SVLAN: Secure & Scalable Network Virtualization

Jonghoon Kwon
ETH Zürich
jong.kwon@inf.ethz.ch

Taeho Lee
ETH Zürich
tlee@inf.ethz.ch

Claude Hähni
ETH Zürich
chaehni@student.ethz.ch

Adrian Perrig
ETH Zürich
adrian.perrig@inf.ethz.ch

***Abstract*—Network isolation is a critical modern Internet service. To date, network operators have created a logical network of distributed systems to provide communication isolation between different parties. However, the current network isolation is limited in scalability and flexibility. It limits the number of virtual networks and it only supports isolation at host (or virtual-machine) granularity. In this paper, we introduce Scalable Virtual Local Area Networking (SVLAN) that scales to a large number of distributed systems and offers improved flexibility in providing secure network isolation. With the notion of *destination-driven reachability* and *packet-carrying forwarding state*, SVLAN not only offers communication isolation but isolation can be specified at different granularities, e.g., per-application or per-process. Our proof-of-concept SVLAN implementation demonstrates its feasibility and practicality for real-world applications.**

I. Introduction

Network virtualization has become increasingly prominent in the modern Internet and is recognized as a core technology for future networking. Data-center operators often resort to creating logical networks for VMs, called virtual networks, to provide communication isolation between core and edge clouds. In the fifth-generation mobile network (5G), the notion of network slicing (which allows the partitioning of a network into virtual slices) is recognized as a key innovation. Essentially, network virtualization creates multiple virtual networks on top of a shared physical network infrastructure, striving for security, isolation from malicious activities, and cost-effective network management.

In particular, virtual extensible LAN (VXLAN) is a network-virtualization technique that enables end-to-end network isolation and is widely used in many data centers to support large cloud-computing environments [33]. It connects multiple VXLAN tunnel endpoints (VTEPs) configured with a same VXLAN network identifier (VNI), applying an overlay technique to encapsulate layer-2 frames within layer-3 packets, to isolate the communication from unwanted external entities. Despite its short history, it has become a dominant protocol since its introduction in 2013. Nonetheless, VXLAN has two main limitations: scalability and flexibility.

First, each VTEP device needs to be frequently updated to maintain a mapping between VMs and VNIs. The size of the mapping information will grow with the number of VMs and VNIs that servers host. In addition, with the increase in the number of VMs and VNIs, the volume of ARP traffic between VMs will increase. Already in today's data centers, ARP traffic requires significant bandwidth, and this will only worsen as more VMs and VNIs are created [17].

Second, VXLAN supports a static isolation of communication at host or VM granularity. It is difficult to isolate communication at different granularities, e.g., per-application, which may be useful when one subset of VMs wishes to allow communication for a specific application, while another subset disallows that application. In VXLAN, two separate VXLAN segments must be created in such a setting, which leads to scalability issues when the demand for other isolation granularities increases. In addition, once a virtual network is created, removing a VM from the network is cumbersome as it requires coordination between multiple parties (e.g., remove state from each VTEP where the virtual network is deployed, configuring each VM in the virtual network). However, such dynamism is necessary for isolating communication on per-process or per-application basis, since applications and processes could be short-lived.

To address these limitations, in this paper, we propose a secure and scalable virtual LAN (SVLAN) architecture. Each endhost (or VM) dynamically initiates virtual networks by expressing its consent for reachability, facilitating the separation of enforcement and access management/delegation. On a high level, SVLAN achieves communication isolation based on *explicit consent*. That is, the sender (e.g., a VM or a virtualized application) that wishes to communicate with the receiver must acquire consent from the receiving VM (or application). The core network of the data center ensures that the sender has obtained receiver consent and only forwards packets that carry the consent; packets without consent are dropped, providing communication isolation.

SVLAN weds local network slicing with inter-domain routing. To this end, we integrate two emerging technologies and introduce new approaches to neutralize their drawbacks. For capability-based networking, we separate the roles of authentication, authorization, and verification by introducing new entities including authorization delegate (AD) and verifier. This allows us to push capabilities instead of requiring global pull, thereby dispersing load, improving deployability, and reducing propagation delay. For segment routing, we also introduce a new approach, verifiable segment routing, which improves the integrity and authenticity of routing control. With an authorization proof, path segments remain unalterable, being subordinated to the corresponding sender and receiver pair, preventing potential routing-path hijacking, packet injection, or replay, and dramatically improving the robustness of routing control.

Network and Distributed Systems Security (NDSS) Symposium 2020
23-26 February 2020, San Diego, CA, USA
ISBN 1-891562-61-4
https://dx.doi.org/10.14722/ndss.2020.24162
www.ndss-symposium.org

SVLAN communication proceeds in 3 steps: (1) the sender acquires an authorization token from the receiver's *authorization delegate*; (2) it includes the token in each transmitted packet; and (3) the network forwards the packets only if the tokens are valid. We do not fix or impose a specific entity to validate the tokens. Instead, we create a conceptual entity called *verifier* and design our protocol to be generic such that any entity in the network, as long as they are on packets' communication paths, can serve as a verifier (e.g., VTEPs or routers). Such flexibility enhances the deployability of our architecture since the verifiers can be determined based on a variety of factors, including market demand or policy regulation. Nonetheless, different choices of verifiers do provide different technical benefits, and we evaluate the technical merit for each potential deployment scenario.

We have implemented a prototype framework that includes a fully-functional tunneling endpoint, authorization delegate, and verifier, and have extended the data plane to support SVLAN packet forwarding. Through extensive evaluation in a real-world environment, we demonstrate that SVLAN introduces negligible processing overhead (6.7μs for authorization and 26ns for verification) and tolerable latency inflation (32.0μs on average). In addition, we further discuss an incremental deployability along with various SVLAN deployment scenarios.

In summary, this paper makes the following contributions:

- We introduce SVLAN, a generic framework that enables scalable, flexible, and viable network virtualization at various granularities, e.g., host and application granularity.
- We enhance security in network isolation by enforcing the receiver's consent in communication by design, that enables network filtering of the edge of the network.
- We provide an evaluation of tradeoffs between the new approach and existing approaches.

II. Related Work

VLAN and VXLAN. Data centers are shared infrastructures that host various services operated by different parties, even potentially conflicting ones. Furthermore, security-sensitive data are increasingly processed in data centers as well. Hence, securing data centers to prevent leakage of one party's data to another has become an important issue. To this end, data centers rely on the concept of isolation—isolating computing, storage, and network resources.

To date, the operators of data centers typically rely on virtual networks (VNs), which are isolated logical networks of VMs, to achieve network isolation. The virtual LAN (VLAN) standard (IEEE 802.1q) describes a way to create virtual networks between hosts that are not directly connected to the same network switch, dramatically improving security, efficiency, and traffic management. However, VLAN cannot scale to a large number of VNs [33], [55]. For example, the VLAN identifier (VID) in the VLAN header is only a 12-bit value, which imposes a hard limit of 4094 virtual networks (`0x000` and `0x0xFFF` are reserved). To provide better scalability, the network community has proposed virtual extensible LAN (VXLAN) [33], which interconnects layer-2 networks over an underlying layer-3 network.

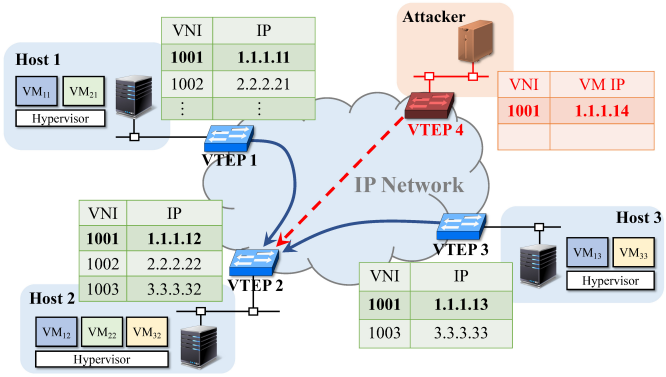

Fig. 1: The stateful VTEP hampers flexibility on VN migration. In addition, adversaries can easily join the other VNs and manipulate VNIs to forward unwanted traffic.

VXLAN achieves better scalability than VLAN in two ways. First, the VNI is a 24-bit value, which allows for up to 16 million VXLAN segments (or virtual networks) to coexist. Second, it organizes the core network of data centers as a layer-3 network (in contrast to layer-2 for VLAN) enabling IP-based routing, which provides better scalability. It reduces the amount of state (i.e., per-VM state) at top-of-rack switches in data centers, and enables equal-cost multipath to utilize redundant links in the network. Although VXLAN scales better than VLAN, it is still limited in scalability and flexibility [21], [49].

Security has never been a major consideration in VXLAN. VTEPs in different network segments recognize each other by joining the layer-3 multicast group via the Internet Group Management Protocol (IGMP). It helps fill up the forwarding tables on VTEPs by broadcasting ARP requests to the multicast group. Only VTEPs that are listening to the multicast-group traffic respond to the ARP requests, enabling VMs to discover the MAC addresses of target VMs and allowing unicast packets to other VMs in the same VNI. However, as shown in Figure 1, there is no concrete countermeasure against adversaries with enough capability to alter packet headers to impersonate another VNI. Therefore, it might fail to isolate the VN from unwanted traffic. Lately, some mechanisms have been suggested to secure the VXLAN environment using VXLAN membership information [11]. However, the authorization of membership requires additional state (i.e., per-VM state), and hence it worsens the flexibility and scalability.

To overcome these limitations, we design a new network virtualization approach with the notions of "destination-driven reachability" and "packet-carrying forwarding state", achieving a high degree of freedom in network virtualization with a strong guarantee for isolation. The following are brief introductions to related areas.

