Fast IP Hopping Randomization to Secure Hop-by-Hop Access in SDN

Sang-Yoon Chang, Younghee Park, and Bhavana Babu Ashok Babu

***Abstract*—Moving target defense (MTD) is useful for thwarting network reconnaissance and preventing unauthorized access. While previous research in MTD focuses on protecting the endnodes, we leverage software-defined networking to implement MTD on the data-plane switches, which significantly decreases the controller communication overhead and enables quicker defense response to reduce the attack impact. This paper not only randomizes the IP addresses for MTD but also uses the IP addresses for synchronization across the nodes in the networking path by generating hash-chain-based synchronization signatures. Our scheme is practical as it builds on and encodes the existing IP addresses for randomization to construct a modular solution independent to the routing/flow rule implementation and does not incur additional networking overhead except for the seed distribution (which can occur offline). Our scheme is also effective (the attacker's required cost to achieve timely network reconnaissance increases by more than an order of magnitude than the previous state-of-the-art having the controller actuate the MTD randomization) and scalable (the relative overhead cost of our scheme becomes smaller as the network grows). We analyze our scheme and implement and experiment it on an Open vSwitch-based testbed and on CloudLab to validate these properties.**

***Index Terms*—Moving target defense, access randomization, network synchronization, IP address control, software-defined network (SDN), data plane security, network security.**

I. Introduction

Computer networking uses the IP protocol, in which the IP addresses are used to address the networking nodes and to route/forward the packets. IP addresses are generally arbitrarily chosen and assigned to each networking nodes, and there is significant freedom in choosing the IP addresses.

We propose encoding information to the IP address values and introduce additional functionalities using the IP address field of the networking header. More specifically, we use the IP addresses to construct moving target defense (MTD) on the data-plane switches (which forward/route the networking packets on the endnode's behalf), providing an efficient security measure to prevent the unauthorized access of the networking paths. In contrast to the network security measures at the network perimeter based on filtering or intrusion detection/prevention, MTD provides a defense-by-depth measure which can be effective even against the attackers who already breached the network-perimeter defense and have the access to the links in the networking forwarding paths. MTD varies the networking parameters (such as the address, the medium access, and the networking configuration) to build path integrity. The authorized nodes who share the key knows the varying pattern whereas the unauthorized parties who do not share the key do not know the MTD pattern. MTD is useful in preventing the unauthorized attackers from achieving *network reconnaissance*, which is the process of investigating and acquiring networking-relevant knowledge for vulnerability discovery and is often the pre-requisite for passive eavesdropping and active injection threats (e.g., denial of service (DoS) injections), because MTD significantly increases the unauthorized attacker's cost in probing and achieving network reconnaissance.

While the MTD technique is generally considered effective against unauthorized attackers and used in many contexts (such as in configuration randomization, memory protection, and wireless/spread-spectrum, as discussed in greater detail in Section II), a major challenge for deploying MTD defense is the overhead cost of implementing and executing the MTD on the legitimate parties holding the key. While the advantage from the key decreases the MTD effort compared to the attackers without the key, the MTD implementation still incurs overhead compared to having no MTD (static configuration) and such overhead may be large enough to limit its practicality in some applications. Prior literature proposes building MTD defense in software-defined network (SDN), in which the trusted SDN controller actuates the MTD on the data-plane endnodes via explicit OpenFlow-based northbound communications controlling the MTD execution [3], [4]. To address the overhead issues, we significantly improve the prior work to construct MTD defense in SDN by spreading the information from the northbound communications (so that one controller communication can start a chain and be used for many MTD updates) and by having the data-plane switch nodes execute the MTD.

In addition to making it lightweight for greater practicality, our IP-address-based access randomization scheme is more effective than the prior literature, because we implement the defense on the switches in addition to the destination endnodes (in contrast, prior work only protects the endnodes) in order to

Manuscript received September 23, 2018; revised December 12, 2018; accepted December 23, 2018. Date of publication December 27, 2018; date of current version March 9, 2019. This work was supported by the grant from National Science Foundation (NSF-DGE-1723804). This work is an extended version of the short paper published at IEEE/IFIP NOMS, Istanbul, Turkey, April, 2016 [1]. The associate editor coordinating the review of this paper and approving it for publication was M. Conti. *(Corresponding author: Younghee Park.)*

S.-Y. Chang is with Computer Science, University of Colorado Colorado Springs, Colorado Springs, CO 80918 USA (e-mail: schang2@uccs.edu).

Y. Park and B. B. Ashok Babu are with Computer Engineering, San Jose State University, San Jose, CA 95192 USA (e-mail: younghee.park@sjsu.edu; bhavanababu.ashokbabu@sjsu.edu).

Digital Object Identifier 10.1109/TNSM.2018.2889842

limit the attack impact in the number of links/switches affected by the attacker's injections.

Because of the involvement of the switches along the forwarding path, we also use the IP addresses for synchronization to ensure that the source-destination endnodes and the intermediate switches are in the same MTD phase. We propose synchronization signatures, which are generated from a one-way hash chain and will replace the IP addresses in the networking header field.

Our schemes for access randomization and for synchronization are unobtrusive and modular to the rest of the networking operations, for example, it generally applies across the networking implementations and does not affect or require change to the routing protocols or the routing/flow-rule establishment. In addition, because the information is encoded in the IP address field, it has no overhead in the data networking throughput. In other words, our schemes for randomization and synchronization piggyback on the data communications and do not require separate communications. (The only extra communication needed is for the seed distributions to reset the randomization/synchronization pseudo-random generation (PRG) chains, which overhead can be amortized because we can control the chain length to spread the information from that communication to multiple randomization/synchronization instances.) Therefore, our overhead/cost evaluation includes the computational latency and computational resources of the MTD/IP hopping and analyze the control/set-up overhead in Section VIII.

The lack of real-time networking overhead (no additional communication packets) distinguishes our work from prior research implementing MTD based on real-time controller communications [3], [4]. Our evaluation shows that the attacker cost to achieve network reconnaissance increases by more than an order of magnitude against our scheme than the prior state-of-the-art MTD randomization in SDN. We also show that the overhead costs of our schemes (e.g., the latency for the PRG and the writing of the IP address field) are lightweight, e.g., one to two orders of magnitude smaller than the data communication latency costs in our implementation experiments.

We build our scheme on *software-defined networking* (SDN) architecture, which de-couples the decision-making at the control plane and the lower-level implementation of forwarding at the data plane. SDN offers a centralized infrastructure where the controller resides in the control plane and supports communication between the control and the data plane, e.g., via OpenFlow, and is typically used for intra-networking applications (where the network is governed and operated by a network manager). SDN thus facilitates the distribution of the chain seeds in our work, in addition to its typical functionality of establishing the routing of the networking packets.

