E. Karapistoli, and A. A. Economides, "Eyesim: A mobile application for visual-assisted wormhole attack detection in IoT-enabled WSNs," in *Proc. IFIP Wireless Mobile Netw. Conf. (WMNC)*, Colmar, France, 2016, pp. 103–109.
[27] S. Wilkins, *Switchport Security Concepts and Configuration*, Cisco Press, Indianapolis, IN, USA, Jul. 2011.
[28] R. Perlman, "An algorithm for distributed computation of a spanning tree in an extended LAN," in *Proc. ACM SIGCOMM*, 1985, pp. 44–53.
[29] E. Vyncke and C. Paggen. *Attacking the Spanning Tree Protocol*. Accessed: Mar. 20, 2018. [Online]. Available: http://www.ciscopress.com/articles/article.asp?p=1016582&seqNum=2
[30] *Safe Layer 2 Security in-Depth Version 2*, Cisco, San Jose, CA, USA, 2004. [Online]. Available: http://www.cisco.com/warp/public/cc/so/cuso/epso/sqfr/sfblu_wp.htm#wp998892
[31] IEEE, *Standard for Local and Metropolitan Area Networks—Link Aggregation*, IEEE Standard 802.1AXTM-2014, pp. 1–320, Dec. 2014.
[32] *Catalyst 3750-X and 3560-X Switch Software Configuration Guide—Managing Switch Stacks*, Cisco, San Jose, CA, USA, Nov. 2014. [Online]. Available: http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst3750x_3560x/software/release/12-2_55_se/configuration/guide/3750xscg/swstack.pdf
[33] T. Li and R. Atkinson, "IS-IS cryptographic authentication," IETF, Fremont, CA, USA, RFC 5304, Oct. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5304.txt
[34] R. Perlman, D. Eastlake, D. Dutt, S. Gai, and A. Ghanwani, "Routing bridges (RBridges): Base protocol specification," IETF, Fremont, CA, USA, RFC 6325, Jul. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6325.txt
[35] D. Fedyk and M. Seaman. (Mar. 2012). *802.1aq—Shortest Path Bridging*. [Online]. Available: http://www.ieee802.org/1/pages/802.1aq.html
[36] Cisco Networking Academy, *Routing and Switching Essentials Companion Guide*, Cisco Press, Indianapolis, IN, USA, Feb. 2014.
[37] D. Eastlake, T. Senevirathne, A. Ghanwani, D. Dutt, and A. Banerjee, "Transparent interconnection of lots of links (TRILL) use of IS-IS," IETF, Fremont, CA, USA, RFC 7176, May 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7176.txt
[38] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, vol. 1, 6th ed. Boston, MA, USA: Pearson High. Educ., 2013.
[39] Y. Volobuev. (1997). *Playing Redir Games With ARP and ICMP*. [Online]. Available: http://www.goth.net/iceburg/tcp/arp.games.html
[40] C. Systems. (2004). *Understanding and Configuring Dynamic ARP Inspection*. [Online]. Available: http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst4500/12-2/25ew/configuration/guide/conf/dynarp.html#wp1039231
[41] W. Lootah, W. Enck, and P. McDaniel, "TARP: Ticket-based address resolution protocol," *Comput. Netw.*, vol. 51, no. 15, pp. 4322–4337, 2007.
[42] S. Y. Nam, D. Kim, and J. Kim, "Enhanced ARP: Preventing ARP poisoning-based man-in-the-middle attacks," *IEEE Commun. Lett.*, vol. 14, no. 2, pp. 187–189, Feb. 2010.
[43] C. Hedrick, "Routing information protocol," IETF, Fremont, CA, USA, RFC 1058, Jun. 1988. [Online]. Available: http://www.ietf.org/rfc/rfc1058.txt
[44] G. Nakibly, E. Menahem, Y. Elovici, and A. Waizel, "Owning the routing table—Part II," in *Proc. BlackHat*, 2013, pp. 1–19.