Software-Defined Networking (SDN). As an early pre-cursor of SDN, the SANE [13] system shares some basic design tenets with SVLAN: packets carry capabilities for each traversed switch, for the entire end-to-end path. In contrast, SVLAN is more light-weight, carrying a single capability for the

destination, more in the spirit of VLAN. Furthermore, SVLAN comes along with incremental deployability; it does not require a full deployment across the network nor any modification on the endhost.

The typical SDN-based approach, as proposed by the Ethane [12] system, embodies state on network switches to enforce policy and encode forwarding behavior. Fine-grained forwarding behavior can be defined, at the cost of per-flow state on all intermediate switches. Mobility and network failures require state re-configuration on intermediate switches, which we aim to avoid in SVLAN– as an advantage, flow management in SVLAN can be handled exclusively by the end application without requiring alteration of in-network state.

Micro-segmentation. Micro-segmentation [34], [25] can create secure zones in cloud environments that enable tenants to securely isolate their workloads from others. Traditionally, data centers have employed various security primitives, such as firewalls, IDS/IPS, and ACLs, to protect the internal network and their customers from security breaches. However, once an adversary bypasses the protection methods, they have access to the data center to carry out attacks. With micro-segmentation, fine-grained security policies are applied to individual workloads even for the internal network, enhancing attack resilience.

Since the concept of micro-segmentation has been proposed by VMWare [58], [37], many data centers have introduced micro-segmentation into their network, promoting the idea of how network virtualization can be improved with security. Yet, a specific way to realize the idea has not been standardized, remaining at an intermediate development stage.

Off-by-Default. Over the past decade, researchers have made various proposals to allow receivers to enforce their consent towards incoming traffic, and the proposals can be classified into two broad categories [60]: filtering-based and capability-based approaches. In filtering-based approaches [7], [31], [4], receivers express their consent as filtering rules and these rules are installed at network entities (e.g., routers). Then, for each packet, the router evaluates the filtering rules of the respective receiver to determine its consent. However, the filtering approach can create false positives and false negatives when routers need to aggregate filtering rules of the users. Moreover, disseminating and updating the filtering rules on Internet routers is a non-trivial task.

On the other hand, in capability-based approaches [3], [59], the receivers—not routers—grant permissions to the senders, where a permission is often implemented as a (cryptographic) token that can be validated by the network. Then, the senders include the tokens in their packets, and the network only forwards packets with valid tokens. Capability systems have one major vulnerability: denial-of-capability (DoC) attacks [5], where an adversary floods the receivers with capability-request messages so that legitimate senders cannot receive capabilities from the receivers. To mitigate DoC attacks, previous work has focused on limiting the capability-request rate by the senders [42], [8], [1].

Segment Routing. Segment routing [18] realizes the source-routing paradigm [56]; a source who wants to communicate with a destination builds the forwarding path of network packets by collecting a set of routing pieces, called "path segments", and assembling them as an ordered list of segments. Network infrastructure ensures that the packets are steered through the intended forwarding path. Hence, it greatly improves transparency and control over packet forwarding, resulting in many desirable properties such as multipath communication, path-aware networking and high-performance data transmission. Furthermore, considering the slow speed of the current Internet to converge after network failures [28], [15], [23], in fact, segment routing would provide faster recovery compared to the current Internet, also achieving higher routing flexibility [19].

SR-MPLS [54] is MPLS-based segment routing in which a sender specifies the forwarding path by adding stacked MPLS labels to the packet header. Thanks to the MPLS data plane—enabling flexible and efficient network programming—and the backward compatibility with existing MPLS-enabled networks, SR-MPLS is being considered as one of the most viable approaches. The segment-routing architecture has evolved to also embrace the IPv6 data plane, called SRv6 (segment routing for IPv6) [10]. With the introduction of the network programming concept into SRv6, it mitigates the significant encapsulation overhead, a list of 128-bits IPv6 addresses. Recently, segment-routing approaches have been presented as promising technologies that would fuel 5G innovation [24], [43]. In addition, the SCION future Internet architecture can express segment routing semantics at inter-domain scale by expressing paths at AS-level granularity [61], [44].

III. Overview

The goal of this paper is to build a lightweight architecture that enables secure, scalable, and fine-grained network slicing. That is, each host expresses consent towards incoming traffic while the network only delivers authorized packets to their respective destinations. In this section, we describe the desired properties for our architecture, a high-level system model that we consider to achieve the goal, and the assumptions.

A. Desired Properties

Scalability. The new architecture should allow a *high degree* of expressiveness for the definition of virtual networks in multi-tenancy environments, such that it achieves scalable network virtualization.

Flexibility. Network administrators should be able to explicitly regulate the communication policies without an ambiguity that might lead to the failure of network isolation. In addition, the virtual network should be easily updatable at any time.

Security. Our architecture must ensure that packets without receivers' consent do not reach their intended destinations, so that the virtual network is completely isolated. In addition, the sender should not be able to transfer the authorization to another sender.

Practicality. We consider the practicality of our architecture from the following two perspectives:

- *Performance Overhead:* The additional latency for acquiring the receiver's consent should be on the order of a round-trip time (RTT) and degradation of the packet-forwarding performance due to the verifier should be minimal.

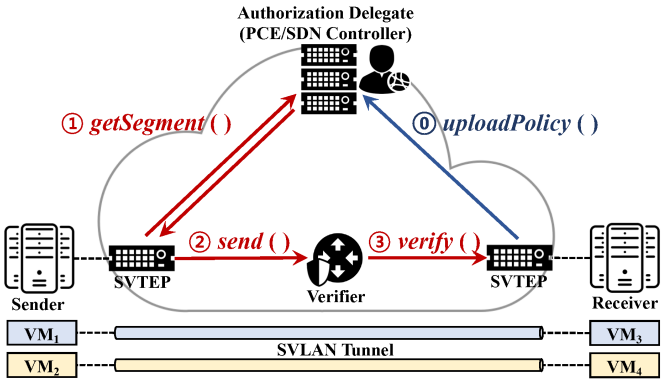

Fig. 2: High-level overview of SVLAN.

- *Deployability:* We must ensure that entities that would deploy the two functionalities (i.e., authorization delegate and verifier) should be incentive-compatible. In addition, compatibility with the existing network-virtualization protocols and devices needs to be ensured.

B. System Model

Figure 2 depicts a high-level overview of our architecture, which consists of three functional entities in addition to the sender and the receiver: 1) an **authorization delegate (AD)** of the receiver that authorizes communication from the sender to the receiver, 2) a **verifier** that ensures the sender has acquired consent from the receiver, and 3) **stateless virtual tunnel endpoints (SVTEPs)** that bridge traffic between SVLAN segments.

SVLAN communication proceeds in four steps:

1) The receiver (or network administrator) uploads its receiving policy to the AD, which represents a virtual network configuration that specifies who can send packets to the receiver.
2) In preparation for a data transmission, the sender's SVTEP acquires the consent of the receiver by requesting an authorization proof from the receiver's AD. The AD evaluates the receiver's policy and issues an authorization proof as a proof of consent.
3) The sender's SVTEP sends a packet to the receiver. In this packet, the sender includes the authorization proof that represents the receiver's consent.
4) The verifier ensures that the packet is valid by verifying the validity of the proof, and forwards the packet only if the packet is valid.

Separation of Roles. A central property of SVLAN's design is its separation of the various roles. There are three important aspects to our model:

First, the process of providing consent (i.e., authorizing a sender) is split between two entities: the receiver and its AD. The receiver generates the receiving policies; however, the AD—not the receiver—grants consent, i.e., provides authorization proofs, based on the receiver's policy. The separation increases resilience against DoC attacks by allowing receivers to choose well-provisioned ADs.

Second, we separate the role of the AD and the verifier. In our model, an AD issues authorization proofs that the senders embed into each packet. Then a verifier ensures that packets have valid proofs. We separate the two functionalities for two reasons. 1) The performance requirement between the two functionalities are vastly different since verifiers handle data packets and need to process packets at a significantly higher rate. 2) Decoupling the functionality fosters deployability since we can assign the roles to the most appropriate entity (i.e., incentive-compatible for each functionality). Note that we are not the first to consider this separation, which is also made by capability-based systems [47], [38].

Third, we also split the role of an endhost and the SVTEP. Similar to the existing network-virtualization schemes, an SVTEP functions as the tunnel endpoint bridging two virtual network segments to be logically connected. The SVTEP acquires authorization proofs from the AD and performs en- or de-capsulation for inbound or outbound packets, respectively. In this design, unchanged endhosts are supported, which improves deployability.

It is important to note that the separation of roles does not mean that the functions cannot be collocated. For example, although the receiver delegates the authorization process to its AD, the receiver could still issue a path segment itself. In addition, a large security service provider, such as Cloudflare, can act as both an AD and a verifier. However, the functional separation means that our protocol should work even if the functions are implemented by different network entities that are not collocated. We discuss various deployment models and their implications later in Section VIII.

C. Assumptions

Source Authentication. Our architecture requires packets to be authenticated to their corresponding sender so that a malicious sender cannot impersonate a legitimate sender to acquire receiver's consent and send packets pretending to be the legitimate sender. There are well-established cryptographic mechanisms [29], [32], [9] that can be used to authenticate packets to their corresponding sender.

Secure Cryptography. We assume that cryptographic primitives that we use are secure: signatures and message authentication codes (MACs) cannot be forged, and encryption cannot be broken, as long as the cryptographic keys remain secret.

Time Synchronization. We also assume that entities in the network (i.e., senders, verifiers, and ADs) are loosely time synchronized within a few seconds by using a protocol such as NTP. Time synchronization is used to enforce the time constraints of a receiver's consent, but does not affect to the control plane in SVLAN.

IV. SVLAN Architecture

We now describe the details of our architecture.