To summarize, we make the following contributions in this paper. First, we design an algorithm and a protocol to construct a switch-centric MTD for securing the access of the networking path. Second, we design a synchronization scheme to allow synchronous MTD for the nodes in the networking path; the synchronization uses the networking-layer field so that it is appropriate and minimizes the overhead to the intermediate switches. Third, we theoretically analyze the MTD and the synchronization schemes and implement them and evaluate the effectiveness, performances, and the cost overheads. More specifically, for MTD, we focus on its effectiveness against cognitive and reactive attacker and compare with the prior approach of the controller-driven MTD randomization; for synchronization, we investigate the cost overheads and the scalability as the nodes increase in the networking path between the source and the destination endnode. Section II further describes our novel contributions and contrasts them to the prior research work.

The rest of the paper is organized as follows. We review the relevant prior work and highlight our contributions beyond the state of the art in Section II. Afterward, we describe the system and threat model in Section III. Our access randomization scheme is presented in Section IV and analyzed in Section V; our synchronization scheme is presented in Section VI and analyzed in Section VII. Section VIII validates its effectiveness and the security cost overhead using an Open vSwitch-based prototype and a CloudLab-based prototype. Lastly, Section IX concludes our paper.

II. Related Work and Our Contributions

Networking security has traditionally placed heavier focus on the defense at the network perimeter, e.g., filtering and intrusion detection/prevention, so that attackers do not have the link access within the system. In contrast, we are motivated to construct a defense-by-depth measure to build security even when the attacker compromised the network boundary and have the link access within the network. Our work is inspired by prior work in wireless security to secure the communication link access and is similar in its goal in preventing the unauthorized access at the first hop. In Section II-A, we discuss the body of work in wireless security that inspired our work (spread spectrum), related work in wired networking that adopts randomization, and then other relevant work that implements security at the data-plane switches. In Sections II-B and II-C, we highlight our novel contributions beyond prior work.

A. Related Work

1) Spread Spectrum: As wireless communication is inherently broadcast and anybody equipped with a radio can access the communication, researchers and military experts have long been working on securing access. In wireless medium access control, MTD is used for building link resiliency against jamming [5]–[8] and securing confidentiality against eavesdropping [9], [10]. Spread spectrum is a popular approach to realize MTD [11], [12], and the access parameters are time, frequency, and code. For instance, frequency hopping spread spectrum (FHSS) in wireless communications dynamically varies the carrier frequency and the frequency band in order to prevent the attacker's access, and fast hopping with short hopping duration thwarts reactive attackers who sense the channel use and adapt their strategy in real-time [13]. We emulate the following two properties of spread spectrum: first, the hopping is dynamic and quick (making it difficult for attackers to react to the protocol and adjust their parameters); second, the hopping is proactive and autonomous (as opposed to being reactive and triggered). These properties distinguish our work from prior work that deploys MTD at the network layer, which work we discuss later.

The aforementioned spread spectrum work assumes single-hop network, i.e., destination nodes are directly reachable by source nodes, and is typically applied to secure the last hop of communication link, e.g., between the base station and the mobile endnode. In contrast, to adopt MTD techniques is more challenging for communication that consists of multiple hops, in which case, the links across the hops must be synchronized, and the implementation spills beyond the source-destination pair (who are directly involved in the communication payoffs) and to the intermediate switches (who merely relay the packets using the networking header fields).

2) Address Space Randomization: We use the network address as our randomization parameter. Dynamic host configuration protocol (DHCP) is widely deployed for IP control but is not designed for security. In the security context, researchers have used dynamic addressing to prevent network reconnaissance [14]–[19]. An adaptation of network address translation (NAT) has also been proposed [20]. However, these prior work in address space randomization build security only at the endnodes (and not at the intermediate switches) and thus the attackers can still access the path until the traffic reaches the endnodes. In contrast, we aim to prevent the attacker access early in the forwarding path. Beyond networking, address randomization is also used to protect data and memory access against software vulnerabilities such as buffer overflow [21], [22].

3) Symmetric Encryption: Our scheme is similar to cryptographic encryption in its goal to scramble the IP address and make them unavailable without the key (the PRG seeds in our case). However, efficiency presents a critical challenge in using cryptographic encryption for our purpose. Symmetric encryption such as DES or AES (which are generally superior in processing efficiency than public-key encryption) protects application-layer information and is not appropriate for network layer, e.g., symmetric encryption protects the message while the network header is in cleartext (not encrypted). The switches, on the other hand, only process up to the networking headers and are thus inappropriate to adopt encryption as it is designed. In addition, the dynamic nature of our scheme would require heavy loads and overheads in key/entropy generation and distribution if trying to encrypt the IP addresses using symmetric encryption. Key/entropy generation and distribution present challenges in cryptography in general, e.g., one-time pad ensuring perfect secrecy against cryptanalysts has limited use in real-world practice due to these challenges. Due to these challenges, we design a scheme which is more appropriate to network-layer processing than application-layer encryption.

4) Software-Defined Network: Since most intelligence is on the SDN controller, e.g., routing control, much prior work in SDN proposes to offload the security implementation to the controller (e.g., detection and filtering based on flow analyses [23]–[25] and traffic analyses [26]–[29]) and build on secure communication between the switches and the controller [30], [31]. In specific, Kampanakis *et al.* [4] sketches the adoption of the address space randomization technique on the SDN controller, and OF-RHM [3] describes an instantiation of such approach.

While we also use SDN for routing control/establishment, our work is different from these prior work in SDN-based MTD work in three aspects. First the security implementation scope is different, as these prior work in address randomization are implemented only at the endnodes and need to tolerate the attacker traffic until it reaches the destination endnode (much like the previously mentioned network address randomization work in non-SDN contexts in Section II-A2. Second, they require substantial overhead as they require interacting with the controller whenever the network address changes; in contrast, our work spreads the information entropy delivered from the controller and significantly reduces the controller communication overhead; to achieve this, we delegate some intelligence (computations) to the switches (others have also proposed to offload responsibilities to the data-plane switches to achieve other security properties [30], [32]). Third, in contrast to prior work, our work is proactive in hopping/randomization and is resilient to adaptive attackers that sense the traffic flow.

