An End-Host-Importance-Aware Secure Service-Enabled Hybrid SDN Deployment

Wendi Feng, *Member, IEEE*, Chuanchang Liu, Bo Cheng, *Member, IEEE*, Junliang Chen, and Zhiguo Wan, *Member, IEEE*

***Abstract*—Security is critical to networks, but TCP/IP-based *legacy* networks are difficult to advance new security functions due to the use of costly inflexible hardware devices and error-prone network configurations. Recent literature explores the paradigm of consolidating security services with the forwarding functionality using Software-defined Networking (SDN). Existing *full SDN* deployment, replacing all legacy network devices with SDN devices, is cost-prohibitive. Whereas the *hybrid SDN* that only upgrades partial legacy devices to SDN switches is considered practical. However, the challenge is to minimize threats and deployment expenses simultaneously under heterogeneous end-host businesses that have various importance. In this paper, we study the challenge and propose the *End-host-importance-Aware secure service-enabled hybrid Sdn deplOymeNt (EASON)* problem. We mathematically formulate the EASON problem as an integer programming problem, prove its non-polynomial time complexity, and propose a heuristic algorithm called Algorithm. We conduct rigorous simulations on real-world topologies and traces. Experimental results show that Algorithm achieves comparable security and cost performances to the optimal solution on small topologies. Meanwhile, it is scalable on larger topologies.**

***Index Terms*—Hybrid SDN, network security, network deployment, end-host importance aware.**

I. Introduction

NETWORK security is of paramount importance and has become a top consideration for network operators [1]. In traditional TCP/IP legacy networks, *new* dedicated security devices (e.g., DDoS defenders) are required to enable security services, and complex forwarding rules are necessary to be applied for "detouring" traffic to the security devices. The whole process is costly, tedious, and error-prone. Recent Zero Trust security [2] advocations, distrusting any entities in the network, further raise the bar for network operations in legacy networks. Owing to the complexity, network vendors present new generations of TCP/IP-based network devices with security services consolidated [3], [4]. However, leveraging these benefits must *replace* all old devices, incurring unacceptable costs. Furthermore, security services rely on the provision of hardware vendors, which faces the inability of customization [5] and rapid adaptations of new businesses.

Thanks to Software-Defined Networking (SDN) [6], consolidating security services to the existing infrastructure becomes possible [7], [8]. SDN allows network (functions) provisioned *openly* as software rather than in a *vendors-specific* hardware "boxes" by decoupling network devices' *control* and *data planes* [6]. This concept is later shifted to the decoupling of software and hardware platforms [5] that influences the middlebox community advanced by network function virtualization (NFV) to decouple software (i.e., the control plane) from hardware middleboxes (i.e., the data plane). The recent prevalence of programmable data planes brings even more flexibilities, providing the capability of running network functions (NFs) on programmable SDN switches [9], [10], [11], [12], [13], [14]. Hence, SDN is the "fertilized soil" for security services.

However, employing SDN brings new challenges in practice. Most networks are still *legacy* TCP/IP-based networks, but SDNs, by default, are incompatible with legacy networks. Hence, traditional SDN deployments need to replace all legacy network devices with SDN devices, which results in unacceptable costs. Moreover, the all-new full SDN deployment is onerous because all configurations should be re-implemented from the ground up due to the distinctive architecture. To this end, practical SDN deployments follow an incremental approach [15], [16], [17], [18], [19], [20], [21], in which partial legacy network devices are replaced with SDN devices, and the remaining legacy devices may needless to make modifications [20], [21].

Consequently, we leverage the idea of hybrid SDN and propose *Hybrid SDN as Security Services (HSaSS)* to practically advance security on networks. In HSaSS, security services are directly built on programmable SDN switches [12], [13], and new end-to-end flows are "attracted" to the SDN devices for security inspection and forwarding.[1] When malicious traffic

Manuscript received 3 July 2022; revised 18 September 2022; accepted 19 September 2022. Date of publication 22 September 2022; date of current version 6 July 2023. This work is supported in part by the National Natural Science Foundation of China under grant U21A20468, 61972043, 61921003, Zhejiang Lab under grant 2021PD0AB02, the Fundamental Research Funds for the Central Universities under grant 2020XD-A07-1, Beijing Natural Science Foundation under grant 4214061, and BISTU fund under grant 2022XJJ19. The associate editor coordinating the review of this article and approving it for publication was F. Valenza. *(Corresponding authors: Wendi Feng; Chuanchang Liu.)*

Wendi Feng is with the School of Computer Science, Beijing Information Science and Technology University, Beijing 100192, China (e-mail: wendi.feng@hotmail.com).

Chuanchang Liu, Bo Cheng, and Junliang Chen are with the Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: lcc3265@bupt.edu.cn).

Zhiguo Wan is with Zhejiang Lab, Hangzhou 311122, China.

Digital Object Identifier 10.1109/TNSM.2022.3208695

[1] Specific techniques of implementing the hybrid SDN and security network functions on the SDN controller or the programmable data plane are out of the scope of this paper.

comes, the traversed non-SDN network devices and their attached end-hosts suffer unless the traffic is intercepted by a security service-enabled programmable SDN device.

The challenge is to deploy SDN switches and achieve a secure yet cost-effective hybrid SDN network. We argue that using a single SDN switch may fail to guarantee each end-to-end flow passing an SDN switch and thus lose *flow programmability* (e.g., the ability to dynamically adjust paths). However, the full SDN deployment is cost-prohibitive, while arbitrarily deploying multiple SDN switches in the hybrid SDN network may be insufficient or beyond necessary due to the divergent importance of nodes and attached end-hosts of different networks. Therefore, the hybrid SDN needs a wise deployment decision.

In this paper, we take a step further from our previous work, CLÉ [22]. We explore the optimal hybrid deployment scheme that is i) *end-host-importance-aware:* Important businesses should be preferentially protected. ii) *Secure:* Malicious traffic should be intercepted as early as possible to diminish the impact. iii) *Cost-effective:* A minimal number of SDN switches are used in the hybrid SDN to reduce deployment expenses. We name it the *End-host-importance-Aware secure service-enabled hybrid Sdn deplOymeNt (EASON)* problem. It aims at using a minimum number of SDN switches to intercept malicious traffic expeditiously. We mathematically formulate the EASON problem as an optimization problem and prove the non-deterministic polynomial-time hardness (NP-hardness) complexity. This complexity originates from node significances determined by the SDN deployment and path selections. Due to the complexity, we present an efficient heuristic algorithm called Algorithm to solve the EASON problem. Algorithm employs an iterative approach that selects the *most common anterior nodes* in each flow's *shortest* path to deploy the programmable SDN switches.

The contribution of this paper is three-fold.

- We present the EASON problem in the hybrid SDN deployment and formulate it as an optimization problem.
- We prove the NP-hardness of the EASON problem. Hence, we propose an efficient heuristic algorithm called Algorithm to solve the problem.
- We conduct rigorous simulations on multiple real-world topologies, including Abilene and GÈANT with real-world traffic traces. Experimental results demonstrate the effectiveness of Algorithm.

The remainder of the paper is organized as follows. Section II introduces background knowledge and state-of-the-art. Section III presents our considered attack models and motivates the EASON problem with examples. We mathematically formulate the problem in Section IV. We then prove its NP-hardness and propose the Algorithm algorithm in Section V. In Section VI, we present the simulation setup and results, and Section VII concludes the paper.

II. Background and Related Work

In this section, we first briefly introduce concepts of SDN, Hybrid SDN, and programmable switches. We then present related work. To the best of our knowledge, this paper is the first to study the security-service-enabled programmable SDN switch deployment with the awareness of end-host importance in the hybrid SDN.

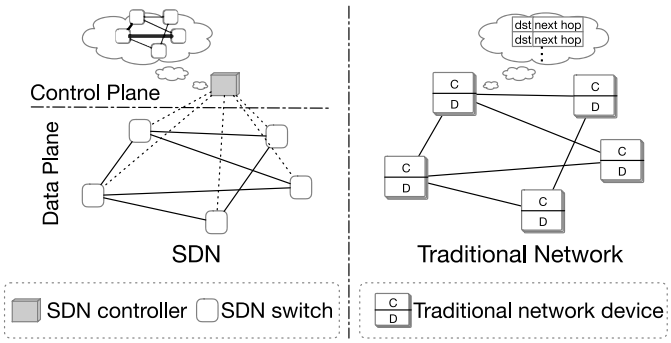


Fig. 1. Software-Defined Network and legacy network.

A. Background

1) Software-Defined Network: SDN [6] *softwarizes* the network and brings *openness* and *flexibility* to network management. Hence, customized network functions[2] can be easily realized in SDN. These are achieved by decoupling the *control plane* from forwarding devices and reserving the *data plane* on the devices. Traditional network devices tie the control and data planes closely together inside the forwarding device with dedicated software, which is hard to adapt to customized services. As shown in Fig. 1, SDN employs a (logically) central controller to retrieve the "global view" of the network instructing the data plane (controlled devices) forwarding packets. Communications between the controller and SDN switches can be either *in-band* or *out-of-band* using SDN protocols (e.g., OpenFlow [23]), where the former transmits the control messages by sharing the links with the data plane, and the latter uses dedicated "controller – switch" links.

Unlike legacy networks that often employ proprietary middlebox devices[3] to perform (security) network functions (NFs), deploying network functions on SDNs can be done by installing SDN applications on the SDN control plane [25], [26], [27], thanks to the flexibility and openness. Hence, security services, e.g., firewalls, intrusion detection systems (IDS), and intrusion protection systems (IPS), can be deployed on the SDN without introducing new devices to the network but greatly simplify network management and lower deployment costs.

2) Hybrid SDN: Deploying SDN in practice follows an incremental deployment strategy due to technical, financial, and business challenges [28], [29], [30], [31], [32]. During the incremental SDN deployment, both legacy and SDN switches present and cooperate to achieve *programmability* and *network management-ability*, making the network a hybrid SDN. A plethora of recent work on hybrid SDN has been proposed, including *hybrid SDN deployment* [16], [17], [18], [19], *new*

[2]Including forwarding and routing.

[3]Although recent routers [3], [4] have added support for integrated NFs, production networks still leverage dedicated middleboxes to conduct functionalities [24].

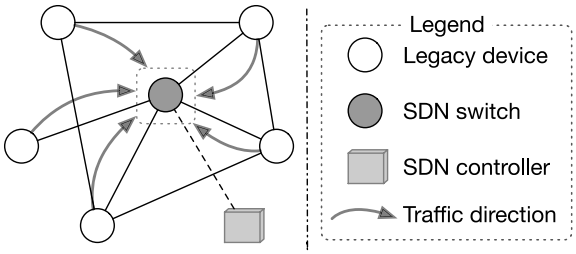


Fig. 2. Hybrid SDN demonstration.

architectures for hybrid SDNs [33], [34], [35], [36], *hybrid SDN management* studies [17], [37], and *traffic engineering* aided by hybrid SDN [38], [39], [40]. Although many security-related arts have been put forth (detailed in Section II-B), they neglect that hybrid SDN can enhance network security with security services deployed on programmable switches.