A. Path Segment as Receiver's Consent

Receiver's consent is an integral part of our architecture, achieving dynamic network virtualization based on each receiver's needs. With the concept of segment routing, we utilize the path segment as the receiver's consent towards incoming

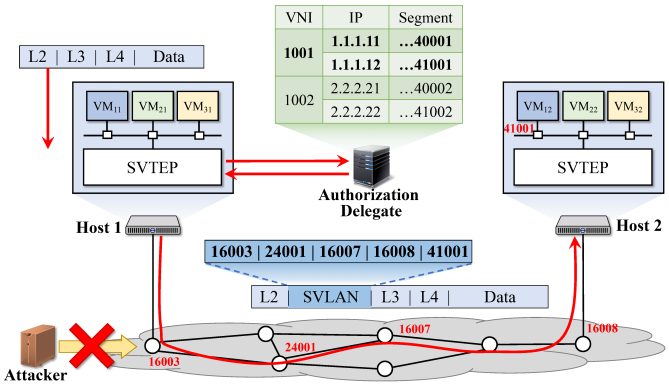

Fig. 3: Only authorized senders can retrieve valid path segments and forward packets to the receiver.

traffic; only authorized senders are able to acquire the path segments that steer packets to the destination, so that only members in the same virtual network can communicate, as Figure 3 shows. We extend the concept of *path segment as receiver's consent* even further to be more secure, by providing segments with authorization proofs that indicate the validity of the segments.

Requirements. We now identify the requirements for designing the authorization based on the desired properties.

- Ensure that a segment is only valid for authorized senders. That is, a segment cannot be transferred to or shared with any other sender.
- Ensure that the bandwidth overhead due to segments and authorization proofs in the packets remains small.
- ADs must be able to compute and disseminate the valid path segments efficiently.
- Verifiers must be able to verify the path segments efficiently to ensure that there is no significant increase of the processing delay. To this end, we should not require large amounts of state, e.g., per-host state, on verifiers. In addition, packet verification should not require additional communication overhead (e.g., a challenge–response protocol [2], [39]) for checking the validity of the segments.

Strawman Approaches. The process of authorizing packets is based on two pieces of information that the AD provides to the verifier and the SVTEP. The first piece of information is provided to the verifier as a verification instruction to evaluate if an incoming packet has been authorized by its receiver. The second piece of information is provided to the sender-side SVTEP as the authorization proof, which the SVTEP includes in every packet and the verifier uses to verify receiver consent.

There are various ways to design the authorization process based on the amount of information placed on the above-mentioned two pieces of information. We first present two strawman approaches (*state centric* and *asymmetric centric*) that place all necessary information in one of the two pieces. Then, we present our design, which represents a middle ground and combines the advantages of the two approaches.

In the *state-centric approach*, we can place all necessary information onto the verification instructions that the AD provides to the verifier. Specifically, the AD generates a separate verification instruction for each sender (or for each flow) and sends the instructions to the verifier. The verifier stores all received instructions in its forwarding table; then for each incoming packet, it finds a matching instruction using the packet content (e.g., network and transport headers) and only forwards packets if it can find a matching instruction. This approach is similar to software-defined networks (SDN), such as the OpenFlow protocol: the AD can be compared to the centralized controller in SDN; verification instructions to the forwarding instructions; and the verifier to the SDN switch.

This approach reduces bandwidth overhead since we do not add any additional information—authorization proof—to data packets. However, it increases state overhead at the verifier, since the verifier needs to store per-sender or per-flow instructions.

In the *asymmetric-centric approach*, we can place authorization proofs in the packets. Specifically, we can define a certificate for the AD and use digital signatures using the corresponding private key to create the authorization proofs. For example, to create a per-sender authorization proof, we use a digital signature over the address of the sender. When the verifier receives a packet, it ensures that the authorization proof is valid by verifying the digital signature in the proof.

This design avoids the state overhead at the verifier, since the verifier does not need to maintain per-sender or per-flow forwarding instructions. However, this approach requires asymmetric cryptography, which is computationally expensive. Specifically, the AD needs to generate signatures when creating authorization proofs, and the verifier needs to verify signatures when authorizing packets. Furthermore, the high computational overhead introduces the signature-flooding vulnerability.

Segment with Authorization Proof. We combine the two strawman approaches to benefit from both. Specifically, our design avoids a large amount of state found in the first approach and, at the same time, avoids asymmetric cryptography required for the second approach.

We design the proof based on message authentication codes (MACs) using a shared key k between the AD and the verifier as shown in Equation (1). Namely, the proof is the MAC with the path segment, the address of the sender, the expiration time of the proof and the number of proofs as its input:

$$Proof(k, Seg_{S\to R}, Addr_S, ExpTime, N) = \mathsf{MAC}_k\left(Seg_{S\to R} \mid Addr_S \mid ExpTime \mid N\right) \quad (1)$$

This avoids the disadvantages of the strawman approaches and satisfies the requirements. Since modern hardware can compute symmetric cryptography efficiently, our design enables efficient processing on the AD and the verifier. Furthermore, our design requires only per-AD state on the verifier to store the symmetric keys shared with the ADs. That is, the AD does not need to create per-sender or per-flow forwarding instructions for the verifier, and the verifier does not need to store these instructions. Lastly, our design prevents a sender from sharing the path segments, since the segment ($Seg_{S\to R}$)

and sender's address ($Addr_S$) are inputs to compute the proof and the shared key is known only to the AD and the verifier.

Similar to the SDN controller in [48], the AD generates and distributes the shared keys with the verifiers; the key sharing can be done over the secure communication channel. The centralized key distribution simplifies key management, and thus sophisticated key-establishment protocols are not required.

B. Protocol

This section describes the three protocols (Figure 2) in SVLAN: `getSegment()`, `send()` and `verify()`.

`getSegment().` The sender, S, must obtain the consent of a receiver, R, to send a packet to the receiver. To this end, the sender-side tunnel endpoint (SVTEP_S) asks for path segments including authorization proof from the receiver's AD. The AD creates an authorization proof for the sender after it verifies the sender's authorization using the receiver's receiving policy. Specifically, the protocol proceeds as follows:

1) S requests an authorization proof to A (R's AD):

$$S \rightarrow A: \ Addr_S \mid Addr_R$$

2) A checks R's receiving policy and issues segments with proof:

$$A \rightarrow S : Seg_{S \rightarrow R} \mid Addr_A \mid ExpTime \mid N \mid Proof_{1 \leq i \leq N} \ (k_i, Seg_{V_i \rightarrow R}, Addr_S, ExpTime, N)$$

On the request message (1), the sender provides the address ($Addr_S$) from which it would use to send a packet to the receiver and the address of the receiver ($Addr_R$). While it is possible to infer the address of the sender from the source address in the network header of the request packet, we explicitly include the address that would be used as the source address for multi-homed hosts: If the sender uses different networks to communicate with the receiver's AD and the receiver itself, the sender must separately specify the address that it would use to communicate with the receiver.

The AD consults the receiver's policy to determine if the receiver is willing to receive packets from the sender ($Addr_S$). If yes, the AD generates an authorization proof based on Equation (1) using $Addr_S$ and the path segment $Seg_{S \rightarrow R}$ that leads to $Addr_R$ (2). Additionally, the AD specifies the expiration time ($ExpTime$) to limit the validity of the proof. The $ExpTime$ can be determined from the receiver's policy or the AD could specify an arbitrary but small value.

The AD may generate multiple proofs so that the sender's packet can be verified by multiple verifiers. In such a case, the AD generates a proof for each of the verifier (V_i) using a subset of segments ($Seg_{V_i \rightarrow R}$), because the segments can be modified while the packet travels. Each proof is generated using a symmetric key (k_i) that the AD shares with each verifier. In Section VIII-B, we describe the choice of verifiers when we discuss about the deployment locations of verifiers.

`send().` A sender can successfully send a packet to the receiver only if it has path segments with a valid (e.g., non-expired) authorization proof. In a data packet to the receiver, the sender embeds the segments ($Seg_{S \rightarrow R}$) and the authorization proof ($Proof_{1 \leq i \leq N}$) to prove to the verifiers that the sender has acquired the receiver's consent:

3) S embeds Seg_R and $Proof$ on its data packet to R:

$$S \rightarrow R : Seg_{S \rightarrow R} \mid Addr_A \mid ExpTime \mid N \mid ptr \mid Proof_{1 \leq i \leq N} \mid Data$$

The packet also contains fields to help the verifiers verify the proof. Namely, it has $Addr_A$, which a verifier uses to determine the shared symmetric key (k_i); N to indicate the number of proofs; and ptr to indicate the proof that the verifier should verify. Lastly, the sender must use the source address ($Addr_S$) that was specified in the authorization-request message (1) as the source address in the network header. Otherwise, the packet would be dropped by the verifier for containing an incorrect proof.

A verifier runs `verify()` to ensure that the proof in the packet is valid. To this end, the verifier first identifies the symmetric key (k_i) based on the AD's address ($Addr_A$) and the proof ($Proof_i$) that it verifies. Then, it verifies the proof using the source address and path segment ($Addr_S$ and $Seg_{V_i \rightarrow R}$, respectively) from the packet header, and $ExpTime$ and N as input. The verifier drops the packet as invalid (i.e., without receiver's consent) if the proof is expired or the proof cannot be verified correctly. Once the verifier successfully verifies the proof, it increments ptr to indicate the next proof ($Proof_{i+1}$) and continues to forward the packet towards the receiver.

C. Authorization Policy

Requirements. We design the authorization policies with the following requirements:

- **Expressiveness:** It should be easy for the receivers to express their receiving policies.
- **Flexibility:** We would like to design the authorization policies such that users can easily describe their receiving policies at different granularities.
- **Scalability:** While guaranteeing expressiveness with the authorization policies, we need to limit the number of policy rules that a user can have, such that ADs can evaluate receiver's policy efficiently.