5) Security at Routers/Switches: Our scheme implements security at the data-plane switches and aims to prevent unauthorized access at the first node that encounters the access. Other researchers proposed orthogonal approaches to achieve the same goal, e.g., against DoS, and adopt two classes of approach. First, the filter-based approaches detect the unauthorized access based on traffic analyses (e.g., using source identities and anomalous behaviors) and assume the ability to distinguish the unauthorized traffic and the authorized traffic [33], [34]. Second, the capability-based approaches involve handshaking (typically with the destination endnode or a dedicated server) and explicit authorization for the path access [35]–[39]. While our contribution shares similarities with such protocols in its objective and the fact that security is implemented at the switches, we take a different approach and adopt MTD; our schemes also does not require the classification of malicious traffic nor explicit handshaking per flow.

B. We Implement Randomization at the Switches

We implement our randomization and synchronization schemes at the data plane and on the switches and thus the implementation scope is different from the endnode-centric prior work, described in Section II-A. In other words, prior research builds security at the endnodes and their communication with the network representative (e.g., SDN-controller or a DHCP server); the endnodes' operations are orthogonal to the data-plane networking and there is no defense at the data-plane forwarding/routing. In contrast, our scheme implementation is at the data-plane forwarding; to provide a modular solution (while keeping the rest of the layers intact), we construct virtual switches at the nodes in Section IV. Furthermore, since our scheme builds protection on the path itself and at the hop-level, it enables quicker response and prevents attackers from overwhelming the switches and multiple communication links; our scheme is also effective against direct or indirect link-targeted DoS [40], [41].

C. We Make Use of the IP Addresses

While it is typical to select random and arbitrary IP addresses for addressing/routing, we encode additional information to IP addresses so that they can be useful for randomization and synchronization. The nodes address the source and the destination nodes with the correct dynamic IP addresses (which are the outputs of the IP hopping and random to the attacker) to access the packet forwarding path in randomization, and they broadcast explicit reference packets with pseudo-random IP addresses for synchronization.

The randomization and the synchronization use separate and independent pseudo-random generation (PRG). Therefore, we introduce two orthogonal pseudo-random block chains.[1] Our implementation uses linear-feedback-shift-register (LFSR)-based pseudo-random generator (PRG) for the randomization and hash-based PRG for the synchronization chain. While any PRG can be used for the randomization and synchronization chains in principle, our choice of the PRG implementation is influenced by the LFSR's/hash's respective use in each applications. LFSR is popularly used for real-time access randomization (e.g., spreading spectrum) as the LFSR register itself stores the current LFSR PRG output/state and the computation is quick and based on bit-by-bit XOR, while hash-based chain highlighted by its one-way property is popularly used for one-time random token/password generations (e.g., S/Key [42], [43]). These existing functions are well-studied in their security properties and are well-adopted for security applications.

III. System Model

A. System Assumptions

We consider a connected multi-hop network and assume IP-protocol-based forwarding. Our scheme is independent to the routing decisions/implementations and generic across the routing protocols as long as they are IP-driven, so we assume that they are given. While our scheme is generally applicable to the routing protocols (including those that are primarily driven by the destination-node address, e.g., Routing Information Protocol (RIP) and Open Shortest Path First (OSPF)), the switches (executing routing and forwarding) check and process both the source IP address field and the destination IP address field, which are included in the networking header by IPv4 and IPv6 standards. Our scheme makes use of both source IP address and the destination IP address for randomization.

We build our scheme on the SDN architecture (where a controller establishes and communicates the routing decision and the switches forward traffic accordingly) and focus on the intra-domain communication applications (e.g., governed by a single operator). We also leverage a public key infrastructure (PKI) for the distribution of the seed values that will activate our scheme. More specifically, we have three seeds: two for randomization hopping (IP seed and PRG seed) in Section IV and the hash seed for synchronization in Section VI. The security of our work relies on the PRG seed and the hash seed. The SDN controller distributes the seed values to the data-plane nodes. We assume secure control communication between the control and the data plane, e.g., [30] and [31] (for example, by using asymmetric encryption for confidentiality and digital signature for integrity) and secure controller, e.g., [32] and [44], and we rather focus on the threats on the data plane.

Our scheme can be used for multiple flows where each flow is for a sender-destination pair. For multiple flows, our scheme introduces multiple chains for randomization in Section IV (one chain per flow and keeping tracking of PRG_{seed} and *t* per flow) and multiple chains for the synchronization in Section VI (one chain per flow) . However, for storage, while the synchronization requires storing multiple signatures per flow, the randomization requires the switches and the endnodes to only keep track of PRG_{seed} and t per flow and do not need to store beyond those values for each flow, assuming that the routing/flow rules are given as described earlier (e.g., storing and keeping track of the endnodes' IP addresses to identify the flow and the corresponding routing rules). For simplicity, we assume a single flow when describing our scheme, i.e., the scheme is presented in a per-flow perspective.

B. Threat Model

An attacker accesses the networking path without the legitimate authorization for the access. We also consider compromised network with the attacker being physically connected to the network and having bypassed the perimeter filter and gateway, for example, the attacker is connected to the network from within. Our scheme defends such attack by thwarting network reconnaissance. Network reconnaissance is to learn about the victim network and its vulnerability and is a pre-requisite for many access-based attacks, such as eavesdropping, injection-based DoS, and malware/exploit delivery.

We do not consider the orthogonal threat of disrupting the routing and the forwarding process, which requires a stronger attacker with increased attacker requirements than our threat model. Even though the attacker has access to the path links, it does not control the switches, e.g., to take active measures in the routing and forwarding process. For example, if a node along the forwarding path is compromised and disrupts the forwarding process, e.g., re-routing and packet dropping, then the routing protocol can choose another forwarding path which does not include that misbehaving/faulty node. Such threat model is more difficult to implement than ours (increasing the attack barrier) as it requires the compromise of a switch which is a part of the networking infrastructure, as opposed to just

[1] Our PRG chains share similarities with blockchain technology in the ledger data structure (based on one-way hash chains) and that the functions are computed in a distributed set of nodes (in our case, the nodes along the forwarding path). However, our PRG chains are different from the blockchain technology because the computations, given the seeds, are deterministic and therefore the controller (providing the seeds) drives the consensus process. In contrast, blockchain achieves consensus in a distributed manner and often involves a random process.

having connection to a switch or a networking link. Thus, we do not consider compromised switches that have active control on the forwarding and routing operations.