Our previous work [22] draws the blueprint for utilizing programmable switches to deploy security services. Its foundation is to attract flows to the SDN switches for security analysis. The idea of attracting flows to the SDN switches is borrowed from recent work [20], [21] that employs a "plug-and-play" approach, in which the SDN switches and controller can "cooperate" with the unmodified legacy devices by using the address resolution protocol (ARP). More specifically, the SDN controller generates *gratuitous ARP* messages and leverages the controlled SDN switches to inject the messages to the network and populate ARP cache tables on end-hosts with "fictitious" or "illusory" media access control (MAC) addresses. Hence, the legacy devices can update their forwarding table accordingly and send packets to the SDN switches for *programmability*. Then, the SDN switches can forward packets to the proper destination. As shown in Fig. 2, flows are "attracted" to the SDN-capable devices without modifying the legacy devices like the magnetic force. This architecture is also resilient to failures. For example, if one SDN switch fails, legacy network devices will update the forwarding table with the newly received gratuitous ARP messages from "live" SDN devices to bypass the failed device, and if some flows fail to pass an SDN switch, these flows will be forwarded directly by legacy devices. Besides, when all SDN devices fail, the network falls back to the legacy network. The specific mechanism is out of the scope of this paper, and please refer to [20], [21] for more details.

B. Related Work

Our work is built on the foundation of the hybrid SDN and programmable data plane, where security NFs can be implemented on the programmable SDN switch. Hence, in this subsection, we brief necessary related work for running NFs on the programmable switches, security device placement, and network planning.

1) From NFV to Programmable Switches: The trend of using general-purpose commodity computing platforms to implement NFs, called Network Function Virtualization (NFV), raises attention from both academic and industrial communities [41]. Research has been carried out on improving manageability [42], [43] and performance [44], [45]. However, this pure software approach often suffers from impaired performances (e.g., can only achieve as high as several hundred gigabits per second), and a significant amount of computation resources is "eaten" due to the lack of dedicated packet processors [46]. Notwithstanding simple NFs can be directly applied to SDN switches, complex ones may involve deep participation of the SDN controller, which can again incur significant "controller – switch" overheads [47]. Thanks to programmable switches [48], implementing high-performance complex functions directly on the data plane becomes possible. The presence of the P4 language [49] further greatly simplifies programming switches. With these advances, new paradigms are exploding. For example, stateful network functions [8], in-network key-value stores [50], load-balancing [51], [52], parallel computing devices [24], [53], [54], cooperating with hosts via remote direct memory access (RDMA) [55], in-network telemetry [56], [57], and canary testing [58], [59]. These innovations bring unimaginable potential to the next-generation network and security functions.

2) Security Services on Programmable Networks: A profusion of recent studies has been proposed to implement security functions on programmable switches. Vörös and Kiss [11] present a firewall implementation with protocol, port filtering, and flood detections using P4. P4ID [12] and P4DDPI [60] deploy the IDS on programmable switches, and P4ID can achieve up to 75% of the performance reduction compared with traditional host-based IDS. ElasticSketch [13], SpreadSketch [61], HashPipe [62], and pHeavy [63] profile the traffic and find bursts (i.e., gigantic traffic in a short duration) or spreader (i.e., hosts initiate a large number of connections) from the traffic to identify malicious behaviors. Although these researches are efficient, they focus on a single programmable switch security service implementation without considering the impact of network-wide deployments.

3) Security Middlebox Placement: Our work studies placing security services (functions) on different programmable SDN switches. Hence, we survey related security middlebox placement work. ShieldBox [64] considers creating security services on untrusted commodity server platforms with hardware enclaves. Smith and Bhattacharya [65] present the concept of firewall cascade that aims at maximizing security protection with optimized cost, which leverages a chain of firewalls and places them between the potential attack point and network node that has sensitive data to reduce the protection inability with one firewall. Lee *et al.* [66] propose a firewall deployment scheme in the data center network, which is subject to the bandwidth consumption of links. Bouet *et al.* [67] deploy vDPI (virtual Deep Packet Inspection) network functions on a dedicated programmable switch in full SDNs.

However, all these studies fail to consider that attacks (e.g., DoS attacks) can impact the network nodes along the propagation path resulting in the compromise of the network.

4) Network Security Planning: A plethora of network planning work has been proposed. Hence, we only survey the most relevant ones. NETSPA [68] is a tool to display possible attack sequences of attackers. It generates worst-case

attack graphs using a forward-chaining depth-first search of attack space. Lv *et al.* [69] address physical-layer attacks of the SDN-based optical network control plane. Its goal is to minimize the network service disruption under physical-layer attacks, which can increase the resiliency of the SDN network, similar to our previous work [70], by remapping attacked controller controlled switches to live controllers. However, these researches differ from ours for the following reasons. i) *Different system settings:* Our work employs the concept of security service-enabled programmable SDN switches in the hybrid SDN scenario. ii) *Different attack models:* Our attack model assumes attacks can be mitigated by the programmable SDN switch (detailed in Section III), whereas existing work profiles the attack propagation in a local area network or increases network resiliency by rearranging the "switch – controller" mapping.

III. Attack Model and Motivation

In this section, we first present our considered attack model. Based on the attack model, we employ a set of simple examples to show the security issues and how the proposed End-host-importance-Aware secure service-enabled hybrid Sdn deplOymeNt can intelligently solve them.

A. Attack Model

The attack model is based on the observation that attackers can conduct attacks by generating malicious traffic and injecting them into the network. When malicious traffic propagates, the hosts connecting to each traversed node will be highly vulnerable if no security services are deployed on the node. Many attacks in the real world have these characteristics. We demonstrate a Denial-of-Service (DoS) attack example later in the subsection. In our setting, when the malicious traffic traverse through a programmable SDN switch, the traffic is terminated because security services are deployed on the programmable SDN switch. Hence, all flows should pass at least one programmable SDN switch to analyze the traffic and guarantee the flow programmability.

In practice, each node may connect to distinct types of hosts that provide various services. For example, one node is more important than another when it connects to many servers that serve millions of users while the other node only has personal devices connected. Consequently, our attack model also accounts for the diversity of network node importance. In this paper, we only consider peer-to-peer flows between two nodes, which is the aggregation of all end-to-end flows over the two nodes. We use "flow" to represent peer-to-peer flows unless pointed out.

Example: DoS Attack: DoS [71] attacks send an enormous amount of traffic to compromise the target. It makes hosts less responsive, exhausts resources, and disables services. SYN-flooding [72] is the easiest way to conduct DoS, in which the attacker sends a massive amount of TCP SYN packets to request connection establishments with a host but neglects all subsequent SYN-ACK packets. The target end-host and the nodes along the path will consume enormous computation and memory resources to repeatedly re-generate and re-transmit the SYN-ACK packet for all SYN packets. Simultaneously, a significant amount of traffic "jams" the network, congests the links, and influences the communications of other hosts.

B. Motivation

This subsection motivates the EASON problem with examples. We first show that a legacy network with six nodes, and each node connects to many end-hosts. Due to the lack of security enhancements, end-hosts can be compromised as attacks come. Later, we show that introducing security-service-enabled programmable SDN switches to the network for traffic analysis and malicious traffic interception. The challenge is that using only one such device may fail to retain the programmability for all flows, but deploying all network nodes using security-service-enabled programmable SDN switches and converting the legacy network into a fully security-service-enabled SDN is cost-prohibitive. Besides, arbitrarily deploying multiple SDN switches can also fail to ensure security protection and flow programmability. Hence, we need to judiciously select the nodes to deploy the programmable SDN switches.[4]

1) Insecure Legacy Network and Blessing of Programmable SDN Switches: Fig. 3a depicts a legacy network with six nodes, and each node connects to many end-hosts.[5] The network does not deploy any dedicated security network functions. Thus, as shown in Fig. 3b, when malicious traffic (**red bold line**) is injected into the network, all nodes and end-hosts along the way are influenced (red border).

Owing to the *openness* and *flexibility* of SDN, deploying security services on the SDN (see Section II) is straightforward. After deploying security network functions on programmable SDN switches, the data plane can conduct security analyses for the packets (e.g., DPI, IDS) and mitigate malicious traffic. As shown in Fig. 3c, the network becomes a hybrid SDN after introducing the security-service-enabled programmable SDN switch. Flows are sent to the programmable SDN switch for packet forwarding (or relaying) and security analysis, and the security is thus enhanced.

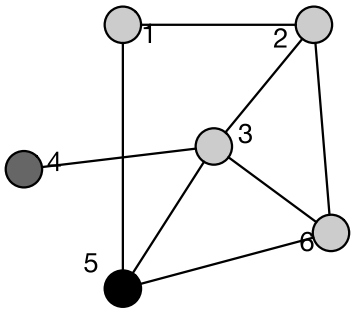
2) Curses of Upgrading a Single Node and Full SDN: Notwithstanding the benefits, using a single programmable SDN switch to process all flows in the network can overload the switch and can thus result in poor network conditions. This will further impact end-hosts and make the network fail to satisfy the Service Level Objective (SLO). Moreover, when a bridge[6] of a network does not have a programmable SDN switch, the programmability of flows between the bridge nodes can fail to be satisfied. Therefore, more security-service-enabled SDN switches are required.

However, it is unnecessary to completely replace all legacy devices with programmable SDN switches (see Fig. 3d). Although a full programmable SDN device replacement

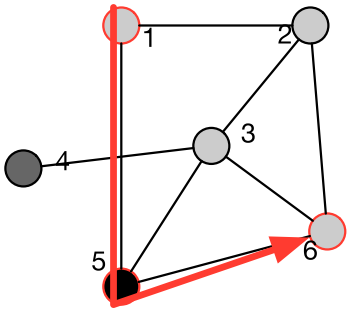
[4]We use "programmable SDN switches" and "SDN switches" interchangeably in this paper.

[5]We use the total rate on this node to represent its significance (detailed in Section IV-B1). For expressiveness, we employ colors to indicate the importance of the node. The darker the color is, and the more important the node is.

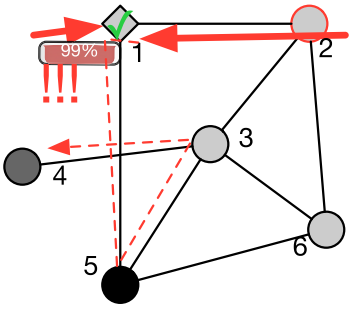
[6]For example, the link between node 4 and the node 3 in Fig. 3a is a (mathematical) bridge. If SDN switches are not deployed at either of these nodes, flows between them are not programmable.



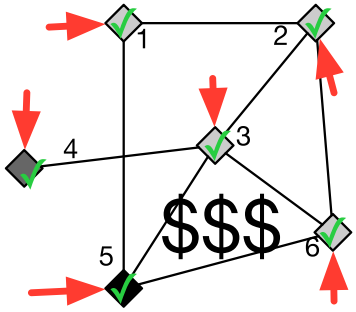
(a) A six-node simple legacy network. Each node connects many hosts, and the color deepness of a node indicates the importance of end-hosts attached to the node. **The darker the color is, and the more end-hosts are connected.** SDN controllers are **omitted** in Figs. 3a-3f.