Blacklist and Whitelist. We allow receivers to express their policies as both a blacklist, to prevent specific senders from sending traffic, and a whitelist, to indicate who can send packets to the sender. Having only one of the lists is sufficient for the receivers to express their policies, but using both lists provides sufficient flexibility to the users in expressing their consent policies while reducing the size of their policies (i.e., the number of rules in the blacklist and whitelist). For example, a receiver that only talks with a selective set of senders can "whitelist" such hosts while a receiver that requires more universal access can use a blacklist to filter unwanted hosts.

Simultaneously having two lists can create conflicts and to resolve such conflicts, we create one additional field *priority*. In case of a conflict, a rule with a higher priority would be enforced. Note that, rules with the same priority are processed according to the order of freshness; a rule later inserted would

be executed. In summary, each entry of the two lists has the format

$$\langle src\_IP, mask, src\_port, dst\_port, proto\rangle \Rightarrow \langle priority, Action\rangle \quad (2)$$

Here, $Action$ determines the action that the AD should take when a sender's request matches the corresponding policy entry. If an entry is a blacklist entry, $Action$ would be not to issue a proof, while if the entry is a whitelist entry, it determines the granularity at which a generated proof would be valid (e.g., per-host, per-application). We provide more detail about the granularity along with the left part of the entry in the next paragraph.

Policy Granularity. We support network-, host-, and also application-level policies. Network-level policies allow the receivers to express their consent for an entire network while host-level policies are for individual hosts. In terms of Equation (2), these policies can be specified by specifying src_IP and $mask$ to reflect the target network or host and setting dst_port and $proto$ to be a wildcard ('$*$'). In addition, we allow application-level policies, which additionally require the destination port and protocol information. We support application-level policies to maximize the flexibility on network isolation.

Basically, we do not consider per-flow policies since flow information does not enrich the policy expressiveness beyond the application-level policies, yet can significantly increase the number of policies. Specifically, source port is typically chosen arbitrarily (i.e., ephemeral ports) and can change (i.e., address translation); thus, src_port is marked as a wildcard by default. Nevertheless, the receiver can regulate the sender's traffic on a per-flow basis by specifying the $Action$ of the corresponding whitelist policy entry to be for a single flow.

NAT. The AD should avoid any ambiguity in host identification, and thus the policy must clearly specify each host with a globally unique address identifier. For the hosts behinds NAT devices, their address might be opaque if the multi-tenancy environment hosting them only supports private address. In such a case, each host can be distinguishable with the translated public IP address together with the forwarding port.

V. Implementation

We implement a prototype SVLAN on top of SCION [44], a future Internet architecture that supports native segment routing, for testing the general functionality and performance. In addition, since SCION is designed for inter-domain segment routing, we expand the SCION data plane with an SVLAN extension header that conveys the intra-domain path segment and authorization proof, enabling communication at an endhost (and even application) granularity. The main reasons of choosing SCION as the underlying architecture for our implementation are 1) it already supports an embedded public-key infrastructure (PKI) that satisfies our assumption described in Section III-C, 2) it also supports segment routing along with the separation of control and data planes, 3) it has a flexible packet design that supports various extension headers, and 4) it is easy to construct a testing environment in SCIONLab[1].

[1]https://www.scionlab.org

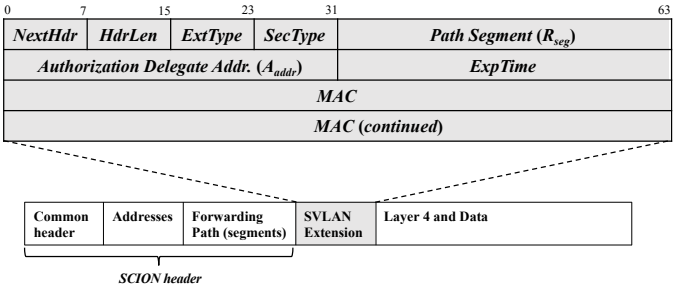

Fig. 4: The header format of the SVLAN extension.

Control Plane. In the latest version of the SCION codebase[2], an endhost employs the SCION daemon, called `sciond`, to interact with the SCION control plane: it initiates a path request message (i.e., `SegReq()`) to acquire path segments necessary to build forwarding paths toward a destination, when the sender attempts a new communication establishment. The path service, a control-plane application similar to an SDN controller, replies to the requests by aggregating path segments, verifying them, and providing them to the requester. Once `sciond` successfully gathers the path segments and constructs forwarding paths, the sender puts a forwarding path into the each packet header and transmits the packets.

We extend the SCION control-plane applications to enable the `getSegment()` protocol: `sciond` operates as a part of SVTEP, managing the `getSegment()` request and reply, and the path service functions as an AD. When an endhost attempts a connection establishment, `sciond` requests path segments and authorization proof to the path service. The path service consults the database that contains the network isolation policy to determine if the sender and receiver are in a same VN. If yes, the path service patches up path segments that lead to the receiver, along with the corresponding authorization proof, and replies back to `sciond`.

Data Plane. We also modify the SCION data-plane code to realize the `send()` and `verify()` protocols. When `sciond` successfully acquires path segments and authorization proofs, SVTEP creates a SVLAN tunnel, generates SVLAN-enabled SCION packets, and forwards the packets. The SCION network protocol, implemented as the `snet` class, provides interfaces for handling the SCION packet transmission. Similar to the general UNIX socket, it supports SCION network APIs such as $Listen$, $Accept$, $Bind$, $Read$, and $Write$, which enable SCION connection establishment, SCION packet generation, and interpretation. Here, we add APIs that extend the SCION header with SVLAN information. For instance, by adding a *Write* overriding method that expands a SCION header with an SVLAN extension, it supports both the existing SCION header generation and the SVLAN-enabled SCION header generation simultaneously.

To generate SVLAN-enabled SCION packets, we leverage the extension header field in the SCION header structure to convey the SVLAN extension. This header design brings the following advantages. First, the SVTEP can easily retrieve the SVLAN header and process the verification. The `HdrLen` in

[2]https://github.com/scionproto/scion

TABLE I: Processing times (in μs) for `getSegment()` requests from authorized and unauthorized senders.

	Legitimate requests			Illegitimate requests		
# of clients	Min.	Max.	Avg.	Min.	Max.	Avg.
1e3	4.8	8.4	6.0	4.0	7.8	6.0
1e4	4.7	9.6	6.0	3.2	9.7	6.0
1e5	4.8	11.0	6.8	4.0	9.6	6.2
1e6	4.9	11.3	6.7	4.0	10.2	6.2

the SCION common header points out the exact offset where the extension header exists, thereby improving the processing delays on SVTEPs. Second, by separating the SVLAN-based routing from the inter-domain routing, it keeps the compatibility to the underlying network infrastructure. Lastly, it simplifies the SVLAN implementation without a significant change in inter-domain routing infrastructure.

Extension Header. Figure 4 details the SVLAN extension header structure. The header begins with three bytes of general extension header fields [44]. $SecType$ specifies the type of MAC that are used for generating the authorization proof, such as `CMAC` or `HMAC`. Note that, in our prototype, we primarily use the 128-bit `AES-CMAC` [53].

The next field contains the path segment that steers the packet to the receiver. Since the table lookup in routing is known to be a bottleneck for high-speed data transmission, we anticipate not to keep a forwarding table. Instead, following the stateless routing principle in SCION, we design Seg_R such that it directly indicates the interface identifier ($IFID$) to which the packet should be forwarded. Furthermore, to enable network isolation at an application granularity, it also indicates the binding port number on which the recipient application listens. In conclusion, the Seg_R consists of two-bytes-long $IFID$ and $Port$. The following $Addr_A$ specifies the IP address of the AD, so that helps SVTEP to retrieve the k_A.

$ExpTime$ is a timestamp generated by the AD, which represents the expiration time of the authorization proof. The timestamp is encoded as a 4-bytes unsigned integer, expressing Unix time with a second granularity. SCION supports time synchronization with second-level precision. That is, it satisfies the last assumption we described in Section III-C. As default, we set $ExpTime = T + 300s$, where T is the time a AD receives a `getSegment()` request. And finally, an authentication tag is following. The length of MAC is variable depending on the type of MAC function and k, but in our implementation, we set $len(\mathsf{MAC}) = 128\ bits$.

VI. Evaluation

Now, we evaluate the performance of SVLAN by scrutinizing the following properties: processing overhead for `getSegment()` and `verify()`, latency influence, and capacity overhead.

A. Microbenchmarks

We first conduct microbenchmarks for the key operations in SVLAN. The operations are performed and measured on commodity machines equipped with Intel Xeon 2.1 GHz processors and an XL710 40 GbE QSFP+ network adopter.

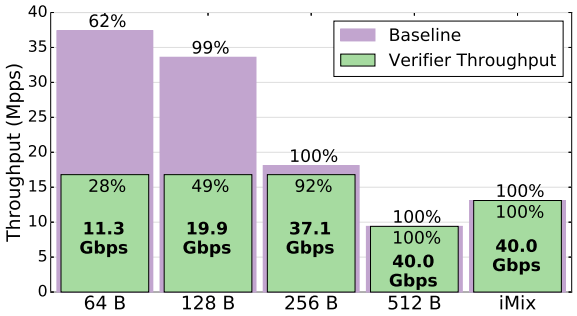

Fig. 5: Forwarding performance of a verifier.

`getSegment()`. To investigate the performance of authorization, we quantify the processing time required for proof generation by the AD; it includes the time for membership checking (receiver's consent) and the MAC computation. For the scalability measurement, we vary the size of the consent database by increasing the number of clients up to 1 million. Each client has 1000 authorized senders. Table I shows the minimum, maximum, and average results for both authorized and unauthorized `getSegment()` requests. The measurements are performed 2000 times for each database.

The results indicate that the processing time is negligible compared to the network latency. The AD requires 6–7 μs to process each request on average, and it does not present any significant differences regardless of whether the request is legitimate or not; that is, the processing delay caused by `getSegment()` is mainly coming from the database lookup, not from the MAC computation.