We consider an advanced attacker that is dynamic and adaptive. It can both passively monitor/analyze the traffic and actively send queries and probe for network reconnaissance; afterward, the attacker can use the information to facilitate their unauthorized access. (An analogy to wireless communication is a reactive jammer where the jammer performs cognitive-radio-like sensing of the spectrum to decide where to focus its jamming for maximum impact.) The attacker's traffic sensing and the adaptation of the access strategy is done in real time. To counter such sophisticated threat, we proactively and rapidly update the IP address for MTD. If the MTD randomization supports reactive trigger (e.g., IP changes when there is a misbehavior detection) and relies on the prior state of the art in controller-induced randomization, such attacker can indirectly launch a MTD-specific DoS on the control communication by repeatedly and intentionally triggering misbehavior detection and forcing the controller to continuously update the security parameters, overwhelming the network with control communication updates and thus blocking the actual goodput delivery.

IV. Access Randomization Scheme: IP Hopping

We deploy IP randomization across all nodes (endnodes and switches) which are involved in the forwarding path. We generate IP addresses, based on a local seed and a random seed, and use them for randomization on the switches. The SDN controller distributes the seeds and controls the routing operations (including the routing/flow rule updates on the switches), as discussed in Section III-A.

Similarly to some other routing schemes [35], [36], our work can be used to establish priority-based forwarding where the traffic that uses our scheme has higher priority in the path access and forwarding. The low-priority traffic which uses the static IP address without MTD can be processed in an unobtrusive manner to the high-priority traffic (only getting processed when there is no higher-priority traffic) or can be banned altogether, depending on the priority scheme design and implementation. As discussed in Section III, we assume a single flow and that the IP randomization is enabled (e.g., high priority traffic) in this section for simplicity in presentation.

To provide a modular solution which can be deployed across the networking/routing implementations, our scheme provides a transformation between the original IP address (used for routing and other networking decisions) and the IP address used for MTD randomization. The process of converting from the original to the randomized IP address is called *IP hopping* and the reverse process is called *IP de-hopping*, and both the source IP address and the destination IP address goes through IP hopping/de-hopping processes. In addition, we construct virtual switches within the nodes, so that the packets are processed for IP hopping after the network-layer operations and before the physical-layer mappings. The operations of the virtual switches for endnode are described in Figure 1 while those for the endnodes and the switches as well as the hopping and the de-hopping processes are described in Figure 2. The operations of the virtual switches at the endnodes are the same as those at the switches, except that the source node does not perform IP de-hopping and the destination node does not perform IP hopping.

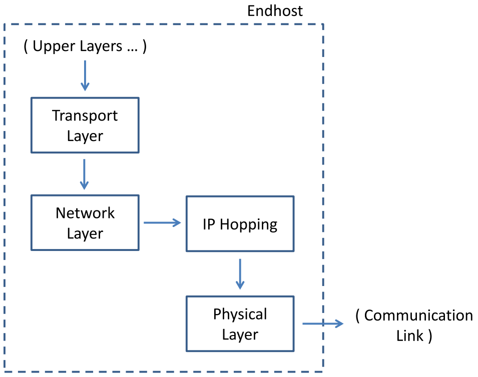

Fig. 1. Operational flow for endnodes (the source node in this case) to highlight the modular design. Our contribution lies at the IP Hopping/De-Hopping module between the other network layer operations and the physical-layer.

A. IP Hopping: Address Generation and Randomization

Unlike other schemes which implement security at the controllers [23], [24], [26]–[29] including the work that builds MTD [3], [4], we offload the control computation to the data plane and on the switches in our work. However, the operations on the switches are marginal in intelligence as they perform routine protocol and computations. The nodes along the forwarding path (both the endnodes and the switches) locally implement pseudorandom generator (PRG), which is uniform and agreed across the nodes. Once the controller provides the initial input, the switches iteratively compute the PRG function to construct a PRG chain. Each PRG function output is used at a time, and the next iteration for computing the PRG function corresponds to updating.

The endnodes and the switches update the IP addresses from the two seeds: the IP address seed (unique for each nodes) and the PRG seed (provided from the controller and used to drive the PRG). The local IP address seeds of the nodes are static and do not need to be private information, for example, the static IP addresses (which have been used before the activation of our scheme or is used to send the low-priority traffic) can be used as this seed. All nodes involved in the forwarding know each other's IP address seeds. The variation for randomization comes from the PRG; the function and the seeds are uniform across the nodes involved in the path (including the source and the destination endnodes, equipped with hopping-capable virtual switches). Nodes not involved in the path forwarding do not know the PRG seed, and thus the PRG output appears random to them.

The updated IP address (IP_{update}) is a function (f) of the IP seed (IP_{seed}) and the PRG output (*PRG*), which in turn is a function of the PRG seed (PRG_{seed}) and the time in packets (t):

$$IP_{update} = f(IP_{seed}\ ,\ PRG(PRG_{seed}\ ,\ t)) \tag{1}$$

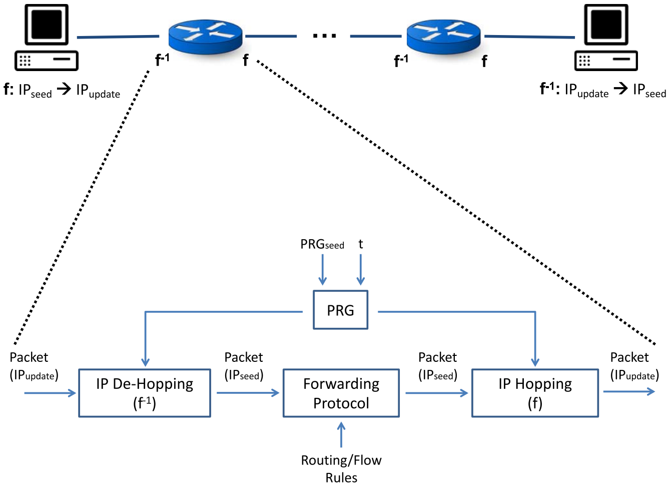

Fig. 2. IP hopping (f) and IP de-hopping (f^{-1}) on the endnodes and the switches. The information/time flows from left (source) to right (destination). The top describes the IP address transformation using hopping and de-hopping of the incoming and outgoing packets, while the bottom shows the transformations in greater detail occurring at the switches.

where both f and *PRG* are some deterministic functions. This IP address transformation is also described in Figure 2 (bottom right). While f^{-1} (given either IP_{seed} and *PRG* output) is easy to compute, PRG^{-1} and *PRG* (without PRG_{seed}) are difficult to compute. While this section focuses on the use of the functions and do not identify the functions, the following section provides concrete functions for IP hopping. The input t corresponds to the time in packets, as opposed to the absolute time, and incrementing/changing t varies the IP update. Section VI describes the synchronization and how t is shared and agreed across the nodes.