[45] D. Thaler and C. Hopps, "Multipath issues in unicast and multicast next-hop selection," IETF, Fremont, CA, USA, RFC 2991, Nov. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2991.txt
[46] C. Hopps, "Analysis of an equal-cost multi-path algorithm," IETF, Fremont, CA, USA, RFC 2992, Nov. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2992.txt
[47] S. Nadas, "Virtual router redundancy protocol (VRRP) version 3 for IPv4 and IPv6," IETF, Fremont, CA, USA, RFC 5798, Mar. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5798.txt
[48] S. Knight *et al.*, "Virtual router redundancy protocol," IETF, Fremont, CA, USA, RFC 23388, Apr. 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2338.txt
[49] R. Hinden, "Virtual router redundancy protocol (VRRP)," IETF, Fremont, CA, USA, RFC 3768, Apr. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3768.txt
[50] *OpenFlow Switch Specification Version 1.5.0*, Open Netw. Found., Menlo Park, CA, USA, Dec. 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf
[51] M. Casado *et al.*, "SANE: A protection architecture for enterprise networks," in *Proc. USENIX Security Symp.*, Vancouver, BC, Canada, 2006, Art. no. 10.
[52] IEEE, *Standard for Local and Metropolitan Area Networks—Station and Media Access Control Connectivity Discovery*, IEEE Standard 802.1AB-2009, pp. 1–204, Sep. 2009.
[53] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "ZooKeeper: Wait-free coordination for Internet-scale systems," in *Proc. USENIX ATC*, Boston, MA, USA, 2010, p. 11.
[54] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proc. USENIX ATC*, Philadelphia, PA, USA, 2014, pp. 305–320.
[55] W. Vogels, "Eventually consistent," *Commun. ACM*, vol. 52, no. 1, pp. 40–44, 2009.
[56] Akka Documentation. (2015). *Network Security*. [Online]. Available: http://getakka.net/docs/remoting/security
[57] H. Yin *et al.*, "SDNi: A message exchange protocol for software defined networks (SDNs) across multiple domains," IETF, Fremont, CA, USA, Internet Draft, draft-yin-sdn-sdni-00.txt, Jun. 2012. [Online]. Available: http://tools.ietf.org/id/draft-yin-sdn-sdni-00.txt
[58] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, 2010.
[59] A. Ron, A. Shulman-Peleg, and E. Bronshtein, "No SQL, no injection?" in *Proc. IEEE S&P Workshop Web 2.0 Security Privacy*, 2015, pp. 1–4.
[60] S. Shin and G. Gu, "Attacking software-defined networks: A first feasibility study," in *Proc. ACM HotSDN*, 2013, pp. 165–166.
[61] N. Dragoni *et al.*, "Microservices: Yesterday, today, and tomorrow," *CoRR*, vol. abs/1606.04036, pp. 195–216, 2017, doi: 10.1007/978-3-319-67425-4_12.
[62] A. Khurshid, W. Zhou, M. Caesar, and P. Godfrey, "VeriFlow: Verifying network-wide invariants in real time," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 467–472, 2012.
[63] N. Gude *et al.*, "NOX: Towards an operating system for networks," *ACM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, Jul. 2008.
[64] (2015). *Floodlight OpenFlow Controller*. [Online]. Available: http://www.projectfloodlight.org/floodlight/
[65] P. Porras *et al.*, "A security enforcement kernel for OpenFlow networks," in *Proc. ACM HotSDN*, Helsinki, Finland, 2012, pp. 121–126.
[66] X. Wen, Y. Chen, C. Hu, C. Shi, and Y. Wang, "Towards a secure controller platform for OpenFlow applications," in *Proc. ACM HotSDN*, Hong Kong, 2013, pp. 171–172.
[67] S. Shin *et al.*, "Rosemary: A robust, secure, and high-performance network operating system," in *Proc. ACM CCS*, Scottsdale, AZ, USA, 2014, pp. 78–89.
[68] R. Sherwood *et al.*, "Can the production network be the testbed?" in *Proc. USENIX OSDI*, Vancouver, BC, Canada, 2010, pp. 365–378.
[69] S. Gutz, A. Story, C. Schlesinger, and N. Foster, "Splendid isolation: A slice abstraction for software-defined networks," in *Proc. ACM HotSDN*, Helsinki, Finland, 2012, pp. 79–84.
[70] E. Al-Shaer and S. Al-Haj, "FlowChecker: Configuration analysis and verification of federated OpenFlow infrastructures," in *Proc. ACM SafeConfig. Workshop*, Chicago, IL, USA, 2010, pp. 37–44.