We observed a slight increase in the maximum processing times for requests from both authorized and unauthorized senders increase while the number of clients increases—$8.4 \rightarrow 11.3$ μs and $7.8 \rightarrow 10.2$ μs for legitimate and illegitimate requests, respectively. This result is expected since the lookup time for $Addr_S$ and $Addr_R$ would increase if the size of database increases. Nonetheless, the increase in processing time is negligible; only a few microseconds for a million entries. This results also indicate that SVLAN scales well in the number of clients.

`verify()`. We also evaluate the processing overhead on a verifier. To this end, we have implemented the `verify()` on the Data Plane Development Kit (DPDK) [46], and evaluate forwarding performance for various packet sizes including a representative mix of Internet traffic (iMIX) [36].[3] For the comparison, we also measure the forwarding performance of typical IP forwarding on the same machine as the baseline.

Figure 5 shows the results. For the small packets (64 and 128 bytes), the forwarding performance for SVLAN packets degrades by approximately 50%. Although `verify()` for

[3]iMIX refers to typical current Internet traffic; its profile specifies the proportion of packets of a certain size. Since the profile is based on statistical sampling from actual Internet traces, performance evaluation using an iMIX of packets is considered a good representation of real-world traffic.

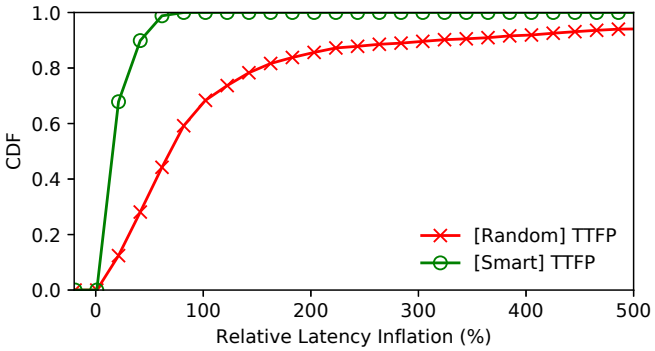

Fig. 6: Latency inflation of the `getSegment()` protocol for the deployment on the Amazon EC2 Cloud.

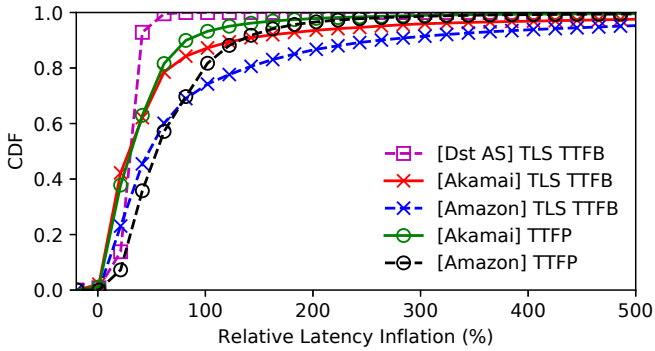

Fig. 7: Latency inflation of the `getSegment()` protocol for the large-scale simulation.

a single packet requires only 26 ns, the baseline exhibits extremely short processing time—20 and 26 ns for 64-byte and 128-byte packets, respectively—leading to an overall decrease in forwarding performance. For large packets, however, it shows optimal performance and reaches the maximum throughput. The evaluation results show the efficiency of the SVLAN verifier that can handle 40 Gbps links that are fully saturated with common Internet traffic patterns at line speed.

B. Amazon Deployment

To evaluate the latency inflation on the connection initialization in SVLAN, we deploy SVLAN on the Amazon Cloud. We initiate an EC2 instance at the 14 data centers distributed over four continents, namely Europe, North America, Asia, and Oceania. By deploying a fully functional SVLAN prototype, each EC2 instance can perform as an SVLAN endhost empowered with SVTEP, a verifier, and an AD. In this setup, a simple client–server application runs on the endhost to transfer data over SVLAN at application granularity.

Next, we select three instances as the sender, the receiver, and the AD. Note that we collocate the verifier and the receiver to realize an on-path verifier that avoids unnecessary detours in data transmission. There are two different selection strategies applied: random selection and smart selection. In the random selection, we randomly select three EC2 instances and conduct experiments for all possible combinations. It gives us 2184 rounds of experiments. In the smart selection, we first choose two EC2 instances for the sender and receiver, and then assign another instance closest to the sender as the AD. This approach is more realistic since it reflects the typical cloud-based service model where clients generally contact the closest regional cloud. From the smart selection, we get 168 different combinations in total.

For each round of experiments, we measure two competitive perspectives; the communication latency with and without SVLAN respectively. For the first measurement, we disable the SVLAN functionality such that the sender could engage a communication with the receiver directly. This measurement serves as the baseline latency. Second, we enable SVLAN introducing an additional latency due to the `getSegment()` and `verify()`, and measure the latency for the time to first packet (TTFP). The comprehensive latency is compared with the baseline latency to compute the relative latency inflation. Figure 6 depicts the results in the form of a cumulative distribution function (CDF).

The random-selection approach introduces a significant latency inflation. This is expected since a high portion of the combinations of three instances has an inefficient deployment model where the sender and receiver are close to each other while the AD is far away. For example, in an extreme case, the latency inflation increases by up to 1873% if the sender and receiver are in Europe (e.g., Frankfurt and Paris) while the AD is located in East Asia (e.g., Seoul).

For the smart-selection approach, the latency inflation is below 75% in all combinations. More precisely, the latency inflation is less than 50% for 93.4% of the combinations, and the 93.3% of additional latencies do not exceed 78 ms (32.0 ms on average) which is a tolerable latency overhead. The results drive us to the intuition that the latency overhead introduced by SVLAN is negligible for the modern cloud environment.

C. Large-scale Simulation

Now, we take one step further to investigate the latency inflation of SVLAN for a large-scale deployment. To this end, we leverage RIPE Atlas[4] to simulate the sender and receiver distributed geographically over wide areas. We randomly select the RIPE nodes including 590 probes and 122 anchors distributed across 684 ASes in 178 countries. In this experiment, the probes and anchors serve as the sender and receiver respectively. We further extend the list of receivers with Alexa's top-100 domains to see the impact of SVLAN on TLS connections.

To simulate the AD, we introduce three different deployment scenarios: 1) We use the 14 Amazon EC2 instances we have initiated as a cloud provider with a small footprint. 2) For a cloud provider with a large footprint, we leverage Akamai's CDN network. To determine Akamai's edge-cloud servers that are closest to the senders, we utilize the DNS system; we trigger DNS queries from the RIPE probes to Akamai's DNS server, which in turn reply with the server addresses that are closest to the probes. 3) We also simulate the AD on the

[4]https://atlas.ripe.net/

receiver's AS. The different deployment simulation allows us to evaluate the impact of the AD's location.

We cannot deploy our SVLAN code to the RIPE nodes, Akamai network, and the Alexa's top-100 servers. To overcome this limitation and investigate the latency inflation, we perform analysis based on latency measurements. This ignores the processing overhead of `getSegment()` and `verify()`, but these overheads are negligible compared to network latency (see Section VI-A). We use ping measurements to estimate the latency between two entities, and then project the latency measurement to the number of RTTs needed to complete the connection. Figure 7 shows the relative latency inflations that we simulate with the aforementioned experimental setup.

From the results, we make the following observations: 1) The latency inflation in the modern Internet environment is tolerable. The latency inflation is less than 50% for 67% of the measurements and averages 70%. 2) The cloud-provider model with a large foot print shows overall better performance than on a small cloud footprint. 3) TLS does not strongly affect the latency inflation. 4) In many cases, the AD on the receiver's AS demonstrates lower latency inflation than the cloud-based AD models. The observations suggest that the deployment of the AD is the key to minimizing the latency inflation. We further discuss the location of the AD in Section VIII-A.

D. Bandwidth Overhead

To measure the bandwidth overhead introduced by SVLAN, we first measure the size of the extra header required to send packets. The extra header may differ depending on the implementation, but here we measure the size of the extra header based on two implementation scenarios: with (i) SR-MPLS and (ii) SCION. In SR-MPLS, an additional header for the authorization proof is necessary. SR-MPLS normally allows up to three labels and each label has a size of 4 bytes. Including additional 24 bytes for an authorization proof, the SVLAN header becomes 36 bytes in total. The maximum payload size is therefore 1424 bytes per frame when attributing 20 bytes each to layer-3 and layer-4 header and 36 bytes to the SVLAN header on a general Ethernet frame with $MTU = 1500bytes$.

SCION requires 8 bytes for the common header, 16 bytes for the addresses, 24 bytes for the forwarding path, and 32 bytes for the SVLAN header for the same number of labels and the authorization proof (Figure 4). This results in up to 1400 bytes of payload for each Ethernet frame. In VXLAN, a total of 50 bytes of additional headers are generated including 8 bytes of VXLAN header and 42 bytes of encapsulation header. Therefore, the maximum size of the payload for each frame is 1410 bytes. Table II shows the comparison results.

We estimate the goodput on a fully saturated 1 Gbps link to see the bandwidth overhead. With a normal Ethernet frame, it requires 1538 bytes including an interframe gap of 12 bytes, and thus the link supports 81274 packets per second. The total amount of data that can be actually transmitted is approximately 949 Mbps. If we apply the same measurement to others, the goodputs of VXLAN, SVLAN (SR-MPLS), and SVLAN (SCION) are 916, 926 and 910 Mbps, respectively. The results show that SVLAN has no severe bandwidth overhead compared to VXLAN.

TABLE II: Comparison of the header sizes, maximum payload, and network performance on a 1 Gbps link. The SVLAN header contains three segment labels and one authorization proof.

			SVLAN	
	Ethernet	VXLAN	SR-MPLS	SCION
Extra header (bytes)	-	50	36	60
Max payload (bytes)	1460	1410	1424	1400
Max goodput (Mbps)	949	916	926	910

VII. Security Analysis

We now discuss potential attack scenarios, their significance, and how effectively our SVLAN design mitigates them.