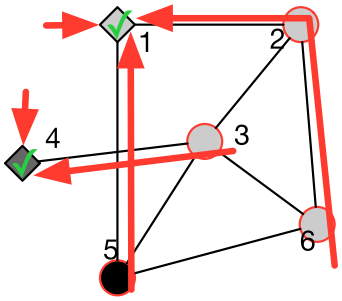
(b) Malicious traffic (red **bold** line) is injected at one node and targeting another node in the network. Nodes on the path of the traffic propagation alone with their connected end-hosts (red border) suffer due to the lack of security protection.



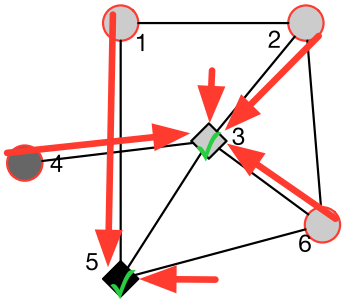
(c) Deploying only a single security-service-enabled programmable SDN switch can mitigate malicious traffic. However, flows are detoured to the SDN switch. Besides, some flows may lose programmability.



(d) The full SDN deployment allows security services deployed on programmable switches to achieve the best performance, but this deployment can be cost-prohibitive.



(e) Arbitrarily deploying multiple security-service-enabled programmable SDN switches may be insufficient to protect the network. Besides, The malicious traffic passes more nodes before being terminited and thus may compromise more hosts.



(f) The EASON deployment achieves the best security enhancement (shortest average malicious traffic propagation and a minimum number of compromised hosts) at the lowest cost (using a minimal number of programmable SDN switches).

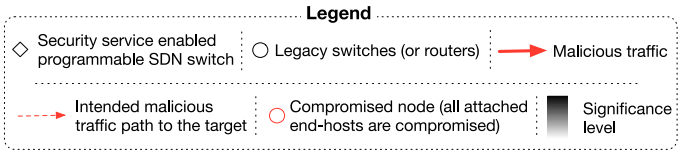


Fig. 3. Motivation examples. The number by each node is its ID number, and the figures only show partial flow cases.

can protect the network and forward packets at the best performance, the cost of replacing all legacy devices can be prohibitive. Moreover, the process of re-configure the all-new devices can be tedious and error-prone. Businesses may thus be impacted. Hence, the full SDN deployment is not readily available for production environments.

3) Arbitrarily Upgrading Multiple Nodes: Arbitrarily selecting some nodes and upgrading them to a security-service-enabled programmable SDN device may not be enough. If the chosen nodes are "improper", 1) programmable SDN switches can be overloaded and 2) traffic can be routed over a long detoured path to pass a programmable SDN switch, which results in malicious traffic being propagated "throughout" the network, impacting (compromises) the network end-hosts (see Fig. 3e).

4) Secure Yet Cost-Effective Deployment With End-Host Importance Awareness: When judiciously selecting "proper" nodes in the network topology, as shown in Fig. 3f, the network can mitigate malicious traffic as early as possible and reduce the number of impacted (compromised) end-hosts. Besides, the number of SDN switches used is minimal without overloading the network. We call it the *End-host-importance-Aware secure service-enabled hybrid Sdn deplOyMeNt (EASON)*. When a path is selected, traffic can only pass through that path. Other unused links can provide resiliency, as we have mentioned in Section II-A2. The rest of the paper systematically explores the method of finding the deployment.

IV. Formulation

In this section, we formally formulate the EASON problem. We first mathematically describe the network structure and then present *path-traversal-based* network security metrics. Next, we introduce constraints and the objective function, and finally formulate EASON as an *integer programming* problem.

A. System Description

This subsection mathematically formulates the network. For quick referencing, all used notations are shown in Table I. A network contains network devices (nodes) and links, which can be represented as a graph $G = (V, E)$, where $V = \{v_1, v_2, \ldots\}$

TABLE I
NOTATION DESCRIPTIONS

Notations	Descriptions
$\bigvee$ or $\vee$	The logical OR operation.
$\bigwedge$ or $\wedge$	The logical AND operation.
$\neg$	The logical NOT operation.
$\lvert\cdot\rvert$	The number of elements in a set.
$\bigcup$	The set UNION operation.
$(\cdot)^\intercal$	The matrix transpose operation.
iff.	If and only if.
V	The network node set. $V = \{v_1, v_2, \dots\}$.
E	The network link set. $E = \{e_1, e_2, \dots\}$.
P	The path set. $P = \{\overrightarrow{p_1}, \overrightarrow{p_2}, \dots\}$.
F	The flow set. $F = \{f_1, f_2, \dots\}$.
$load(f_j)$	The load of flow f_j.
S	The node significance vector. $S = \{s_1, s_2, \dots, s_{\lvert V\rvert}\}$.
C^{node}	The node processing capability set. $C^{node} = \{c_1^{node}, c_2^{node}, \dots c_{\lvert V\rvert}^{node}\}$.
C^{link}	The link capacity set. $C^{link} = \{c_1^{link}, c_2^{link}, \dots c_{\lvert E\rvert}^{link}\}$.
A	The relationship matrix of links and nodes. $A = \{a_{ki}\}$.
H	The relationship matrix of nodes and paths. $H = \{h_{im}\}$.
B'	The relationship between links and flows. $B' = \{b'_{kj}\}$.
X	The mapping relationship matrix between node v_i and programmable SDN switches. $X = \{x_i \mid x_i = 1, \forall\ v_i$ is an SDN switch, and $x_i = 0, \forall\ v_i$ is a legacy device$\}$.
Y	The "flow – path" relationship matrix. $Y = \{y_{jm} \mid y_{jm} \in \{0, 1\}\}$. If path $\overrightarrow{p_m}$ is selected for flow f_j we have $y_{jm} = 1$; and 0 otherwise.
$\overrightarrow{p_j}'$	The node vector of propagated path used by flow f_j. Notation $\overrightarrow{\cdot}$ represents the sequence.
"$\dots \to \dots$"	A path.
M	A traffic matrix.

is the set of nodes, and $E = \{e_1, e_2, \dots\}$ is the set of links. Each node can be (deployed with) either an SDN switch or a legacy network device (i.e., a legacy switch or router). Node v_i represents the i^{th} node. Let $P = \{\overrightarrow{p_1}, \overrightarrow{p_2}, \dots\}$ be the set of paths, and let $F = \{f_1, f_2, \dots\}$ be the set of flows. Each flow f_j has many path candidates and is represented by $Y = \{y_{jm}\}$, where $y_{jm} = 1$ denotes flow f_j uses path $\overrightarrow{p_m}$, and otherwise $y_{jm} = 0$. Let $A = \{a_{ki}\}$ be the relationship matrix of links $\{e_k\}$ and nodes $\{v_i\}$. $a_{ki} = 1$ if node v_i in link e_k, and 0 otherwise. The load of each flow f_j is written as $load(f_j)$. Let $X = \{x_i\}$ indicate whether node v_i is an SDN switch, where $x_i = 1$ represents that node v_i is deployed with an SDN device, and otherwise $x_i = 0$. Let $H = \{h_{im} \in \{0, 1\}\}$ be the relationship matrix of nodes and paths. Let $C^{node} = \{c_1^{node}, c_2^{node}, \dots\}$ be the set of switch processing capabilities, where c_i^{node} denotes the processing capability of node v_i. Let $C^{link} = \{c_1^{link}, c_2^{link}, \dots\}$ be the link capacity set, where c_k^{link} denotes the capacity of link e_k.

B. Metrics

In this subsection, we present three *path-traversal-based metrics*. The *node significance* metric indicates the significance of the node determined by its connected services. The *propagated path* metric describes all nodes passed by a flow from its entry node (from which a flow enters the network) to the first node deployed with the security-service-enabled programmable SDN device. Lastly, to evaluate the overall security performance, we present the *compromised network significance* metric that accumulates the node significances of compromised network nodes across all flows.

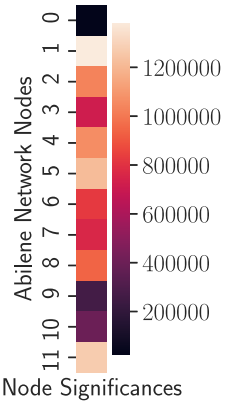


Fig. 4. The Node significance example. Calculated from real-world traces [73] of the Abilene [74] topology.

1) Node Significance: We define the significance of a node as the total rates (traffic transmission rate) passing the node. A node (i.e., a router or switch) usually connects many devices, and different types of devices have distinct usages of various importance. For example, a personal device only matters its owner, while a server that runs tens or hundreds of services (e.g., social networking and financing services) and serves a large group of people matters a massive amount of users. When the node only connects to personal devices, the overall traffic rates passing through the node are usually small, but when it connects to servers running many services, traffic can be gigantic. Hence, the metric can be formally written as

$$s_i = \sum_{f_j \in F} load(f_j)\, o_{ij}, \tag{1}$$

where $o_{ij} = 1$ represents node v_i in flow f_j's selected path, and 0 otherwise. Hence, matrix $O = \{o_{ij}\}$ is calculated as

$$O = HY^\intercal. \tag{2}$$

Therefore, Equation (1) can be reformulated as

$$s_i = \sum_{f_j \in F} load(f_j) \sum_{m=1}^{|P|} h_{im} y_{jm}. \tag{3}$$

2) Propagated Path: The goal is to intercept malicious traffic *as early as possible*. Hence, we need to describe the status of a flow that has propagated on the network. We represent it with a sequence of nodes from the *entry node* to the first *security service node* (i.e., the programmable SDN switch) on the flow path. It can be formally represented as

$$\begin{aligned}\overrightarrow{p_j}' &= \left\{ v \mid v \in \bigvee_{\overrightarrow{p_m} \in P} \overrightarrow{p_m} \wedge y'_{mj},\ \text{until first } x_v = 1 \right\} \\ &= \left\{ v \mid v \in \bigcup_{v_i \in \bigvee\limits_{\overrightarrow{p_m} \in P} \overrightarrow{p_m} \wedge y'_{mj}} v_i \left(\neg \bigvee_{i_1=1}^{i} x_{i_1} \right) \right\}, \end{aligned} \tag{4}$$

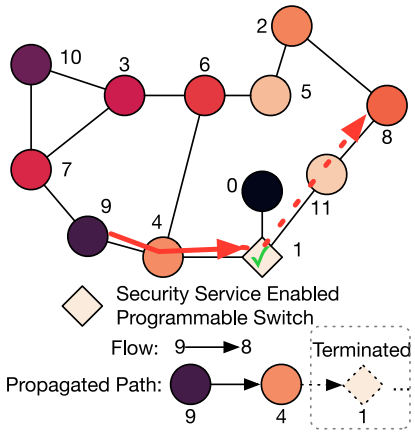


Fig. 5. A propagated path example on Abilene. Malicious traffic is terminated at the security-service-enabled programmable SDN switch. Colors indicate the node significances shown in Fig. 4.

where y'_{mj} is derived from y_{mj}, and $y'_{mj} = \varnothing$ if $y_{mj} = 0$, and otherwise $y'_{mj} = \overrightarrow{p_m}$. Expression $\neg \bigvee_{i_1=1}^{i} x_{i_1}$ indicates whether an SDN switch has been passed at the i^{th} node along the path. As long as v_i is an SDN switch, all succeeding nodes cannot change the value of $\bigvee_{i_1=1}^{i} x_{i_1}$ that will always be 1. As shown in Fig. 5, the selected path for flow $9 \rightarrow 8$ is "$9 \rightarrow 4 \rightarrow 1 \rightarrow 11 \rightarrow 8$". Since node 1 is a programmable SDN switch and has security service, malicious traffic will be dropped by node 1. Thus, the propagated path is "$9 \rightarrow 4$".