B. IP Hopping: Our Implementation

In our implementation, we use Class A private addresses with subnetwork address of 10.X.X.X and subnet mask of 255.0.0.0, leaving us 24 bits for randomization; because the entropy is the same as the size of the host address space. adopting our scheme with IPv6 will significantly increase the randomness and the security strength. We implement the IP hopping by applying the followings: the randomization is at the IP addresses of the nodes; the static IP address, e.g., for low-priority traffic, acts as the IP_{seed}; we use a linear feedback shift register (LFSR)-based[2] PRG (with t increment triggering the shift and $t = 0$ corresponding to no shift and being in the state of PRG_{seed}); and f is cyclic addition for each decimals (representing one byte each). Thus, both f and *PRG* are linear and computationally efficient in our implementation.

The IP seed can be found by reversing the operation and using f^{-1} from IP_{update}, i.e., cyclic subtraction by the PRG output, as described in Figure 2. For the packet at time t, the legitimate users that forward the packets know the PRG output (as they are using the same *PRG*, PRG_{seed}, f, and t). The IP seed information, requiring the correct PRG value, identifies the source and the destination of the networking packets, as we will discuss further in the following section.

We illustrate the f of decimal-based cyclic addition with an example. If a user has a static address of 10.0.240.25 and the PRG output for time $t = t'$ is 10.0.18.25, then the updated IP address is $IP_{update} = 10.0.2.50$, and that user will use the IP address of 10.0.2.50 at time $t = t'$. The least significant byte is derived from (25+25) mod 256 = 50 while the second-least significant byte is from (240+18) mod 256 = 2. Other switch nodes along the networking path also compute PRG output of 10.0.18.25 at $t = t'$ and cyclic-subtract it from the received 10.0.2.50 to retrieve the IP seed of 10.0.240.25.

C. IP De-Hopping & Forwarding

The route is established based on the static IP_{seed} addresses, e.g., the routing table lists the routing rules and policies and is only updated when directed by the SDN controller. The IP header (which is network-layer information and is thus accessed by the switches by design) enables the switches to conduct MTD/path access validation. If the IP_{update} values of the source and the destination IP addresses of the received packet do not get mapped to the correct IP_{seed} from one of the pre-stored flows, then the packet is invalid in MTD.

As depicted in Figure 2, given the routing and the flow establishment, the forwarding nodes locally decrypt the IP randomization by performing f^{-1} with the PRG output to check whether the traffic is addressed correctly (and hence legitimate). Once packets arrive, the forwarding nodes first check the source's and the destination's updated IP address; by computing f^{-1} of cyclic subtraction, the forwarding node learns the IP_{seed}s of the addresses and identify the routing path according to them. If the IP addresses are valid and the corresponding route/path exists in the routing rule, the switch locates the next-hop node based on the IP_{seed} but performs the IP-hopping function to forward the packet with the IP_{update} addresses (IP seed is invisible outside of that node's computation); if the flow/route is not established based on the IP_{seed} addresses, the packet is dropped. All switches supporting our scheme perform this process.

V. IP Hopping Randomization Analyses

A. Hopping Collision for Multiple Flows

We consider multiple traffic flows in this section. If multiple flows coexist and the path carries traffic from multiple sources, there can be collisions in IP_{update} in Equation 1. However, if f given IP_{seed} is injective with *PRG* (i.e., different *PRG* yields distinct $f \mid IP_{seed}$), then we can resolve collision and distinguish packets that are addressed to the same IP_{update} by coupling the information of IP_{update} and IP_{seed}. In other words, if traffic from different flows address the switch with the same IP address at the same t, then the two packets can be distinguished by computing f^{-1} and identifying the IP_{seed} of the respective flows.

B. Security Analyses

Attackers who wish to compromise a forwarding path need to know that the routing path based on the IP_{seed} is established

[2] LFSR is popularly used for military applications such as cryptography and scrambling (e.g., DSSS) and is known to generate maximum entropy against brute-force threats, as it produces maximal length sequences (i.e., if the bit length is l, it produces $2^l - 1$ distinct states before it repeats itself).

in the routing table and the corresponding IP_{update} values of the source and the destination. The security of our scheme relies on the secrecy of PRG_{seed} and consequently the PRG output. To defeat the reactive attackers who observe the packet traffic and compromise the relevant set of IP_{update}, we have the IP update more quickly than the attackers' reaction time. Because the IP generation and updates are being processed within the host (as opposed to via control communication with a separate entity, e.g., the controller [3], [4]), our scheme is fast and significantly increases the attackers' cost, as we will also see in experiments in Section VIII-B.

To protect PRG_{seed}, we use $t > 0$ for the IP randomization to avoid using the plaintext PRG_{seed} as the *PRG* output with the initial packet (to defeat lurking attackers who capture the network-layer information at the first packet). However, we do not rely on the secrecy of *f* and *PRG* themselves; both *f* and *PRG* are public information by Kerckhoff's principle. Our main contribution does not involve building a secure *PRG*, and we rather rely on the security properties of mature and well-adopted functions (for example, it is difficult to compute *PRG* without PRG_{seed} and *t*); because it is widely used for real-time scrambling for security applications, we use LFSR-based scrambling for PRG.

Our scheme is secure against an attacker that monitors the traffic over time and gathers the history of the utilized IP addresses to acquire the PRG_{seed}. For such attacker, linking the individual packets to the flow and the source-destination pair (to learn the per-flow packet sequences) is challenging, especially when multiple users and flows are using the link (where the attacker compromise resides). Even when the attacker is successful in gathering and storing the history of the IP address updates using some side channel information (for instance, it has the assurance that there is only one traffic flowing), the attacker needs to break the PRG to learn the PRG_{seed} and access the path.

VI. Synchronization Scheme

Since the intermediate switch nodes serving the source and the destination keep track of the IP addresses of the source and the destination nodes for access/forwarding path randomization, they need to be synchronized. More specifically, the aforementioned nodes require packet-level synchronization and the agreement of *t* (used for MTD randomization in Section IV) since *t* changes/varies the IP address.

Our synchronization scheme supplements the randomization scheme in Section IV and constructs an explicit network-layer synchronization signature based on IP address control. The signature is embedded in the network layer of the packet carrying data and therefore does not cause additional overhead in communications, in contrast to using separate control packets exclusively for synchronization, e.g., [45].