Threat Model. We mainly consider two different goals of the adversary: 1) infiltrate an isolated network without authorization, and 2) disrupt network operation by leveraging the SVLAN protocols. We consider that the adversary has enough capability to compromise and control all SVLAN entities in the network except for the AD; the AD is typically allocated on well-provisioned and highly-secured systems, e.g., core network, that can tolerate large amounts of incoming traffic and security breaches.

A. Compromise the SVLAN Isolation

The objective of this attack class is to compromise the SVLAN isolation without proper authorization. To this end, an attacker may attempt to acquire a valid authorization proof, or enforce unauthorized packet forwarding. The attacker has clear incentives to perform such attacks, e.g., gaining access to a restricted network zone or reserving more capability in packet forwarding. We start by describing attacks that deceive the SVLAN control plane and data plane; then we describe brute-force attacks and compromisation attacks.

Source-Address Spoofing. An attacker may perform source address spoofing to defeat SVLAN isolation. This attack can be performed in the control plane to obtain an authorized proof from the AD by impersonating an authorized address, and in the data plane to misuse a sniffed authorization proof and send packets to the destination.

We consider to use authentication to secure the control plane. If the AD is in the same local network (also within the same VLAN), then source authentication can be performed by the AD by issuing an unique authorization proof to each host through configuration or secure DHCP. In that case, we can assume that the additional authorization proofs that are fetched are secure. If the AD is outside the LAN (or the VLAN that is created across domains), then a secure channel from the source to the AD needs to be established. This can be a TLS-protected connection, in which case the source will need a certificate that the AD can verify. The SVLAN design is scalable with respect to management, as the source verification and communication policy is verified at a single place (the AD) and enforced at a single place (the verifier).

For the data plane, we consider the same setting as all the virtual LAN approaches; the network ensures separation of

TABLE III: The number of packets per second (PPS) and the required time to brute-force the SVLAN MAC (in years) for different link bandwidths.

	64-bit MAC		128-bit MAC	
Link	PPS	Time [years]	PPS	Time [years]
1 Gbps	976562	5.99e6	919177	1.17e25
10 Gbps	9765625	5.99e5	9191176	1.17e24
100 Gbps	97656250	5.99e4	91911764	1.17e23

traffic to prevent eavesdropping, thus the tags in the data plane are considered secure. To achieve a simple system, the on-path dataplane devices are assumed to be trusted, as otherwise any of those devices could inject malicious traffic toward any host on the virtual LAN (e.g., replay attacks). Within the tradeoff space of security vs. efficiency/deployability, this is the design point selected by virtual LAN systems—it would be an interesting research challenge to also defend against malicious data plane devices but that would likely dramatically increase the complexity of the system.

Brute-Force Attack. An adversary may attempt to deceitfully generate a valid authorization proof for $Addr_M$. The brute-force attack is a classic attack method that allows an attacker to generate all the possible combinations to derive a valid MAC. In our SVLAN prototype, however, the attacker must send $2^{128} \approx 3.4e38$ probe packets to brute-force the MAC. This requires 3.48e32 seconds (or 1.17e25 years) for transmission over the smallest SVLAN frame (136 bytes for 128-bit MACs) on a 1 Gbps network link. Table III shows the time needed in the worst case to successfully brute-force the MAC depending on different sizes of the MAC and the underlying link capacity.

Compromised Verifiers. Compromising verifiers allows an attacker to forward packets without a valid authorization proof. Especially, in the case where only a limited number of verifiers is deployed between two endpoints, the attack has a great impact. For example, the compromised verifier positioned in the hop right before the destination, e.g., the receiver's SVTEP, can pretend all the incoming packets are legitimate. This has the same impact of compromising firewalls which is the last defense line for the victim. For such an attack, no fundamental solution exists. Deploying more verifiers in the network would degrade the impact of the attack by early filtering the attack packets before they reach the compromised verifier. Another viable mitigation is to apply a verification method for infrastructure that monitors invariant security properties [52], [14], [62].

B. Attacks leveraging SVLAN

In this attack class, we consider an attacker who abuses the SVLAN protocols. The purpose of the attack is to disrupt either network operation or SVLAN itself.

Bypassing Security Middleboxes. The original source routing approach was leveraged to bypass the network defense mechanism, e.g., firewall [22]. The attacker specifies the routing path that detours around security equipments, such that attack packets are not filtered and arrive at the victim. Such an attack is prevented by devolving the path construction to the AD; the AD provides predefined $Seg_{S \to R}$ and S can not specify or manipulate the routing path, enforcing routing over verifiers. Furthermore, as the receiver's SVTEP is able to act as a verifier (see Section VIII-B), no attack packets will bypass the last verifier.

Man-In-The-Middle Attack (MITM). An adversary may attempt MITM attacks against applications communicating over SVLAN for eavesdropping, forgery of packet payload, or packet injection. The network isolation mechanism of SVLAN will prevent any host not belonging to the virtual LAN to be able to obtain access to the packets, and thus, prevent the MITM attack. Malicious on-path network equipment, however, might observe the traffic and could attempt an MITM attack; which is a fundamental aspect of such systems in that the network elements are trusted to perform their expected functions. Nevertheless, in SVLAN, the bidirectional communication path between two endpoints does not need to be symmetric, such that the on-path MITM attack can also be mitigated by using asymmetric communication paths.

Amplification Attack. To flood a target host, an adversary may abuse the AD and amplify the attack volume. More precisely, a compromised host sends `getSegment()` requests to the AD with the address of its victim. The AD then replies to the victim with an authorization proof. Nonetheless, this attack is hardly successful, since the authorization-request has a small amplification factor of 4 (i.e., 8 bytes request and 32 bytes reply). Compared to the typical amplification attacks using DNS or NTP, which have the amplification factor of up to 52 and 556 respectively, the `getSegment()` is barely effective.

VIII. Discussion

In this section, we describe some practical considerations and discuss how our model can be realized on today's Internet. More precisely, we discuss the entities that could serve as the ADs or verifiers, as well as how we coordinate SVLAN entities.We then describe how SVLAN supports bidirectional communication when the receiver also needs to send packets to the sender. Later, we discuss incremental deployability.

A. Location of Authorization Delegates

There are two requirements for an entity on the Internet to become an AD. From a technical perspective, ADs should be positioned close to the senders so that the senders can receive authorization proofs with minimal latency overhead. From a business perspective, the entity should have incentives to serve as the ADs for the receivers. We consider two different candidates to serve as the ADs, i.e., the receiver's AS and a third-party entity, such as a cloud service provider.

Receiver's AS. The receiver's ISP has a clear incentive to become an AD for its customers. It can offer AD services as part of a security bundle for their customers or as a value-added service for their premium customers. In addition, it can use the service as a distinguishing feature from other ISPs to attract customers in today's competitive ISP market.

However, using the receiver's AS as the AD may increase communication latency. For senders far way from the receiver, the process of getting authorization proof would incur one additional RTT.

Third-Party Entity. Alternatively, we can use a third-party entity such as a cloud service provider as the AD. Similar to today's cloud-based traffic-scrubbing services, cloud providers can bill the receivers based on the volume of (granted) authorization requests. Communication overhead would be typically lower than when using the receiver's AS as AD but the latency depends on the footprint of the cloud; if the cloud is geographically diverse and has distributed points-of-presence (PoPs), the communication overhead would be reduced. There is also a disadvantage of using a cloud provider as an AD: the cloud operator learns which entities communicate. However, the privacy loss is not as severe as today's traffic-scrubbing services as the data between the sender and the receiver are not forwarded through the cloud.

B. Choice of Verifiers

An entity that serves as a verifier also needs to have an incentive to serve as a verifier. From a technical perspective, the choice of verifiers has implications on the necessary state; specifically, the ADs need to store every symmetric key that they share with the verifiers (Section IV-A). Now, we provide a summary of state overhead at the ADs based on the choices of ADs and verifiers. We consider four entities as candidate verifiers, of which three are on the path between the sender and the receiver (i.e., the receiver, its ISP, and the sender's ISP),[5] and the other is a third-party entity (e.g., a cloud provider) that may be off-path. We also discuss the advantages and disadvantages of each choice.

Receiver. A receiver serves as the last line of defense to drop a packet that it does not agree to, and it can drop the packet with a light-weight operation (i.e., verify the validity of the authorization proof), since symmetric cryptography can be computed efficiently. In such a case, the authorization proof plays a similar role as a TCP SYN cookie [50], which is used to prevent SYN flooding attacks. However, the fact that an unwanted packet has reached the receiver may be problematic: 1) the network has already wasted bandwidth to forward a packet that would be dropped anyways, 2) the receiver may have latent vulnerabilities (e.g., backdoors) that the packet could trigger, and 3) the adversary may be able to congest the links to the receiver or overload the receiver's processing capabilities with superfluous traffic.

In terms of state implication on the ADs, using the receiver as the verifiers does not increase the amount of state at the ADs, since they already store the receiving policies of all receivers.

Receiver's AS. Using the receiver's AS as the verifier alleviates the disadvantages of the above approach, since unwanted packets would be filtered before reaching the receiver. Moreover, the receiver's ISP would be interested in serving as the verifier, since the early filtering increases the efficiency of its network and protects the receivers from potential danger, which the ISP can sell as a value-added service to its customers or use as a distinguishing feature to attract more customers.

To use the receiver's AS as verifiers, the ADs need to store per-AS keys, increasing the state overhead. Note that the number of ASes could be relatively large compared to the number of potential customers on an AS.

Sender's AS. The main advantage of placing the verifier at the sender's AS would be to drop packets early and thus avoid the transmission through the network. However, as seen by other technologies such as egress filtering [6], the sender AS may not have an incentive to filter out traffic for remote destinations, or a malicious source AS could still flood the receiver. Nonetheless, if the entire SVLAN is configured by a single administrative entity such that it is one trusted network, the sender-side verifier becomes an attractive choice.