[71] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, "FLOWGUARD: Building robust firewalls for software-defined networks," in *Proc. ACM HotSDN*, Chicago, IL, USA, 2014, pp. 97–102.
[72] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using FlowTags," in *Proc. USENIX NSDI*, Seattle, WA, USA, 2014, pp. 533–546.
[73] H. Kim *et al.*, "Kinetic: Verifiable dynamic network control," in *Proc. USENIX NSDI*, Oakland, CA, USA, 2015, pp. 59–72.
[74] S. Son, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Model checking invariant security properties in OpenFlow," in *Proc. IEEE ICC*, 2013, pp. 1974–1979.

[75] P. Kazemian *et al.*, "Real time network policy checking using header space analysis," in *Proc. USENIX NSDI*, Lombard, IL, USA, 2013, pp. 99–112.
[76] S. Matsumoto, S. Hitz, and A. Perrig, "Fleet: Defending SDNs from malicious administrators," in *Proc. ACM HotSDN*, Chicago, IL, USA, Aug. 2014, pp. 103–108.
[77] S. Scott-Hayward, C. Kane, and S. Sezer, "OperationCheckpoint: SDN application control," in *Proc. IEEE ICNP*, Raleigh, NC, USA, 2014, pp. 618–623.
[78] K. Benton, L. J. Camp, and C. Small, "OpenFlow vulnerability assessment," in *Proc. ACM HotSDN*, Hong Kong, 2013, pp. 151–152.
[79] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in *Proc. IEEE LCN*, Denver, CO, USA, 2010, pp. 408–415.
[80] J. Kwon *et al.*, "An incrementally deployable anti-spoofing mechanism for software-defined networks," *Comput. Commun.*, vol. 64, pp. 1–20, Jun. 2015.
[81] Y. Cui *et al.*, "SD-anti-DDoS: Fast and efficient DDoS defense in software-defined networks," *Netw. Comput. Appl.*, vol. 68, pp. 65–79, Jun. 2016.
[82] F. Botelho, A. Bessani, F. M. V. Ramos, and P. Ferreira, "On the design of practical fault-tolerant SDN controllers," in *Proc. IEEE Eur. Workshop SDN (EWSDN)*, London, U.K., 2014, pp. 73–78.
[83] S. Betgé-Brezetz, G.-B. Kamga, and M. Tazi, "Trust support for SDN controllers and virtualized network applications," in *Proc. IEEE NetSoft*, London, U.K., 2015, pp. 1–5.
[84] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proc. ACM HotSDN*, Helsinki, Finland, 2012, pp. 7–12.
[85] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," in *Proc. USENIX OSDI*, vol. 10. Vancouver, BC, Canada, 2010, pp. 351–364.
[86] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *Proc. USENIX INM/WREN*, San Jose, CA, USA, 2010, p. 3.
[87] J. Sonchack, J. M. Smith, A. J. Aviv, and E. Keller, "Enabling practical software-defined networking security applications with OFX," in *Proc. NDSS*, 2016, pp. 1–15.
[88] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *Recent Advances in Intrusion Detection*. Heidelberg, Germany: Springer, 2011, pp. 161–180.
[89] S. Shirali-Shahreza and Y. Ganjali, "Efficient implementation of security applications in OpenFlow controller with FleXam," in *Proc. IEEE High Perform. Interconnects (HOTI)*, San Jose, CA, USA, 2013, pp. 49–54.
[90] M. Antikainen, T. Aura, and M. Särelä, "Spook in your network: Attacking an SDN with a compromised OpenFlow switch," in *Secure IT Systems* (Lecture Notes in Computer Science), vol. 8788. Cham, Switzerland: Springer, 2014, pp. 229–244.
[91] P.-W. Chi, C.-T. Kuo, J.-W. Guo, and C.-L. Lei, "How to detect a compromised SDN switch," in *Proc. IEEE NetSoft*, London, U.K., 2015, pp. 1–6.
[92] H. Wang, L. Xu, and G. Gu, "FloodGuard: A DoS attack prevention extension in software-defined networks," in *Proc. IEEE/IFIP Dependable Syst. Netw. (DSN)*, Rio de Janeiro, Brazil, 2015, pp. 239–250.