A. Synchronization Signature Generation Using One-Way Hash Chain

As described in Section II-C, we introduce a separate, independent PRG chain from the one used for randomization in Section IV. While other PRG implementations can also be used for synchronization signature, we use a one-way hash chain for the synchronization signature. We assume two properties of the hash function *h*, which are typical for many cryptographic applications; the hash *h* produces random/pseudo-random output (and the attacker cannot identify a pattern on the output) and the *h* function has one-way property, i.e., it is easy to compute but difficult to solve the reverse computation of h^{-1}. (Our scheme does not require collision resistance as discussed later in this section.) Given a one-way hash function of *h*, $h^{\alpha}(x)$ takes the input *x* and applies *h* on that input α times, for example, $h^2(x) = h(h(x))$.

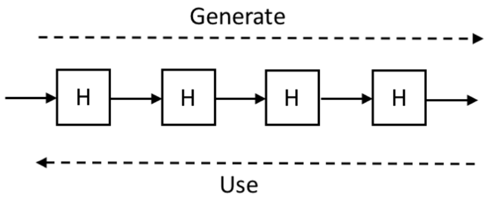

Fig. 3. Hash chain for synchronization signature generation.

For each path (or for each source-destination pair), all nodes in the path (the source, the destination, and the switches) compute the hash chain of length *m* from $h^1, h^2, \ldots, h^m$ by using the input h^0, e.g., $h^1 = h(h^0)$ and $h^2 = h(h^1) = h(h(h^0)))$, where h^0 is shared/communicated to the nodes a priori by the SDN controller. This provides *m synchronization signatures* providing *m* synchronization instances to notify the nodes of the time to update. From the computed and stored hash chain, the nodes use the hash values in the opposite order of its generation, i.e., they use h^m first and then h^{m-1} and then h^{m-2} and so on, as depicted in Figure 3. When all of the *m* synchronization signatures are used, another northbound communication to the SDN controller takes place for a new h^0 and a new chain. *m* therefore provides a design parameter to control the tradeoff between SDN controller overhead, the security impact, and the switch memory. As *m* grows, the synchronization-chain-refresh rate decreases and the controller-communication overhead decreases, but the attacker has greater impact once it does compromise the chain and it requires greater memory on the switches for storing the synchronization signatures (the switch needs to store all the remaining synchronization signatures to be used and therefore stores all *m* signatures when the chain is newly refreshed). Our hash-chain-based PRG chain construction and use is inspired by and is similar to the S/Key one-time password/token generation [42], [43] and TESLA broadcast authentication for wireless network [46], [47].

The use of hash values in the opposite order of their generation in the hash chain provides security due to the one-way property of hash functions. This construction also makes collision irrelevant as the attacker needs to find the exact input to breach the hash chain; our scheme thus does not require the collision resistance property, which is a typical requirement of cryptographically secure hash functions. For example, suppose the current synchronization signature uses h^m, then the attacker breaks the security by finding the exact h^{m-1} and not another h' which generates a collision, i.e.,

$h(h') = h(h^{m-1}) = h^m,\ h' \neq h^{m-1}$, as the attacker using the h' to generate/transmit a false synchronization packet to break the integrity of the synchronization does not match with the pre-stored h^{m-1}. Therefore, our requirement for the hash function has less constraints than the hash functions in other cryptographic applications requiring collision resistance.

The hash values are converted to IP addresses, for example, given the hash outputs, the least significant bits (LSB) of the length of the IP address becomes the IP address. These IP addresses, which we denote with IP_{sync}, are used as the synchronization signature.

B. Synchronization Signature Propagation

Whenever the source node wants to increment t to update/change the IP addresses for the randomization, it embeds the synchronization signature IP address (IP_{sync}) on the source address field of the networking header. In other words, while the destination node is still referred to by the IP_{update} in Equation 1 so that the switches know the routing paths, IP_{sync} replaces the source IP_{update}. Because the path-unique destination's IP_{update} can be used to identify the path and the next-hop switch, the synchronization packet can travel the same path as the data packets and therefore the synchronization packet can still be used for data communication. While the data gets communicated, there will be occasional synchronization signatures (IP_{sync}) embedded in the source IP address field, indicating the synchronization/update instance, while the other non-synchronization data packets use IP_{update} for the source IP address.

For verification of the synchronization packet, each node in the forwarding path checks the destination IP field (IP_{update} from Equation 1 for the access randomization) and the source IP field (IP_{sync} from the synchronization chain); the two IP addresses are from separate PRG chains and are independently generated to each other, and all nodes have the capabilities to compute both IP addresses while they remain random to the attackers.

Once the synchronization packet is transmitted to notify the update in the access randomization, the current synchronization signature can be broadcasted as it is designed for notifying the nodes in the forwarding path. Our construction and the analyses in Section VI-A assume insecure channel and that the attacker can compromise the current synchronization signature once it is transmitted by the source endnode; if the attacker repeats the packet and conduct replay attack, then it actually helps the cause for synchronization. In fact, in the cases of lossy channel, e.g., a faulty link or switch sporadically drops the packets, then the source endnode can repeat the synchronization signatures in the IP address field in order to ensure the delivery of the synchronization signature. We leave such redundancy control to resolve lossy packets as future work; our implementation testbeds in Section VIII had no issues with lossy packets.

VII. Synchronization Overhead Analysis

We analyze the overhead of our IP_{sync}-based synchronization scheme in computation and storage and more specifically its scalability with respect to *m* (the number of randomization updates supported per randomization PRG chain) and *n* (the number of endnodes, and not the switches, which can become source or destination, depending on their communication needs at the time). The networking overhead (additional communication needed) is zero since the synchronization packet can also be used for data communications without needing separate synchronization-exclusive packets. The overhead costs apply to all nodes including the switch nodes.

Our synchronization scheme requires the computations of the hash chain, which involves exactly *m* hash computations per path. At bootstrapping (when *n* nodes get activated), there can be $n \cdot (n - 1)$ paths (there are ${}_nC_2 = n(n-1)$ possible source-destination combinations, and we assume single path per source-destination pair for our scheme), resulting in *mn* $(n - 1)$ hash computations. The initial bootstrapping overhead (when *n* endnodes are first introduced) is thus *mn* $(n - 1)$.

After the bootstrapping, the nodes will generate synchronization signatures as is needed and on a per-path basis and therefore the computational overhead is actually *m* hash computations to generate a *m*-long hash chain. Therefore, per chain update, it requires *m* hash computations. We implement the hash chain and measure the hash computational cost in Section VIII-E in order to demonstrate the relatively low computational overhead of hash chain construction.

While each chain requires *m* hash computations, for storage, all computation outputs for all possible paths need to be stored, and therefore the storage overhead has the the computation cost of O(*mn* $(n - 1)) \sim$ O(mn^2) where O(.) is the big O notation. The exact storage overhead also increases with the length of the IP address; for example, in our implementations in Section VIII, we use 24-bit host address for randomization and synchronization, so the storage overhead is 24 *mn* $(n - 1)$ bits.