3) Compromised Network Significance: With the above metrics, we now consider the impact of malicious traffic over the whole network and propose the compromised network significance metric under security-service-enabled programmable SDN switches deployment scheme X. We model the compromised network significance as the summation of all compromised node significances across all flows. It can be written as

$$\begin{aligned} r &= \sum_{f_j \in F} \sum_{v_i \in \overrightarrow{p_j}'} s_i \\ &= \sum_{f_j \in F} \sum_{v_i \in \bigvee\limits_{\overrightarrow{p_m} \in P} \overrightarrow{p_m} \wedge y'_{mj}} s_i \left(\neg \bigvee_{i_1=1}^{i} x_{i_1} \right). \end{aligned} \tag{5}$$

C. Constraints

1) Hybrid SDN Constraint: The total number of SDN devices needed should be less than the total number of nodes in the network. Besides, the network should at least replace one legacy device with a programmable SDN device to leverage programmability and enhance security. We mathematically formulate this constraint as

$$1 \leq \sum_{v_i \in V} x_i < |V|, x_i \in \{0, 1\}. \tag{6}$$

2) Path Programmable Constraint: Each flow should pass at least one programmable device (i.e., a programmable SDN switch) to leverage the programmability and utilize security services. Hence, for each flow $f_j \in F$, we have

$$\sum_{v_i \in V} \left(x_i \sum_{\overrightarrow{p_m} \in P} h_{im} y_{jm} \right) \geq 1. \tag{7}$$

3) Programmable Device Capability Constraint: As the network starts running, the programmable SDN switch starts processing traffic and intercepts malicious traffic. However, devices have limited processing capabilities, and the traffic processing demand should not exceed their capability. Therefore, for each node $v_i \in V$, this is formulated as

$$\sum_{f_j \in F} \left(load(f_j) \sum_{\overrightarrow{p_m} \in P} h_{im} y_{jm} \right) \leq c_i^{node}. \tag{8}$$

4) Link Capacity Constraint: Similarly, each link has a capacity limit, and total traffic rates that pass through a link cannot exceed the link's capacity.

For brevity, we first formulate the relationship between links and flows as

$$B = A(HY^{\intercal}), \tag{9}$$

where H is the node – path relationship matrix. The value of each $b_{kj} \in B$ represents the number of nodes of a link on the path (e.g., the number of nodes of link $\langle u, v \rangle$ on path "$\cdots \rightarrow u \rightarrow v \cdots$" is 2). Therefore, the value of each b_{kj} can be "0, 1, 2". $b_{kj} = 2$ *iff.* link k is in flow f_j. Hence, each element in the "link – flow" relationship matrix $B' = \{b'_{kj}\}$ can be formulated as

$$b'_{kj} = \max\{0, b_{kj} - 1\}. \tag{10}$$

Therefore, for each link $e_k \in E$, the link capacity constraint is formulated as

$$\sum_{f_j \in F} load(f_j) b'_{kj} \leq c_k^{link}. \tag{11}$$

D. Objective Function

Merely considering security using the network compromise ratio metric may result in a prohibitive deployment cost. Hence, we consider two objectives in the EASON problem: the *compromise ratio* and *number of SDN switches used*. The first objective is the key to enhance security by intercepting malicious traffic, while the latter reduces expenses for the hybrid SDN deployment.

The first objective is represented as

$$\begin{aligned} obj_1 &= r \\ &= \sum_{f_j \in F} \sum_{v_i \in \bigvee\limits_{\overrightarrow{p_m} \in P} \overrightarrow{p_m} \wedge y'_{mj}} s_i \left(\neg \bigvee_{i_1=1}^{i} x_{i_1} \right), \end{aligned} \tag{12}$$

and the second objective is written as

$$obj_2 = \sum_i x_i. \tag{13}$$

The objective function is a weighted summation of obj_1 and obj_2 as

$$\begin{aligned} obj &= \lambda obj_1 + obj_2 \\ &= \lambda \sum_{f_j \in F} \sum_{v_i \in \bigvee\limits_{\overrightarrow{p_m} \in P} \overrightarrow{p_m} \wedge y'_{mj}} s_i \left(\neg \bigvee_{i_1=1}^{i} x_{i_1} \right) + \sum_{v_i \in V} x_i. \end{aligned} \tag{14}$$

where λ $(0 \leq \lambda \leq 1)$ is a constant to indicate the relationship between the two objectives. It is a fixed value in our setting and normalized in the simulation based on topology sizes.

E. Problem Formulation

As mentioned in previous subsections, the goal of the EASON problem is to minimize the compromised network significance *r* by using the minimal number of SDN switches $\sum_{v_i \in V} x_i$. The problem is formulated as

$$\begin{aligned} \min_{x,y} \quad & \left\{ \lambda \sum_{f_j \in F} \sum_{v_i \in \bigvee\limits_{\overrightarrow{p_m} \in P} \overrightarrow{p_m} \wedge y'_{mj}} s_i \left(\neg \bigvee_{i_1=1}^{i} x_{i_1} \right) + \sum_{v_i \in V} x_i \right\} \\ \text{s.t.} \quad & (6), (7), (8), (11), \\ & x_i \in \{0,1\}, y_{jm} \in \{0,1\}, \\ & \forall i \in [1, |V|], \forall j \in [1, |F|], \forall m \in [1, |P|], \\ & v \in V, s \in S, \end{aligned} \tag{P}$$

where $\{s_i\}$ are calculated constants. The designed variables $\{x_i\}$ and $\{y_{ij}\}$ are binary integers. Hence, the formulated Problem (P) is an *Integer Programming* problem.

V. Solution

In this section, we analyze the complexity of the EASON problem and prove its NP-hardness. Due to its complexity, we present an efficient heuristic algorithm called Algorithm.

A. Complexity Analysis

In this subsection, we prove the NP-hardness of the EASON problem by reducing a special case to the *Set Cover Problem (SCP)* [75].

Theorem 1: For a special case when the relationship variable $\lambda = 0$, the End-host-importance-Aware secure service-enabled hybrid Sdn deplOymeNt problem is NP-hard.

Proof: We first introduce the SCP problem. The SCP problem describes that given a set $E = \{e_1, e_2, \ldots, e_m\}$ and its non-empty power set $\mathcal{P}(E) \backslash \emptyset = \{E_1, E_2, \ldots, E_n\}$. Each subset E_j has a weight c_j, where $j \in [1, n]$. A set cover is a collection $T \subseteq \{1, \ldots, n\}$ such that $\cup_{j \in T} E_j = E$. The difference between *T* and $\mathcal{P}(E)$ is that *T* is a collection of subset indices from $\mathcal{P}(E)$, whereas $\mathcal{P}(E)$ contains all subsets of *E*. Let $x_j = 1$ represents the subset E_j is selected, and $x_j = 0$ otherwise. Let $y_{ij} = 1$ denotes $e_i \in E_j$, and 0 otherwise. The SCP problem finds a minimum weight set cover, which is

$$\min_{T} \left\{ \sum_{j \in [1,n]} c_j x_j \right\}, \tag{15}$$

$$\text{s.t.} \bigcup_{j \in T} E_j = E, \tag{16}$$

$$1 \leq \sum_{j \in [1,n]} x_j \leq n, \tag{17}$$

$$\sum_{j} y_{ij} = 1. \tag{18}$$

It has been proved that the SCP problem is NP-hard [75].

We then prove for the aforementioned special case, Problem (P) and the SCP problem are equivalent. Given the special case in Theorem 1, obj_1 can be eliminated. The Objective function of Problem (P) can be reformulated as

$$obj = \sum_{i} x_i. \tag{19}$$

Besides, the obj_1 related Constraints (7), (8), (11) are eliminated.

Hence, Problem (P) can be reformulated as

$$\begin{aligned} \min_{x} \quad & \sum_{i} x_i, \\ \text{s.t.} \quad & 1 \leq \sum_{v_i \in V} x_i \leq |V|, \\ & \sum_{\overrightarrow{p_m} \in P} y_{jm} = 1. \end{aligned} \tag{P'}$$

Problem (P') aims to minimize the number of SDN switches. We can regard node v_i and flow f_j in Problem (P') as element e_i and subset E_j. Under such a construction, we can prove that there exists an optimal solution using the minimum number of programmable SDN switches to serve the network, *iff.* the SCP problem has an optimal solution that has the minimum number of subsets to cover set *E*. This problem construction can be achieved in a polynomial time, but finding the optimal solution to Problem (P') is NP-hard due to the NP-hardness of the SCP problem. ■

Hence, Problem **(P') is a special case of the SCP problem. Consequently, we can conclude that**

Theorem 2: The EASON problem is NP-hard.

B. The Algorithm Algorithm

The complexity of the EASON problem results from that the significance of each node is calculated from the SDN switch arrangement *X* and the flow path assignment *Y*. Due to the complexity, we present an efficient heuristic solution.

As depicted in Algorithm 1, the intelligence behind the Algorithm algorithm is to find the *most common anterior nodes* in the shortest paths of all flows. This is a three-step approach detailed as follows.

1) *Finding the shortest paths:* Algorithm finds the shortest path(s) for all flows, in which a flow is defined as traffic from one node *a* to *another* node *b*. We omit flows whose source and destination nodes are the same. When a flow has multiple shortest paths, the first one in the calculated shortest paths list is selected for the flow.

2) *Calculating the node significances:* After selecting the shortest paths for flows, each node's significance is calculated using (3), which accumulates traffic rates of all flows passing the node.

3) *Iteratively choosing common significant nodes:* As shown in Fig. 6, this step iteratively chooses the *most common anterior nodes* by recording the occurrences and the significance of the idx^{th} node in each flow's selected paths. A set of all idx^{th} nodes thus is generated. Then, the significant ratio is calculated

Algorithm 1: The Algorithm Algorithm

Input: $G = (V, E)$: The topology;
Input: M: The traffic matrix.
Output: X : The SDN switch deployment.