[93] T. Sasaki, C. Pappas, T. Lee, T. Hoefler, and A. Perrig, "SDNsec: Forwarding accountability for the SDN data plane," in *Proc. IEEE ICCCN*, 2016, pp. 1–10.
[94] R. Klöti, V. Kotronis, and P. Smith, "OpenFlow: A security analysis," in *Proc. IEEE ICNP*, Göttingen, Germany, 2013, pp. 1–6.
[95] R. Kandoi and M. Antikainen, "Denial-of-service attacks in OpenFlow SDN networks," in *Proc. IFIP/IEEE Integr. Netw. Manag. (IM)*, Ottawa, ON, Canada, 2015, pp. 1322–1326.
[96] M. Moshref, A. Bhargava, A. Gupta, M. Yu, and R. Govindan, "Flow-level state transition as a new switch primitive for SDN," in *Proc. ACM HotSDN*, Chicago, IL, USA, 2014, pp. 61–66.
[97] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "OpenState: Programming platform-independent stateful OpenFlow applications inside the switch," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 44–51, 2014.
[98] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti, "A survey on the security of stateful SDN data planes," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1701–1725, 3rd Quart., 2017.
[99] D. Kreutz, F. M. V. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proc. ACM HotSDN*, Hong Kong, 2013, pp. 55–60.
[100] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 623–654, 1st Quart., 2016.
[101] I. Alsmadi and D. Xu, "Security of software defined networks: A survey," *Comput. Security*, vol. 53, pp. 79–108, Sep. 2015.
[102] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, "Security in software defined networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2317–2346, 4th Quart., 2015.
[103] S. Ali, V. Sivaraman, A. Radford, and S. Jha, "A survey of securing networks using software defined networking," *IEEE Trans. Rel.*, vol. 64, no. 3, pp. 1086–1097, Sep. 2015.
[104] L. Schehlmann, S. Abt, and H. Baier, "Blessing or curse? Revisiting security aspects of software-defined networking," in *Proc. Workshop Manag. SDN NFV Syst. (ManSDN/NFV)*, Rio de Janeiro, Brazil, 2014, pp. 382–387.
[105] D. B. Rawat and S. R. Reddy, "Software defined networking architecture, security and energy efficiency: A survey," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 325–346, 1st Quart., 2017.
[106] R. Sahay, G. Blanc, Z. Zhang, and H. Debar, "Towards autonomic DDoS mitigation using software defined networking," in *Proc. NDSS Workshop Security Emerg. Netw. Technol.*, 2015, pp. 1–7.
[107] M. Yan, J. Casey, P. Shome, A. Sprintson, and A. Sutton, "AetherFlow: Principled wireless support in SDN," in *Proc. IEEE ICNP*, 2015, pp. 432–437.
[108] I. F. Akyildiz, P. Wang, and S.-C. Lin, "SoftAir: A software defined networking architecture for 5G wireless systems," *Comput. Netw.*, vol. 85, pp. 1–18, Jul. 2015.

AbdelRahman Abdou received the Ph.D. degree in systems and computer engineering from Carleton University in 2015. He is a Post-Doctoral Researcher with the Institute of Information Security, ETH Zurich, Switzerland. He was a Post-Doctoral Fellow with the School of Computer Science, Carleton University, Canada. His research interests include location-aware security, SDN security, system security, authentication, and using Internet measurements to solve problems related to Internet security.

Paul C. van Oorschot is a Professor of computer science with Carleton University, and the Canada Research Chair in authentication and computer security. His research interests include authentication and Internet security. He was the Program Chair of USENIX Security 2008, NDSS 2001–2002, NSPW 2014–2015. He has co-authored a book entitled *Handbook of Applied Cryptography* and a Past Associate Editor of the IEEE TDSC, IEEE TIFS, and ACM TISSEC. He is an ACM Fellow, and a fellow of the Royal Society of Canada.

Tao Wan received the Ph.D. degree in computer science from Carleton University in 2006. He is a Security Researcher with Huawei Canada, Ottawa. He has published in top security conferences, including IEEE Security and Privacy, ACM CCS, USENIX Security, and NDSS. His research interests include security of SDN, cloud computing, 5G, and Internet infrastructure.