Third-Party Entity. We also consider using an off-path third-party entity, such as a cloud provider, to serve as the verifier. This approach has three disadvantages: 1) it requires a detour through the cloud, which can potentially increase latency and the size of the packet due to the additional tunnel header to redirect the packet to the cloud; 2) it requires additional per-cloud state at the verifiers; and 3) similar to clouds that offer today's traffic-scrubbing services, the cloud can observe all data traffic, leading to potential privacy problems.

C. Distributed Authorization Delegates

Running a cluster of multiple ADs is a possible deployment approach for enhancing reliability, scalability, and performance. For instance, SDN-based networks, which have a similar architecture as SVLAN, often employ more than one controller to mitigate the issue of single points of failure on the control plane [26], [20]. Furthermore, instead of simply employing an additional AD as a backup system, deploying multiple ADs running in parallel such that each covers a geographical area would help load balancing, achieving scalability [16]. It would also reduce latency by locating ADs closely to the end hosts [51]. To ensure secure operation in running multiple ADs deployed over a wide area, we consider two coordination models for consistency in authorization policy and SVTEP migration amongst ADs.

Coordination of Authorization Delegates. Keeping consistency in authorization policy amongst ADs becomes an essential part of the coordination process. In the context of distributed computing, the overhead in synchronization between the distributed ADs increases as more ADs are joined into the cluster, raising issues of scalability.

We consider consensus algorithms to ensure consistency across the cluster, that can be categorized as mainly two approaches: strong consistency model [41], [45], [35] and eventual consistency model [57], [27], [30]. With the strong consistency model, the authorization policies across the distributed ADs are replicated, assuring the ADs have the latest policies. In contrast, the eventual consistency model omits the consensus process, thus improving the reactivity perceived by SVTEP. The main drawback are possible short-term inconsistencies. To provide a consistent control logic for the entire network, the strong consistency model can be leveraged. Furthermore, open-source projects which enable reliable distributed coordination can be used, such as ZooKeeper[6] or Consul[7].

[5] We do not consider intermediate ISPs, since incentives for such ISPs are unclear.

[6] https://zookeeper.apache.org/

[7] https://github.com/hashicorp/consul

SVTEP Migration. Once consistency of the authorization policies amongst the distributed ADs is secured, the coordination of the SVTEPs becomes less critical; SVTEPs are able to get the same result from any of the ADs. Therefore, the main consideration for the SVTEP coordination is to discover the best AD in terms of scalability, reliability and performance. There are several ways to find the AD, for instance:

- *Explicit configuration*: each SVTEP is configured with AD information as an initial rendezvous point. Since virtual LANs used to be provisioned by a single or a few administrative entities, configuring SVTEPs upon setup is straightforward approach.
- *DNS-based discovery*: through DNS entries (e.g., additional text field), a SVTEP can obtain information on the AD. If only the destination IP address is known, a reverse DNS lookup can first be performed.

We consider that an SVTEP is initially configured with a primary AD IP address and a set of secondary AD IP addresses. The SVTEP first tries to connect to the primary AD and if the connection fails, then tries one of the secondary ADs. That is, an SVTEP is connected to an AD at a time, preventing duplicate processing of asynchronous requests that could result in duplicate path segments or unnecessary resource consumption. Unlike the concept of master and slave in the distributed SDN controller architecture, the primary and secondary ADs are functionally equal except for the delay in the `getSegment()` protocol. Thus, the SVTEP migration keeps to find the best primary AD in terms of latency as well as load balancing, and automatically adjust the target AD when the network changes.

Similar to the multiple-controller support in OpenFlow [40], we intend the migration is initiated by the ADs, which enables fast recovery from potential failure and load balancing. The ADs coordinate the migration of the SVTEP amongst themselves via the management plane, and decides an AD to be a primary. Then, the next primary AD sends a `RoleChange()` message to the SVTEP. It swaps the primary AD from the current one to the requested one. In the migration process, we intend to minimize the functionality in the SVTEPs since it is not desirable and would cause unnecessary overheads.

D. Bidirectional Communication

Thus far, we have only considered one-way communication where the sender sends packet to the receiver. However, in reality, most communication is bidirectional; that is, the receiver also sends packets back to the sender. In this section, we discuss how we support bidirectional communication.

Implicit Consent. One possibility is to implicitly assume that the sender would be willing to accept packets from the receiver, since the sender initiates the communication to the receiver. This model is promising as most communications are bidirectional and has been adopted by NAT and other past proposals [7].

However, the implicit model cannot support the case where the sender wants the communication to be entirely unidirectional. For example, fragile IoT devices may transmit measurement data to the data-aggregation hub but may not want to receive any message back from the hub for security reasons. In addition, the realization of the implicit model in NAT and off-by-default [7] requires the verifier (in case of NAT, the NAT device) to remember all active communication to approve and/or forward packets from the receiver to the sender.

Explicit Consent. Instead, we consider an explicit consent where the receiver must acquire consent from the sender to send a packet to the sender. In one approach, the receiver can acquire consent by requesting an authorization proof from the sender's AD; however, such an approach incurs additional communication latency. Instead, we add a flag (i.e., $RepFlag$) to the proof Equation (1) to indicate that the sender approves packets from the receiver; then, the verifier would only forward a packet to the sender if the $RepFlag$ is set. In terms of the protocol (Section IV-B), we extend protocol 1, 2, and 3 to include the $RepFlag$.

E. Deployment

A major deployment difficulty for many new technologies is the lack of incremental deployability. We conjecture that the deployment of new technologies follows a similar trend: First-movers with a critical need for a new technology start to adopt the technology to their network. As the followers observe customer demand and recognize the necessity of the technology, the mainstream deploys it. In terms of incremental deployability, a viable technology needs to have a clear incentive for the early adopters and incremental benefits for the early majority. However, many proposed schemes are often valuable only if the late majority adopts; there is no benefit for the early adopters.

SVLAN provides strong incremental deployability properties. First, it does not require a global deployment of new protocols, but a partial deployment for endhosts who wish to establish a virtual network. Although it requires coordination amongst the ISPs, setting up a tunnel requires minimal coordination only between networks in which the two endpoints are located; to ensure a secure transmission over untrusted intermediate ISPs between the endpoints, existing underlying mechanisms (e.g., VPN) can be applied for the early deployment phase. The partial deployment of source and destination network ensures the same level of security, scalability, and flexibility, encouraging early adoption.

Second, SVLAN does not necessarily rely on a specific network architecture (e.g., SCION), since it is designed as a generic scheme that can be easily adapted to various architectures that support segment routing. Segment routing is already supported and deployed by all major router vendors: Cisco, Huawei, Juniper, etc. Furthermore, to keep compatibility with the current Internet, we avoid any substantial changes in our design of new entities; the AD can be easily implemented on top of SDN controllers as an add-on application, and the SVTEP and verifier can be realized with vSwitch.

Third, SVLAN benefits early adopters with clear market incentives. Cloud service providers can offer premium services to their customers by setting up secure virtual networks between data centers. ISPs can achieve a better provision for their network with the flexible and scalable network virtualization.

IX. Conclusion

Network virtualization is one of the key components of future Internet innovation. To improve scalability, flexibility and security, we have introduced a framework that leverages the concept of destination-driven networking and packet-carrying forwarding state. SVLAN ensures receivers' consent in communication, enabling fine-grained network virtualization. With the stateless routing and expressive authorization policies, we achieve management scalability for dynamic network slicing. Our evaluation demonstrates that SVLAN introduces a small one-time overhead (32.0 μs of additional latency on average) to the initial communication setup without a significant performance degradation in data transmission. We envision SVLAN to support diverse demands on network slicing, leading to secure communication and efficient administration.

Acknowledgements

We thank Patrick Bamert, Markus Legner, Ankit Singla, and the anonymous reviewers for their insightful feedback and suggestions. We gratefully acknowledge support from ETH Zürich and from the Zürich Information Security and Privacy Center (ZISC).

References

[1] Z. Al-Qudah, E. Johnson, M. Rabinovich, and O. Spatscheck, "Internet With Transient Destination-Controlled Addressing," *Transactions on Networking (TON), IEEE/ACM*, vol. 24, no. 2, pp. 731–744, 2016.

[2] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker, "Accountable Internet Protocol (AIP)," in *Proceedings of the ACM Conference on SIGCOMM*, 2008.

[3] T. Anderson, T. Roscoe, and D. Wetherall, "Preventing Internet Denial-of-Service with Capabilities," in *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*, 2003.

[4] K. Argyraki and D. R. Cheriton, "Active Internet Traffic Filtering: Real-Time Response to Denial-of-Service Attacks," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2005.

[5] K. Argyraki and D. R. Cheriton, "Network capabilities: The good, the bad, and the ugly," in *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*, 2005.

[6] F. Baker and P. Savola, "Ingress Filtering for Multihomed Networks," RFC 3704 (Best Current Practice), IETF, Mar. 2004. [Online]. Available: https://www.ietf.org/rfc/rfc3704.txt

[7] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker, "Off by Default!" in *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*, 2005.

[8] C. Basescu, R. M. Reischuk, P. Szalachowski, A. Perrig, Y. Zhang, H.-C. Hsiao, K. A., and U. J., "SIBRA:Scalable Internet Bandwidth Reservation Architecture," in *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*, 2016.

[9] A. Bender, N. Spring, D. Levin, and B. Bhattacharjee, "Accountability as a Service," in *Proceedings of the USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, 2007.

[10] J. Brzozowski, J. Leddy, C. Filsfils, R. Maglione, and M. Townsley, "Use Cases for IPv6 Source Packet Routing in Networking (SPRING)," Tech. Rep., 2018.

[11] F. Cai, Y. Chen, D. Wu, and Z. Fang, "VxLAN Security Implemented using VxLAN Membership Information at VTEPs," May 26 2016, US Patent App. 14/549,915.