Choosing *m* provides a tradeoff between multiple design factors. Increasing *m* decreases the frequency for the northbound controller communication but increases both the computational and storage overhead. *m* also affects the freshness of the chain and smaller *m* may provide stronger security in principle, especially if the PRG generator function has questionable security (currently, in computational security, there is no known attacks that break the PRG generators we use for our instantiation). However, as a constraint for *m* for security purpose, we recommend having *m* smaller than the period of the PRG used for the randomization, e.g., the maximal length of LFSR which is $2^l - 1$ where *l* is the bit-length of the LFSR registers, to avoid repetition (and the corresponding security weakness) in the PRG.

VIII. Implementation Evaluation

To evaluate our scheme, we build an Open vSwitch-based [48] testbed prototype (Section VIII-A describes the prototype implementation) and a CloudLab-based [2] prototype (for analyses with varying networking size and topologies). In a single-flow environment (no other networking traffic), after the seed distribution, we take benchmark measurements

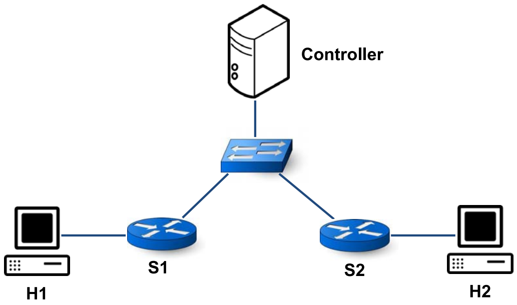

Fig. 4. Our experimental setup in the Open vSwitch testbed.

to show the performance gain of our scheme and the increased security against a reactive attacker, described in Section III-B. Section VIII-B measures the packet delivery latency and compares our IP randomization/hopping scheme with the prior state-of-the-art IP randomization scheme (which involves communication with the controller for every IP update), and Section VIII-C discusses how our fast IP hopping defeats the network reconnaissance attack. In Section VIII-D, we analyze the impact of the size of the network and, more specifically, the length of the path, e.g., the number of intermediate switches between the source and the destination, and in Section VIII-E, we study the computational latency overhead of the synchronization when the forwarding path resets the synchronization-hash chain.

Because we study the scalability and the dependence on network topology, we implement our scheme on CloudLab [2]. CloudLab is an open-cloud platform to provide Infrastructure as a Service (IaaS), supported by National Science Foundation (NSF), and enables greater control in the networking topology. Since the performance values are sensitive to the system implementation details [49], [50], e.g., we also see different measurement values between the Open vSwitch- and CloudLab-based testbeds, we describe the performance in gains over reference measurements. The delay measurements are taken over 10,000 samples for both OVS-based testbed and the CloudLab environment; this number of samples provided a small confidence intervals from the mean values displayed in this section and support the statistical significance of the behavior presented in this section.

A. Open vSwitch-Based Prototype Implementation

To construct a local network for our experiment, we use five computers (Intel Core 2 Duo, 2×2.20GHz CPU with 2.0GB RAM) and an Ethernet switch for the physical devices. There are two endnodes (the source and the destination), two switches, and a controller server; each computer emulates distinct roles. The switches are software-based in Open vSwitch 2.3.0 [48], and the controller is based on OpenDaylight Helium supporting OpenFlow 1.3. Unlike the switches, the physical Ethernet switch device does not support any virtualization and has the sole purpose of bridging the wired connections. Figure 4 depicts the network topology and the roles of the computer hosts; we focus evaluating our scheme given a forwarding path.

We implement a prototype for our scheme. As PRG has comparable data format as IP addresses, we use DHCP protocol for the PRG seed distribution with the DHCP server implemented at the controller. Then, the nodes locally randomize their IP addresses, as described in Section IV-B, and send packets with the updated IP addresses. Only when the IP addresses are correct do the packets go through, as discussed in Section IV-C. For example, when the host H1 directs its packets to other hosts with incorrect IP addresses, e.g., because it is unauthorized and does not have the correct IP update, then the packets get dropped at the next hop at S1.

B. IP Hopping Randomization vs. Controller-Driven Randomization

We compare our scheme to a baseline reference. The *baseline* scheme is to involve the SDN controller for the IP randomization of the nodes, which is adapted from the state-of-the-art mechanism to build MTD on the hosts in SDN [3], [4]. However, the baseline scheme differs from these prior work in that they randomize the addresses of all nodes on the forwarding path as opposed to just the endnodes.

The gain of our scheme over the baseline comes from the fact that the randomization operations are done locally within the nodes as opposed to involving the controller (the baseline). After the PRG seed is distributed to the endnodes and the switches, we generate 64-byte ICMP packets (ping), apply randomization for each packet, and compare the packet latency between the two different MTD approaches. We also isolate the operations and measure the latency when the randomization/update is done locally via LFSR computation and when it is done via the interactions with the controller. The latency performance for our scheme is 0.2680 milliseconds per packet delivery (averaged over 10,000 measurements) and the LFSR computation (which output will be used for the IP update) takes 14.58 nanoseconds (averaged over 10 cycles or $10\times(2^{24}-1)\approx 1.678\times 10^8$ IP addresses), which accounts for a marginal 0.0054% of the total packet latency. On the other hand, the controller-involved baseline scheme has an average latency of 3.7159 milliseconds per packet delivery where the controller overhead for IP update accounts for 93% of the latency. The latency distribution of our measurements are shown in Figure 5. Thus, our scheme compares favorably to the baseline, as the attacker needs to improve its reaction time by more than an order of magnitude to successfully breach the updated IP address.

In addition, Figure 5 shows greater variance/randomness in the measurements when the controller is involved (baseline) than having the randomization performed within the nodes (our scheme) even in our controlled lab environment; the baseline involves multiple nodes for MTD execution while our scheme involves intra-node computation. This corroborates with our findings that the variance increases as there are greater number of nodes involved in Section VIII-D.