1 $X = \{0, \ldots, 0\}$; $P \leftarrow \varnothing$; $obj = \infty$; satisfy= 0;
2 **for** $f \in F$ **do**
3 $paths \leftarrow$ shortest_paths_all(G, f);
4 apply_path$(G,\ f,\ paths[0])$;
5 $P \leftarrow P \cup paths[0]$;
6 apply_traffic$(G,\ M)$;
7 $S \leftarrow$get_node_significance(G);
8 cnt $\leftarrow \left\{v_1 : 0, \ldots, v_{|V|} : 0\right\}$;
/* Iteratively select nodes for deploying SDN switches. */
9 **for** $idx \in [1, |V|)$ **do**
10 **for** $p \in P$ **do**
11 $v \leftarrow$ p$[idx]$;
/* End-host importance aware with node sig. ratio. */
12 cnt$[v] \leftarrow$ cnt$[v] + \frac{s_v}{\sum_{s_i \in S} s_i}$;
13 $v \leftarrow$ get_max_count_node(cnt);
14 $X' \leftarrow X$; $x'_v \leftarrow 1$;
15 status$\leftarrow$re_apply_paths(G, X');
16 apply_traffic(G, M);
17 $S \leftarrow$get_node_significance(G);
18 $obj_{new} \leftarrow$ cal_obj_value(G, X');
19 **if** $satisfy = 0$ **then**
20 **if** $status = 1$ **then**
21 satisfy $\leftarrow 1$;
22 **if** *!dev_contraints*(G, X') *and* *!link_contraints*(G, X') **then**
23 satisfy $\leftarrow 0$;
24 **continue**;
25 cnt.remove(v); $X \leftarrow X'$;$obj \leftarrow obj_{new}$;
26 **continue**;
27 **if** $satisfy = 1$ **then**
28 **if** $obj_{new} \geq obj$ **then**
29 **continue**;
30 cnt.remove(v); $X \leftarrow X'$; $obj \leftarrow obj_{new}$;
31 **return** X

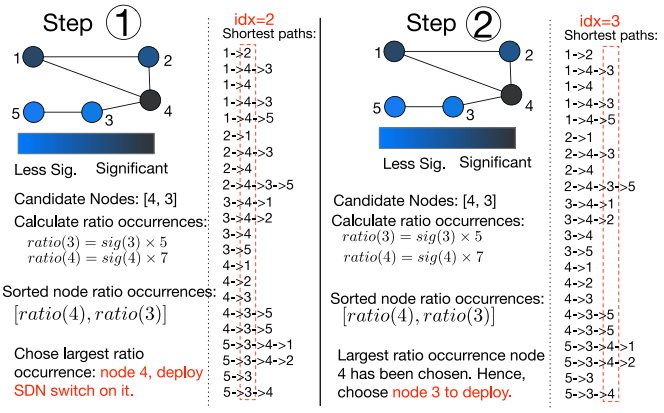


Fig. 6. The Algorithm algorithm demonstration. Note that the last node on each path is not counted on.

for each node in the set, and the one with the highest significant ratio is chosen to deploy the programmable SDN switch. This step contains three procedures. i) *Variable initialization.* Set the current objective value to $obj = \infty$, and set the current node index to $idx = 1$. The variable satisfy here is a flag indicating if there has already been a deployment scheme satisfies the constraints. ii) *Significance calculation.* Count the number of occurrences by accumulating the significances of the idx^{th} nodes used in the shortest paths of all flows and deploy a programmable SDN switch at the node with the maximum count. iii) *Objective calculation.* Calculate the objective value for the deployment as obj_{new}, and compare it with obj. If $obj_{new} < obj$, the newly selected node is then deployed with a programmable SDN switch, and the current node index moves forward to the next node. The algorithm then goes back to "Procedure 31". Otherwise, the algorithm stops. Whenever a new X is calculated, the corresponding path applied to each flow is re-arranged because each flow should pass at least one programmable SDN switch for flow analysis and programmability satisfaction. The path re-arrange procedure is detailed in Algorithm 2.

In Algorithm 1, Lines 1-7 initialize the network, find the shortest path(s) for each flow, apply the traffic matrix, and calculate the node significances. The time complexity of this subprocedure is $O(|F|)$, which can be represented as $O(|V|^2)$. To jointly consider node occurrences and significance, we introduce *significance ratio*.[7] Lines 9-30 iteratively calculate the significance ratio of each node in the idx^{th} occurrences of each path, in which Lines 10-12 gather the idx^{th} node in each path and accumulate their node significance. This subprocedure has a $O(|V||E|)$ time complexity, where the path set can be obtained in linear time complexity $O(|E|)$ [76]. Line 13 finds the node with the max significance "occurrence" (i.e., significance ratio) that becomes a candidate for deploying an SDN switch. The time complexity is $O(\log|V|)$. Then, the network adjusts the paths based on the new SDN switch deployment in Lines 15-17. Whenever a new node is selected as the candidate, Algorithm first tries to re-assign flow paths, re-apply traffic, and re-calculate the node significances. A *new* objective value is then calculated based on the node candidate to compare with the current objective value. If the new value is smaller than the old one, a programmable SDN switch is deployed at the candidate. Next, Algorithm keeps testing each index until the end. The most complex part of this subprocedure is the re_apply_paths() function detailed in Algorithm 2, and its time complexity is $O(|F|)$. Consequently, the overall time complexity for Algorithm is $O(|V|^3)$, which is polynomial complex.

[7]The ratio of a node's significance to the summation of significances of all nodes (see Line 12 in Algorithm 1).

Algorithm 2: `re_apply_paths(`G, X`)`

Input: $G = (V, E)$: The topology; X: The SDN switch deployment.
Output: *flag*: the status of flow programmability satisfaction.

1 $V^{SDN} \leftarrow \varnothing$;
`/* Get current nodes that deployed with SDN switches. */`
2 **for** $i \in |V|$ **do**
3 **if** $x_i = 1$ **then**
4 $V^{SDN} \leftarrow V^{SDN} \cup v_i$;
5 $\overrightarrow{flag} \leftarrow \overrightarrow{0}$;
6 **for** $f \in F$ **do**
7 $p_{cur} \leftarrow$ `get_cur_paths(`G, f`)`;
8 $p_{old} \leftarrow p_{cur}$;
9 **if** $V^{SDN} \cap p_{cur} \neq \varnothing$ **then**
10 **continue**;
`/* If having more than 5 shortest paths, we only reserve 5. */`
11 $P_s \leftarrow$`paths_shortest_5(`G, f`)`;
12 **for** $p \in P_s \setminus p_{cur}$ **do**
13 **if** $V^{SDN} \cap p \neq \varnothing$ **then**
14 `apply_path(`G, f, p`)`;
15 $flag_f \leftarrow 1$;
16 **break**;
17 **if** $(flag_f = 0) \wedge p$ *is the last shortest path.* **then**
`/* Detour the traffic. */`
18 $p \leftarrow$ get_cur_paths(G,$f_{(f_{src} \rightarrow V^{SDN})}$) + get_cur_paths($G$,$f_{(V^{SDN} \rightarrow f_{dst})}$) $flag_f \leftarrow 1$;
19 **return** $\bigwedge_{f \in F} flag_f$;

Algorithm leverages Algorithm 2 to re-assign paths for flows. It also follows an iterative approach. Algorithm 2 gets information on the current SDN switch deployment in Lines 1-4. Then, It retrieves the current path of each flow f in Lines 5-10. Algorithm 2 tests any shortest paths traverse at least one programmable SDN switch (Lines 11-18). As shown in Lines 17-18, if no one satisfies the requirement, Algorithm 2 detours the flow first from the flow's source node to a nearest SDN switch node and then from that SDN switch node to the flow's destination node. This step can satisfy the flow programmable constraint on most topologies unless containing isolated nodes.

VI. Simulation

We present the simulation of the EASON problem in this section. Firstly, we introduce the simulation setup and the compared algorithms. Then, we show the performance of compared algorithms on various topologies from the real-world using public trace datasets. Simulation results indicate that Algorithm can achieve near-optimal performance while it is scalable on larger topologies.

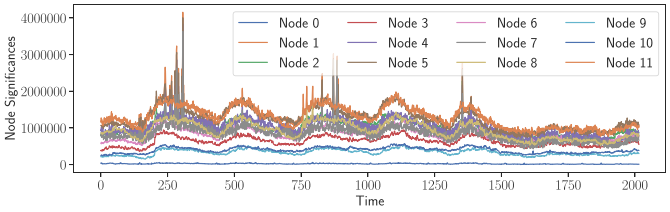


Fig. 7. The Abilene trace in a week.

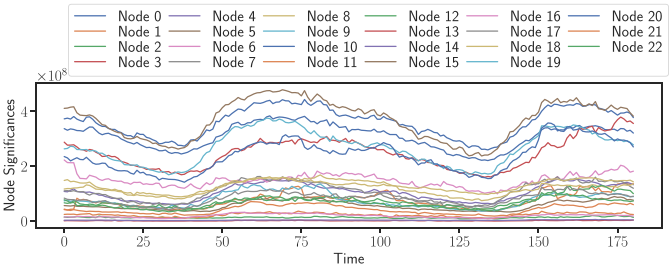


Fig. 8. The GÈANT trace fluctuation as of time.

A. Simulation Setup

We employ real-world backbone topologies from Topology Zoo [77] in our simulation. Topology Zoo provides 262 topologies with standard `gml` files. We conduct the simulations using Python on a Dell PowerEdge R815 server equipped with four AMD Opteron 6220 @3.00 GHz CPU (8-Core) sockets and 192 GB DRAM. We leverage `python-igraph`, a popular Python graph library, to read topologies from `gml` files and find paths. In the simulation, each node contains one switch (or router), and it can be either an SDN device or a legacy device based on the calculated hybrid SDN deployment scheme X. For Abilene [74] and GÈANT [78] topologies, we use real-world trace datasets from [73], [79] to calculate node significances because traffic trace datasets of other topologies are not publicly available. We set the link capacities as the Abilene topology and flow rates as 250 Kbps on other topologies. State-of-the-art network devices support terabits per second level packet processing capability, and we thus set the processing capability of each SDN switch as 1 Tbps.