[12] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking Control of the Enterprise," in *Proceedings of the ACM Conference on SIGCOMM*, 2007.

[13] M. Casado, T. Garfinkel, A. Akella, M. Friedman, D. Boneh, N. Mckeown, and S. Shenker, "SANE: A Protection Architecture for Enterprise Networks," in *USENIX Security*, Aug. 2006.

[14] P.-W. Chi, C.-T. Kuo, J.-W. Guo, and C.-L. Lei, "How to Detect a Compromised SDN Switch," in *Proceedings of the IEEE Conference on Network Softwarization (NetSoft)*, 2015, pp. 1–6.

[15] R. B. da Silva and E. S. Mota, "A Survey on Approaches to Reduce BGP Interdomain Routing Convergence Delay on the Internet," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2949–2984, 2017.

[16] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an Elastic Distributed SDN Controller," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 7–12, 2013.

[17] L. Dunbar, W. Kumari, and I. Gashinsky, "Practices for Scaling ARP and Neighbor Discovery (ND) in Large Data Centers," RFC 7342 (Informational), IETF, Aug. 2014. [Online]. Available: https://www.ietf.org/rfc/rfc7342.txt

[18] C. Filsfils, S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir, "Segment Routing Architecture," Tech. Rep., 2018.

[19] K.-T. Foerster, M. Parham, M. Chiesa, and S. Schmid, "TI-MFA: Keep Calm and Reroute Segments Fast," in *IEEE INFOCOM Workshops (INFOCOM WKSHPS)*. IEEE, 2018, pp. 415–420.

[20] A. J. Gonzalez, G. Nencioni, B. E. Helvik, and A. Kamisinski, "A Fault-tolerant and Consistent SDN Controller," in *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, 2016, pp. 1–6.

[21] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4. ACM, 2009, pp. 51–62.

[22] R. E. Haeni, "Firewall Penetration Testing," Technical report, The George Washington University Cyberspace Policy, Tech. Rep., 1997.

[23] T. Holterbach, E. C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, and L. Vanbever, "Blink: Fast Connectivity Recovery Entirely in the Data Plane," in *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2019, pp. 161–176.

[24] S. Homma, H. Nishihara, T. Miyasaka, A. Galis, V. RAM OV, D. Lopez, L. Contreras-Murillo, J. Ordonez-Lucena, P. Matinez-Julia, L. Qiang, R. Rokui, L. Ciavaglia, and X. de Foy, "Network Slice Provision Models," 2019. [Online]. Available: "https://datatracker.ietf.org/doc/draft-homma-slice-provision-models/"

[25] P. Jain, M. Mehta, S. Jain, and Y. Yang, "Microsegmentation in Heterogeneous Software Defined Networking Environments," Nov. 23 2017, US Patent App. 15/159,379.

[26] N. Katta, H. Zhang, M. Freedman, and J. Rexford, "Ravana: Controller Fault-tolerance in Software Defined Networking," in *Proceedings of the ACM SIGCOMM Symposium on Software Defined Networking Research*, 2015, p. 4.

[27] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A Distributed Control Platform for Large-scale Production Networks." in *Proceedings of the USENIX Conference on Operating Systems Design and Implementation (OSDI)*, vol. 10, 2010, pp. 1–6.

[28] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed Internet Rrouting Convergence," *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 4, pp. 175–187, 2000.

[29] T. Lee, C. Pappas, D. Barrera, P. Szalachowski, and A. Perrig, "Source Accountability with Domain-brokered Privacy," in *Proceedings of the ACM Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, 2016.

[30] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically Centralized?: State Distribution Trade-offs in Software Defined Networks," in *Proceedings of the ACM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2012, pp. 1–6.

[31] X. Liu, X. Yang, and Y. Lu, "To Filter or to Authorize: Network-Layer DoS Defense Against Multimillion-node Botnets," in *Proceedings of the ACM Conference on SIGCOMM*, 2008.

[32] X. Liu, X. Yang, D. Wetherall, and T. Anderson, "Efficient and Secure Source Authentication with Packet Passports," in *Proceedings of the USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, 2006.

[33] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks

over Layer 3 Networks," RFC 7348 (Informational), IETF, Aug. 2014. [Online]. Available: https://www.ietf.org/rfc/rfc7348.txt

[34] O. Mämmelä, J. Hiltunen, J. Suomalainen, K. Ahola, P. Mannersalo, and J. Vehkaperä, "Towards Micro-segmentation in 5G Network Security," in *European Conference on Networks and Communications (EuCNC 2016) Workshop on Network Management, Quality of Service and Security for 5G Networks*, 2016.

[35] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a Model-driven SDN Controller Architecture," in *Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, 2014, pp. 1–6.

[36] A. Morton, "IMIX Genome: Specification of Variable Packet Sizes for Additional Testing," RFC 6985 (Informational), IETF, Jul. 2013. [Online]. Available: https://www.ietf.org/rfc/rfc6985.txt

[37] L. Muller and J. Soto, "Micro Segmentation for Dummies," Tech. Rep., 2015.

[38] J. Naous, M. Walfish, A. Nicolosi, D. Mazières, M. Miller, and A. Seehra, "Verifying and Enforcing Network Paths with ICING," in *Proceedings of the ACM Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, 2011.

[39] D. Naylor, M. K. Mukerjee, and P. Steenkiste, "Balancing Accountability and Privacy in the Network," in *Proceedings of the ACM Conference on SIGCOMM*, 2014.

[40] A. Nygren, B. Pfaff, B. Lantz, B. Heller, C. Barker, C. Beckmann, D. Cohn, D. Malek, D. Talayco, D. Erickson *et al.*, "Openflow Switch Specification Version 1.5.1," *Open Networking Foundation, Tech. Rep.*, 2015.

[41] A. Panda, C. Scott, A. Ghodsi, T. Koponen, and S. Shenker, "Cap for Networks," in *Proceedings of the ACM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2013, pp. 91–96.

[42] B. Parno, D. Wendlandt, E. Shi, A. Perrig, and Y.-C. Hu, "Portcullis: Protecting Connection Setup from Denial-of-Capability Attacks," in *Proceedings of the ACM Conference on SIGCOMM*, 2007.

[43] S. Peng, R. Chen, and G. Mirsky, "Packet Network Slicing using Segment Routing," 2019. [Online]. Available: "https://datatracker.ietf.org/doc/draft-peng-lsr-network-slicing/"

[44] A. Perrig, P. Szalachowski, R. M. Reischuk, and L. Chuat, *SCION: A Secure Internet Architecture*. Springer International Publishing, 2017.

[45] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed Multi-domain SDN Controllers," in *Proceedings of the IEEE Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–4.

[46] D. Project, "Data Plane Development Kit," https://dpdk.org, Nov 2019, retrieved on 1/2020.

[47] B. Raghavan and A. C. Snoeren, "A System for Authenticated Policy-Compliant Routing," in *Proceedings of the ACM Conference on SIGCOMM*, 2004.

[48] T. Sasaki, C. Pappas, T. Lee, T. Hoefler, and A. Perrig, "SDNsec: Forwarding accountability for the SDN data plane," in *Proceedings of the International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2016, pp. 1–10.

[49] H. Shah and A. Ghanwani, "ARP Broadcast Reduction for Large Data Centers," 2011.

[50] W. Simpson, "TCP Cookie Transactions (TCPCT)," RFC 6013 (Experimental), IETF, Jan. 2011. [Online]. Available: https://www.ietf.org/rfc/rfc6013.txt

[51] M. Soliman, B. Nandy, I. Lambadaris, and P. Ashwood-Smith, "Exploring Source Routed Forwarding in SDN-based WANs," in *2014 IEEE International Conference on Communications (ICC)*, 2014, pp. 3070–3075.

[52] S. Son, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Model Checking Invariant Security Properties in OpenFlow," in *Proceedings of the IEEE International Conference on Communications (ICC)*, 2013, pp. 1974–1979.

[53] J. Song, R. Poovendran, J. Lee, and T. Iwata, "The AES-CMAC Algorithm," RFC 4493 (Informational), IETF, Jun. 2006. [Online]. Available: https://www.ietf.org/rfc/rfc4493.txt

[54] X. Su, S. Bryant, A. Farrel, S. Hassn, W. Henderickx, and Z. Li, "SR-MPLS over IP," 2019. [Online]. Available: "https://datatracker.ietf.org/doc/draft-ietf-mpls-sr-over-ip/"

[55] X. Sun, Y.-W. Sung, S. D. Krothapalli, and S. G. Rao, "A Systematic Approach for Evolving VLAN Designs," in *IEEE INFOCOM*, 2010, pp. 1–9.

[56] C. A. Sunshine, "Source Routing in Computer Networks," *ACM SIGCOMM Computer Communication Review*, vol. 7, no. 1, pp. 29–33, 1977.

[57] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed Control Plane for Openflow," in *Proceedings of ACM Internet Network Management Conference on Research on Enterprise Networking*, vol. 3, 2010.

[58] VMWare, "Data Center Micro-Segmentation: A Software Defined Data Center Approach for a Zero Trust Security Strategy," Tech. Rep., 2014.

[59] A. Yaar, A. Perrig, and D. Song, "SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2004.

[60] S. T. Zargar, J. Joshi, and D. Tipper, "A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks," *Communications Surveys & Tutorials, IEEE*, vol. 15, no. 4, 2013.

[61] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen, "SCION: Scalability, Control, and Isolation on Next-Generation Networks," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2011.

[62] H. Zhou, C. Wu, C. Yang, P. Wang, Q. Yang, Z. Lu, and Q. Cheng, "SDN-RDCD: A Real-time and Reliable Method for Detecting Compromised SDN Devices," *IEEE/ACM Transactions on Networking*, vol. 26, no. 5, pp. 2048–2061, 2018.