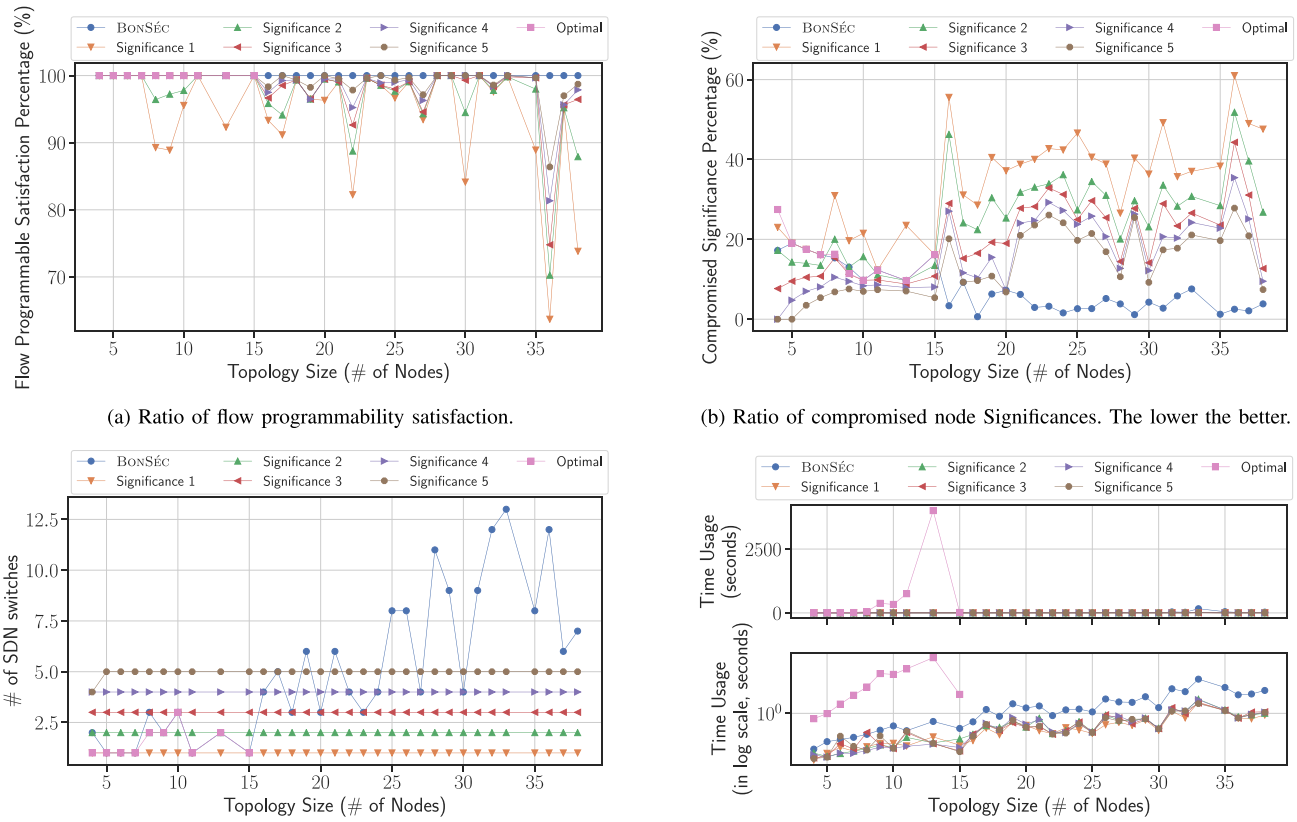
B. Compared Algorithms

We compare the following algorithms in this paper.

- *Optimal:* This is the optimal solution of the EASON problem, which minimizes the propagation of malicious traffic and overall upgrade cost. We use GUROBI [80] to solve it.
- *Significance n:* This solution deploys n SDN switch(es) at n most significant nodes.
- *Algorithm:* This is detailed in Algorithm 1.

C. Simulation Results

This subsection shows our simulation results. We first present the performances of Algorithm, Optimal, and Significance n on several small topologies (the number of links is less than ten). Because of the limited public datasets, we show the stability of the Algorithm algorithm on real-world traces of the Abilene and GÈANT topologies by comparing it



(a) Ratio of flow programmability satisfaction.

(b) Ratio of compromised node Significances. The lower the better.

(c) Number of SDN switches used.

(d) Execution time.

Fig. 9. Performances on different real-world topologies with various sizes.

to Significance n. Our simulation results show that Algorithm can achieve near-optimal performances on small topologies. Meanwhile, it spends only three more seconds to **stably reduce malicious traffic propagation to as low as 4% using less than six SDN switches** (see Fig. 11) without compromising the SDN programmability on Abilene. Likewise, it can achieve a close-to-zero compromised ratio even using less programmable SDN switches compared to Significance n under acceptable time duration on GÈANT (see Fig. 12).

1) Performances on Different Small Topologies: We show the performances of Optimal, Algorithm, and Significance n algorithms on different real-world topologies whose sizes are distinct. We employ a popular optimization problem solver, GUROBI [80], to solve the Optimal algorithm. Due to the complexity of the EASON problem, we fail to run Optimal on topologies over 15 nodes or ten links. Our simulation results show that **Algorithm achieves comparable security protection and cost performances to Optimal using remarkably reduced execution time** on small topologies.

1) *Flow-programmable satisfaction ratio:* As shown in Fig. 9a, Algorithm and Optimal can 100% satisfy the flow-programmable constraint. However, Significance n fails to guarantee, and the inability can be as high as 40% (i.e., Significance 1). By jointly observing the SDN switch usage from Fig. 9c, we can see that Significance n uses more SDN switches but fails to guarantee the programmability. The reason is that the flow programmability constraint is preferentially ensured, and the most secure deployment with a minimal cost is then considered. Significance n's inability to flow-programmable satisfaction is because it merely deploys a single programmable SDN switch at the most significant node(s) without guaranteeing programmability constraint.

2) *Ratio of compromised node significances:* We define the *ratio of compromised node significances* as the ratio of all compromised nodes' significances to all nodes' significances. We leverage this metric to describe how well a network can benefit from deployments from the security perspective. Fig. 9b shows that Algorithm achieves the identical value to that of Optimal on small topologies. Besides, Algorithm can outperform the Significance n algorithm when using the same number of programmable SDN switches (e.g., when topology size is 16, 18, 22, 23, 24, 27, 30), which indicates the inability of the pure node-based approach (i.e., the significance n algorithm). Conversely, the path-traversal approach employed by Algorithm mitigates the attack traversal across the whole network. Algorithm can only achieve an identical security performance on smaller topologies (i.e., topology size is less than 15) using the same number of programmable SDN switches. This is because smaller topologies have limited paths to adjust. Unfortunately, the compromised ratio achieved by the optimal solution on the 4-node topology approaches 30%. The reason is that compromising a single node can significantly fluctuate the performance due to the limited nodes on this topology.

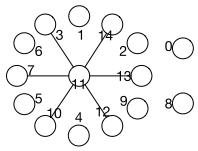


Fig. 10. The Padi topology.

3) *Number of SDN switches:* As depicted in Fig. 9c, when the topology is small, **the number of SDN switches used by Algorithm is the same as that of Optimal in 80% of the cases**. As the topology size increases, the number of instances needed by Algorithm increases, but the compromised node significances ratio is still under 10%. However, up to 60% of the network nodes can be compromised using the Significance *n* algorithm with even more SDN switches.

4) *Execution time:* Optimal is a non-polynomial solution, whereas Algorithm and Significance *n* are polynomial-time solutions. Hence, in Fig. 9d, we can observe an exponential-like increase in the time usage of Optimal, while **Algorithm and Significance *n* use much fewer times** (see the bottom subfigure of Fig. 9d). To demonstrate the differences between Algorithm and Significance *n*, we also convert the linear-scale plot (the top subplot of Fig. 9d) to the log-scale [81] plot (the bottom subplot of Fig. 9d) as the numbers are so close in the linear scale plot. The weird phenomenon in the figure is that the execution time is close to 5 when the topology size is 15. We find this topology is Padi [82], and it contains isolated nodes without degrees (see Fig. 10). Hence, the execution time of Padi is equivalent to a seven-node topology.

2) Performance on Real-World Traces: In this subsection, we show the performances of Algorithm and Significance *n* on the Abilene topology [74] and GÈANT topology [78] as it was in 2005 using the public real-world traffic traces [73], [79]. The Abilene topology consists of 12 nodes and 14 links (see Fig. 5), and GÈANT contains 23 nodes and 37 links. Unfortunately, we fail to run Optimal under both topologies on our testbed due to the NP-hard complexity. We show algorithms' performance over time to indicate that the Algorithm is efficient at all times, which can stably outperform others rather than take the lead on selected occasions.

1) *Flow-programmable satisfaction ratio:* As shown in Fig. 11a, **Algorithm satisfies the programmable constraint for all flows on all tested traces**. While Significance 1 fails on approximately 15% of tested traffic traces under the Abilene topology. Although 98.5% of the satisfaction is achieved on these traces, the hybrid SDN requires each flow to pass an SDN switch for programmability and malicious analysis. Therefore, Significance 1 is insufficient in protecting the network, but when the number of programmable SDN switches is greater than two, Significance *n* can solve this problem on this topology. On the GÈANT topology, we set $n \in [10, 14]$ for fairness as the number of SDN switches used by Algorithm ranges from 11 to 14. Fig. 12a demonstrates the full satisfaction achieved by Algorithm, where all Significant *n* algorithms fail to comply with the flow programmable constraint. Meanwhile, Algorithm can use fewer SDN switches compared to Significance *n*. On some traces, Algorithm may require more SDN switches, as depicted in Fig. 12c, to satisfy the constraint, and we detail the reason in Section VI-C2(3).

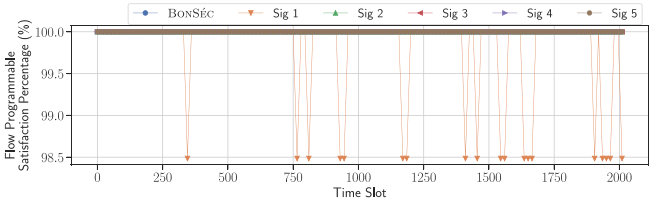
2) *Ratio of compromised node significances:* Fig. 11b shows the ratio of compromised node significances of Algorithm and Significance *n*. **Algorithm can intercept malicious traffic as early as it propagates 4% of a flow's nodes** and can achieve an average of about 10%. However, Significance *n* allows malicious traffic traverse through an average of 34% of flow significances before intercepting the traffic and impacts over $\frac{1}{3}$ of the network. Besides, from both Fig. 11b and Fig. 11c, when using the same number of SDN switches, say four switches, Algorithm can notably outperform Significance 4 for over 30%. As illustrated in Fig. 12b, Algorithm remarkably reduces the ratio of compromised significances compared to the Significant *n* algorithm that is close to 0. However, even the best case of Significance *n* still compromises approximately 6% of the total network significance. This again proves the "security nature" of Algorithm.

3) *Number of SDN switches:* Fig. 11c shows the number of SDN switches used by both algorithms. We can observe a fluctuation in the number of SDN switches used by Algorithm. This fluctuation results from the traffic pattern, as shown in Fig. 7. Significance *n* uses *n* SDN switch(es) to deploy in the Hybrid SDN network, while Algorithm does not require more SDN switches for the flow-programmable constraint. As shown in Figs. 11(c)-11(b), Algorithm can use fewer SDN switches to achieve better security performance, and it employs less than six (50% of all nodes) SDN switches in all traffic traces to satisfy the programmable requirement. Moreover, in about 60% of the test traces, **Algorithm only needs four SDN switches**. Fig. 12c indicates Algorithm uses fewer SDN switches on most traces and uses an identical number of SDN switches on 15% all traces of GÈANT. But as we have mentioned in satisfy the constraint, and we de Sec VI-C2(2), Algorithm can satisfy the flow programmable constraint, whereas Significant *n* fails to. The reason that some traces may use more SDN switches results from the heavy traffic, as depicted in Fig. 8.

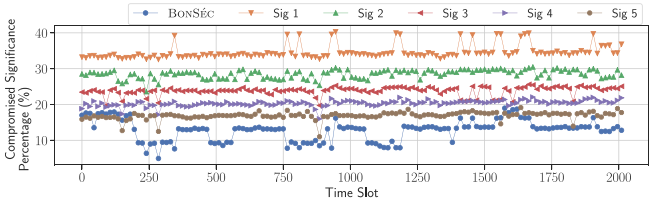
4) *Execution time:* Algorithm employs an iterative approach by selecting nodes in the order of their significance and testing their availability to deploy programmable SDN switch(es). Conversely, Significance *n* only chooses *n* most significant nodes for deploying programmable SDN switches. Hence, the time needed by Significance *n* is shorter. Our simulation results indicate that **Algorithm only needs two seconds** on the Abilene topology and approximately 13s on the GÈANT topology, as depicted in Fig. 11d and Fig. 12d. Since Algorithm is not designed for online deployment purposes, we believe the time usages for Algorithm are acceptable.

VII. Conclusion and Future Work

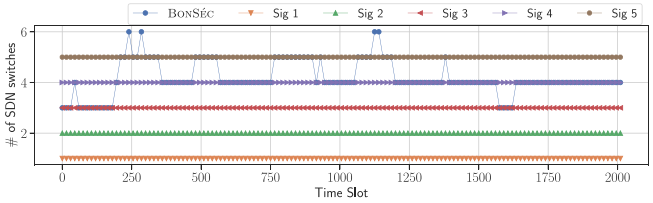
In this paper, we have investigated the End-host-importance-Aware secure service-enabled hybrid Sdn deplOymeNt (EASON) problem. It explores methods of efficiently intercepting malicious traffic and reduces risks using a minimum deployment expense. We have proved the NP-hardness complexity of the EASON problem. Due to the



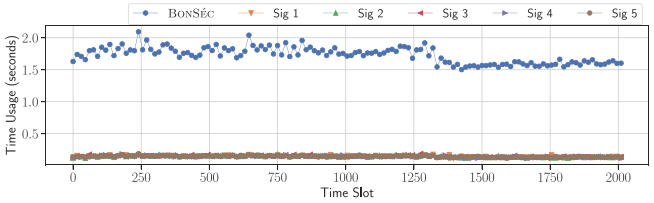
(a) Ratio of flow programmability satisfaction.



(b) Ratio of compromised node significances. The lower the better.

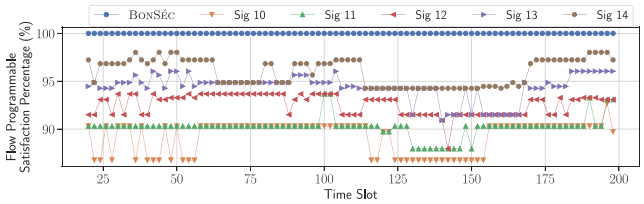


(c) Number of SDN switches used.

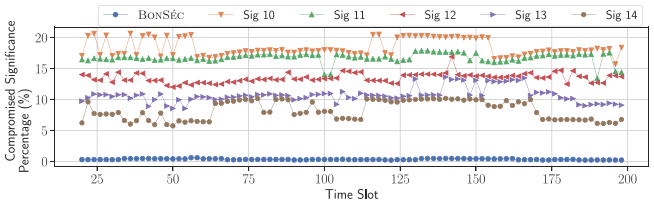


(d) Execution time.

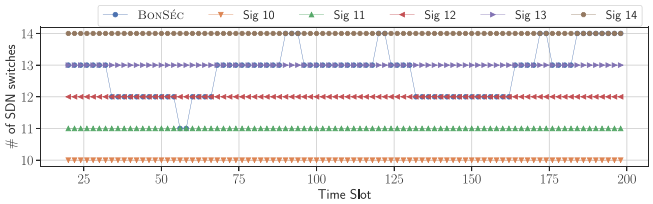
Fig. 11. Performances on the Abilene topology using real-world traces.



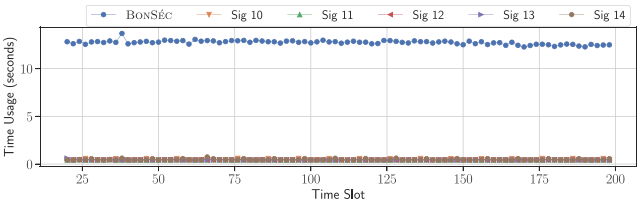
(a) Ratio of flow programmability satisfaction.



(b) Ratio of compromised node significances. The lower the better.



(c) Number of SDN switches used.



(d) Execution time.

Fig. 12. Performances on the GÈANT topology using real-world traces.

complexity, we have proposed an efficient heuristic algorithm called Algorithm. Algorithm retrieves the node significance information from topologies along with their traces and preferentially places SDN switches at the most common anterior nodes. Simulation results indicate that Algorithm can achieve comparable performance to the optimal solution on small topologies and is scalable on larger topologies. Our next step is implementing real-world security services (e.g., network measurement functions for statistics, IDS, IPS) on resource-scarce programmable switches to validate the efficiency of Algorithm on real-world systems. New data structures will be put forth in realizing hardware constraints. We present Algorithm to inspire the use of programmable switches in the hybrid SDN.

ACKNOWLEDGMENT

The authors thank the anonymous reviewers for their valuable comments, which have significantly improved their paper.

REFERENCES

[1] B. A. Forouzan, *Cryptography & Network Security*. London, U.K.: McGraw-Hill, 2007.

[2] B. Zimmer, "LISA: A practical zero trust architecture," in *Proc. Enigma*, Jan. 2018, pp. 1–24.

[3] "Huawei NetEngine AR6000 series enterprise routers datasheet." Huawei Technologies Co., Ltd. 2022. [Online]. Available: https://e.huawei.com/en/material/networking/b17b2a7b964d452b8c44bc889e0d077c

[4] "Router security." Cisco Systems, Inc. 2022. [Online]. Available: http://www.topology-zoo.org/maps/Abilene.jpg

[5] S. Choi *et al.*, "FBOSS: Building switch software at scale," in *Proc. Conf. ACM Spec. Interest Group Data Commun.*, New York, NY, USA, 2018, pp. 342–356.

[6] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[7] H. Mekky, F. Hao, S. Mukherjee, T. V. Lakshman, and Z.-L. Zhang, "Network function virtualization enablement within SDN data plane," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2017, pp. 1–9.

[8] S. Pontarelli *et al.*, "FlowBlaze: Stateful packet processing in hardware," in *Proc. 16th USENIX Symp. Netw. Syst. Des. Implement. (NSDI)*, Feb. 2019, pp. 531–548.

[9] O. Michel, R. Bifulco, G. Rétvári, and S. Schmid, "The programmable data plane: Abstractions, architectures, algorithms, and applications," *ACM Comput. Surv.*, vol. 54, no. 4, pp. 1–36, 2021.

[10] L. Fawcett, S. Scott-Hayward, M. Broadbent, A. Wright, and N. Race, "Tennison: A distributed SDN framework for scalable network security," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 12, pp. 2805–2818, Dec. 2018.

[11] P. Vörös and A. Kiss, "Security middleware programming using P4," in *Proc. Int. Conf. Human Aspects Inf. Security Privacy Trust*, 2016, pp. 277–287.

[12] B. Lewis, M. Broadbent, and N. Race, "P4ID: P4 enhanced intrusion detection," in *Proc. IEEE Conf. Netw. Funct. Virtualization Softw. Defined Netw. (NFV-SDN)*, 2019, pp. 1–4.

[13] T. Yang *et al.*, "Elastic sketch: Adaptive and fast network-wide measurements," in *Proc. SIGCOMM Conf.*, 2018, pp. 561–575.

[14] J. Sonchack, D. Loehr, J. Rexford, and D. Walker, "Lucid: A language for control in the data plane," in *Proc. ACM SIGCOMM Conf.*, 2021, pp. 731–747.

[15] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined wan," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, New York, NY, USA, 2013, pp. 3–14.

[16] D. Levin, M. Canini, S. Schmid, F. Schaffert, and A. Feldmann, "Panopticon: Reaping the benefits of incremental SDN deployment in enterprise networks," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, Philadelphia, PA, USA, Jun. 2014, pp. 333–345.

[17] H. Lu, N. Arora, H. Zhang, C. Lumezanu, J. Rhee, and G. Jiang, "HybNET: Network manager for a hybrid network infrastructure," in *Proc. Ind. Track 13th ACM/IFIP/USENIX Int. Middlew. Conf.*, 2013, pp. 1–6.

[18] M. Markovitch and S. Schmid, "SHEAR: A highly available and flexible network architecture marrying distributed and logically centralized control planes," in *Proc. IEEE 23rd Int. Conf. Netw. Protocols (ICNP)*, 2015, pp. 78–89.

[19] D. K. Hong, Y. Ma, S. Banerjee, and Z. M. Mao, "Incremental deployment of SDN in hybrid enterprise and ISP networks," in *Proc. Symp. SDN Res.*, 2016, pp. 1–7.

[20] C. Jin, C. Lumezanu, Q. Xu, Z.-L. Zhang, and G. Jiang, "Telekinesis: Controlling legacy switch routing with OpenFlow in hybrid networks," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res.*, 2015, pp. 1–7.

[21] C. Jin, C. Lumezanu, Q. Xu, H. Mekky, Z.-L. Zhang, and G. Jiang, "Magneto: Unified fine-grained path control in legacy and OpenFlow hybrid networks," in *Proc. SOSR*, 2017, pp. 75–87.

[22] W. Feng, Z.-L. Zhang, C. Liu, and J. Chen, "Clé: Enhancing security with programmable dataplane enabled hybrid SDN," in *Proc. 15th Int. Conf. Emerg. Netw. Exp. Technol.*, 2019, pp. 76–77.

[23] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[24] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "SilkRoad: Making stateful layer-4 load balancing fast and cheap using switching ASICs," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2017, pp. 15–28.

[25] M. T. Arashloo, Y. Koral, M. Greenberg, J. Rexford, and D. Walker, "SNAP: Stateful network-wide abstractions for packet processing," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 29–43.

[26] N. Foster *et al.*, "Frenetic: A network programming language," *ACM SIGPLAN Notices*, vol. 46, no. 9, pp. 279–291, 2011.

[27] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, "Modular SDN programming with pyretic," in *Proc. USENIX*, Berkeley, CA, USA, 2013, pp. 40–47.

[28] Sandhya, Y. Sinha, K. Haribabu, "A survey: Hybrid SDN," *J. Netw. Comput. Appl.*, vol. 100, pp. 35–55, Dec. 2017.

[29] X. Huang, S. Cheng, K. Cao, P. Cong, T. Wei, and S. Hu, "A survey of deployment solutions and optimization strategies for hybrid SDN networks," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1483–1507, 2nd Quart., 2019.

[30] R. Amin, M. Reisslein, and N. Shah, "Hybrid SDN networks: A survey of existing approaches," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3259–3306, 4th Quart., 2018.

[31] S. Khorsandroo, A. G. Sánchez, A. S. Tosun, J. M. Arco, and R. Doriguzzi-Corin, "Hybrid SDN evolution: A comprehensive survey of the state-of-the-art," *Comput. Netw.*, vol. 192, Jun. 2021, Art. no. 107981.

[32] E. Rojas, R. Amin, C. Guerrero, M. Savi, and A. Rastegarnia, "Challenges and solutions for hybrid SDN," *Comput. Netw.*, vol. 195, Aug. 2021, Art. no. 108198.

[33] K. Poularakis, Q. Qin, K. M. Marcus, K. S. Chan, K. K. Leung, and L. Tassiulas, "Hybrid SDN control in mobile ad hoc networks," in *Proc. IEEE Int. Conf. Smart Comput. (SMARTCOMP)*, 2019, pp. 110–114.

[34] P. Shi, S. Rivera, L. Pike, Z. Fei, J. Griffioen, and K. Calvert, "Enabling shared control and trust in hybrid SDN/legacy networks," in *Proc. 28th Int. Conf. Comput. Commun. Netw. (ICCCN)*, 2019, pp. 1–9.

[35] T. Feng and J. Bi, "OpenRouteFlow: Enable legacy router as a software-defined routing service for hybrid SDN," in *Proc. 24th Int. Conf. Comput. Commun. Netw. (ICCCN)*, 2015, pp. 1–8.

[36] M. Caria, A. Jukan, and M. Hoffmann, "SDN partitioning: A centralized control plane for distributed routing protocols," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 381–393, Sep. 2016.

[37] Z. Guo, S. Dou, Y. Wang, S. Liu, W. Feng, and Y. Xu, "HybridFlow: Achieving load balancing in software-defined WANs with scalable routing," *IEEE Trans. Commun.*, vol. 69, no. 8, pp. 5255–5268, Aug. 2021.

[38] T. Y. Cheng and X. Jia, "Compressive traffic monitoring in hybrid SDN," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 12, pp. 2731–2743, Dec. 2018.

[39] J. Zhang, M. Ye, Z. Guo, C.-Y. Yen, and H. J. Chao, "CFR-RL: Traffic engineering with reinforcement learning in SDN," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2249–2259, Oct. 2020.

[40] J. Galán-Jiménez, M. Polverini, and A. Cianfrani, "A scalable and error-tolerant solution for traffic matrix assessment in hybrid IP/SDN networks," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 1, pp. 251–264, Mar. 2020.

[41] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 90–97, Feb. 2015.

[42] S. Palkar *et al.*, "E2: A framework for NFV applications," in *Proc. 25th Symp. Oper. Syst. Principles*, 2015, pp. 121–136.

[43] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker, "NetBricks: Taking the V out of NFV," in *Proc. 12th USENIX Symp. Oper. Syst. Des. Implement. (OSDI)*, 2016, pp. 203–216.

[44] G. P. Katsikas, T. Barbette, D. Kostić, R. Steinert, and G. Q. Maguire, Jr., "Metron: NFV service chains at the true speed of the underlying hardware," in *Proc. 15th USENIX Symp. Netw. Syst. Des. Implement. (NSDI)*, 2018, pp. 171–186.

[45] T. Barbette, G. P. Katsikas, G. Q. Maguire, Jr., and D. Kostić, "RSS++: Load and state-aware receive side scaling," in *Proc. 15th Int. Conf. Emerg. Netw. Exp. Technol.*, 2019, pp. 318–333.

[46] K. Zhang, D. Zhuo, and A. Krishnamurthy, "Gallium: Automated software middlebox offloading to programmable switches," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl. Technol. Archit. Protocols Comput. Commun.*, 2020, pp. 283–295.

[47] A. A. Pranata, T. S. Jun, and D. S. Kim, "Overhead reduction scheme for SDN-based data center networks," *Comput. Stand. Interfaces*, vol. 63, pp. 1–15, Mar. 2019.

[48] R. Bifulco and G. Rétvári, "A survey on the programmable data plane: Abstractions, architectures, and open problems," in *Proc. IEEE 19th Int. Conf. High Perform. Switch. Routing (HPSR)*, 2018, pp. 1–7.

[49] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[50] X. Jin *et al.*, "NetCache: Balancing key-value stores with fast in-network caching," in *Proc. 26th Symp. Oper. Syst. Principles*, 2017, pp. 121–136.

[51] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "HULA: Scalable load balancing using programmable data planes," in *Proc. Symp. SDN Res.*, 2016, pp. 1–12.

[52] J. McCauley, A. Panda, A. Krishnamurthy, and S. Shenker, "Thoughts on load distribution and the role of programmable switches," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 49, no. 1, pp. 18–23, 2019.

[53] L. Chen, G. Chen, J. Lingys, and K. Chen, "Programmable switch as a parallel computing device," 2018, *arXiv:1803.01491*.

[54] J. Lim, S. Nam, J.-H. Yoo, and J. W.-K. Hong, "Load balancing algorithm with programmable switch," in *Proc. 21st Asia–Pacific Netw. Oper. Manag. Symp. (APNOMS)*, 2020, pp. 326–329.

[55] D. Kim *et al.*, "TEA: Enabling state-intensive network functions on programmable switches," in *Proc. Annu. Conf. ACM Spec. Interest Group Data Commun. Appl. Technol. Archit. Protocols Comput. Commun.*, 2020, pp. 90–106.

[56] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: Query-driven streaming network telemetry," in *Proc. Conf. ACM Spec. Interest Group Data Commun.*, 2018, pp. 357–371.

[57] R. B. Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "PINT: Probabilistic in-band network telemetry," in *Proc. Annu. Conf. ACM Spec. Interest Group Data Commun. Appl. Technol. Archit. Protocols Comput. Commun.*, 2020, pp. 662–680.

[58] R. Govindan, I. Minei, M. Kallahalla, B. Koley, and A. Vahdat, "Evolve or die: High-availability design principles drawn from Googles network infrastructure," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 58–72.

[59] A. Singh *et al.*, "Jupiter rising: A decade of Clos topologies and centralized control in Google's datacenter network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 183–197, 2015.

[60] A. AlSabeh, E. Kfoury, J. Crichigno, and E. Bou-Harb, "P4DDPI: Securing P4-programmable data plane networks via DNS deep packet inspection," in *Proc. Netw. Distrib. Syst. Security (NDSS) Symp.*, 2022, pp. 1–7.

[61] L. Tang, Q. Huang, and P. P. C. Lee, "SpreadSketch: Toward invertible and network-wide detection of superspreaders," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2020, pp. 1608–1617.

[62] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proc. Symp. SDN Res.*, 2017, pp. 164–176.

[63] X. Zhang, L. Cui, F. P. Tso, and W. Jia, "pHeavy: Predicting heavy flows in the programmable data plane," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 4, pp. 4353–4364, Dec. 2021.

[64] B. Trach, A. Krohmer, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer, "ShieldBox: Secure middleboxes using shielded execution," in *Proc. Symp. SDN Res.*, 2018, pp. 1–14.

[65] R. N. Smith and S. Bhattacharya, "Firewall placement in a large network topology," in *Proc. 6th IEEE Comput. Soc. Workshop Future Trends Distrib. Comput. Syst.*, 1997, pp. 40–45.

[66] S. Lee, M. Purohit, and B. Saha, "Firewall placement in cloud data centers," in *Proc. 4th Annu. Symp. Cloud Comput.*, 2013, pp. 1–2.

[67] M. Bouet, J. Leguay, T. Combe, and V. Conan, "Cost-based placement of vDPI functions in NFV infrastructures," *Int. J. Netw. Manag.*, vol. 25, no. 6, pp. 490–506, 2015.

[68] M. L. Artz, "NetSPA: A network security planning architecture," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, 2002.

[69] Q. Lv, J. Zhu, F. Zhou, and Z. Zhu, "Network planning with bilevel optimization to address attacks to physical infrastructure of SDN," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2020, pp. 1–6.

[70] Z. Guo, W. Feng, S. Liu, W. Jiang, Y. Xu, and Z.-L. Zhang, "RetroFlow: Maintaining control resiliency and flow programmability for software-defined WANs," in *Proc. Int. Symp. Qual. Service (IWQoS)*, Phoenix, AZ, USA, Jun. 2019, pp. 1–10.

[71] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni, "Analysis of a denial of service attack on TCP," in *Proc. IEEE Symp. Security Privacy*, 1997, pp. 208–223.

[72] H. Wang, D. Zhang, and K. G. Shin, "Detecting SYN flooding attacks," in *Proc. 21st Annu. Joint Conf. IEEE Comput. Commun. Soc.*, vol. 3, 2002, pp. 1530–1539.

[73] Y. Zhang. "Abilene traffic matrix." 2022. [Online]. Available: http://www.cs.utexas.edu/$\sim$yzhang/research/AbileneTM

[74] "Abilene network topology." Topology Zoo. 2022. [Online]. Available: https://www.cisco.com/c/en/us/products/security/router-security/index.html

[75] N. Alon, B. Awerbuch, and Y. Azar, "The online set cover problem," in *Proc. 35th Annu. ACM Symp. Theory Comput.*, 2003, pp. 100–105.

[76] M. Thorup, "Undirected single-source shortest paths with positive integer weights in linear time," *J. ACM*, vol. 46, no. 3, pp. 362–394, 1999.

[77] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.

[78] "GÉANT looking glass." GÉANT. 2022. [Online]. Available: https://lg.geant.org

[79] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 83–86, 2006.

[80] "Gurobi optimizer." Gurobi. 2022. [Online]. Available: http://www.gurobi.com

[81] "Pyplot scales." Matplotlib. 2022. [Online]. Available: https://matplotlib.org/3.1.3/gallery/pyplots/pyplot_scales.html

[82] "Padi2 network diagram." Internet2. 2022. [Online]. Available: http://web.archive.org/web/20081206174517

Wendi Feng (Member, IEEE) received the Ph.D. degree from Beijing University of Posts and Telecommunications advised by Prof. J. Chen. He was co-advised by Prof. Z.-L. Zhang with the University of Minnesota—Twin Cities from 2018 to 2020. He is currently a Faculty Member with the School of Computer Science, Beijing Information Science and Technology University. His research interests include computer networks, service computing, software-defined networks, and network function virtualization.

Chuanchang Liu is currently an Associate Professor with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His current research interests include mobile device security, cloud computing, and service-oriented computing.

Bo Cheng (Member, IEEE) is currently a Professor and the Vice Director of the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His current research interests include mobile device security, cloud computing, Internet of Things and big data analysis, network service, and intelligence.

Junliang Chen is currently a Professor and the Academic Leader with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His current research interests include service-oriented computing and service generation system. He is a member of the Chinese Academy of Science and the Chinese Academy of Engineering, and a Fellow of the China Computer Federation.

Zhiguo Wan (Member, IEEE) received the B.S. degree in computer science from Tsinghua University, Beijing, China, in 2002, and the Ph.D. degree in information security from the National University of Singapore in 2007. He was a Postdoctoral Fellow with the Katholieke University of Leuven, Belgium, and an Assistant Professor with the School of Software, Tsinghua University. He is a Principal Investigator with Zhejiang Lab, Hangzhou, Zhejiang, China. His main research interests include security and privacy for cloud computing, Internet of Things, and blockchain.