C. IP Hopping Against Attacker Network Reconnaissance

To understand the attacker's perspective against our scheme, we instantiate a network reconnaissance threat by having a

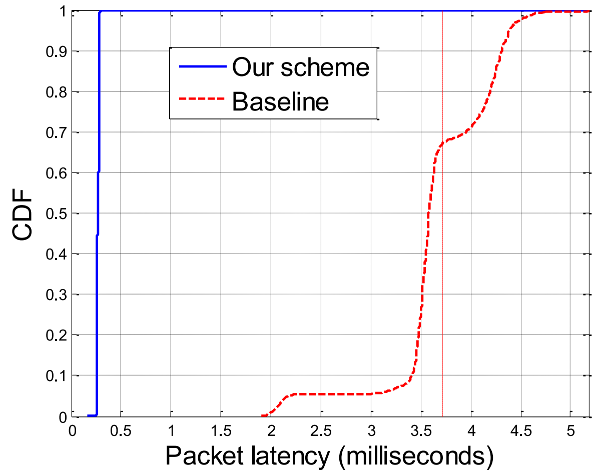

Fig. 5. The measurement distribution in cumulative distribution function (CDF) of our scheme and the controller-driven baseline. For baseline (with greater variance), we also show the average in a vertical line.

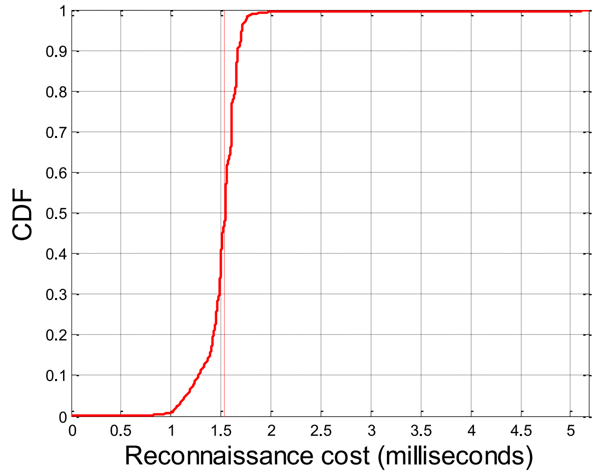

Fig. 6. The measurement distribution in cumulative distribution function (CDF) of attacker's cost in achieving network reconnaissance. The horizontal axis is aligned to Figure 5.

malicious node inquire for the node's IP address via traceroute once it encounters the traffic. This approach provides an immediately feasible implementation of the attack and provides an estimate of the attacker's cost in our experiment. Further analyses or optimization of such attack implementation (e.g., enabling traceroute is not compulsory, although often enabled by default, and the switch can decide not to engage with the attacker) or other attack implementation approaches are not the focuses of our work and are out of scope of this paper. As discussed in the following, our scheme can defend against such network reconnaissance attack.

Figure 6 shows the delay cost of the attacker, which is 1.5462 milliseconds on average. In contrast, our scheme costs 14.58 nanoseconds for IP generation, as the update is locally computed within the node (without requiring communication with outside of that node), and the average packet latency is 0.2680 milliseconds, as discussed in Section VIII-B. Therefore, the attacker's reconnaissance delay is more than five times greater than the packet delivery latency when using our scheme. In other words, in this networking scenario, if we restrict the liveness of the IP update hop by five packets or less, then the information becomes outdated by the time the attacker achieves reconnaissance (and begins exploiting the reconnaissance for further attacks). However, unlike our IP-hopping scheme, the controller-driven IP randomization (the baseline scheme in Section VIII-B) is not fast enough to defend against the implemented reconnaissance attack even if the IP address is changed for every packet, leaving about 1.9 milliseconds for the attacker to use and exploit the vulnerability from reconnaissance.

D. Network Topology Dependence

While Sections VIII-B and VIII-C demonstrated our scheme's superior efficiency over the controller-relying state-of-the-art baseline scheme and the effectiveness against attacker reconnaissance, respectively, we analyze our scheme's dependence on the network topology (and more specifically the path length between the source and the destination nodes) in this section. To support greater level of control in network topology, we implement and experiment in CloudLab [2], comprised of Intel Xeon E5530 processors at 2.4GHz; our implementations in CloudLab corroborate with our findings in Sections VIII-B and VIII-C and thus we focus on the network topology dependence in this section.

Figure 7 shows the measurement results for the processing delay of our randomization scheme while varying the path length in hops, excluding the propagation delay (which we display separately in Figure 8). We measure the processing/CPU latencies for the following tasks: "Randomization update" (IP address update based on LFSR PRG), "Delivery w/o randomization" (processing the networking without IP-address-based access randomization), and "Delivery w/ randomization" (processing the networking with the IP-address-based access randomization). "Delivery w/ randomization" measurement already includes the task of "Randomization update".

Randomization update (which includes running the LFSR PRG and the IP address update) takes 35.36μs and does not vary with the path length, as all nodes are computing them separately and independently to each other. Within the randomization update (comprised of running LFSR and writing/updating the IP address), the time to update the IP address took the most of the time, as the time to run the LFSR PRG takes merely 1.353μs, which accounts for less than 4% of the randomization-update process.

The processing for the networking delivery varies with the number of switches in between and is larger than the randomization-update overhead. When the hop length is one and the source and the destination are directly connected with no switch in between, the networking processing time without randomization scheme ("Delivery w/o randomization") is 4x greater than the randomization update; such difference in the processing overhead between that for networking and that for randomization update continues to grow as there are more switches in between, e.g., when 8 hops between the source and the destination, the difference grows by 6.41x. Therefore, the CPU/processing overhead of adding our IP randomization

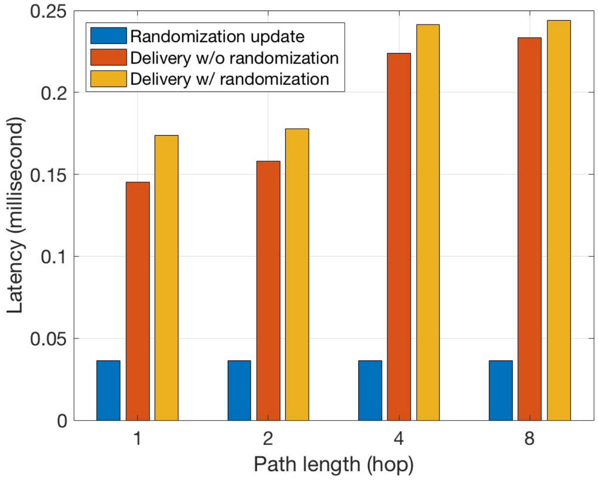

Fig. 7. The processing latency measurement with respect to the path length. This measures the average CPU time/latency over 10,000 samples without the packet propagation time, and the 95% confidence intervals are sub-microseconds, i.e., within 0.001 milliseconds (the propagation variance is greater as we will observe when discussing Figure 8). The horizontal axis shows the number of hops or the number of switches involved, e.g., 1 hop (on the far left) corresponds to direct link between the source and the destination involving 0 switches.

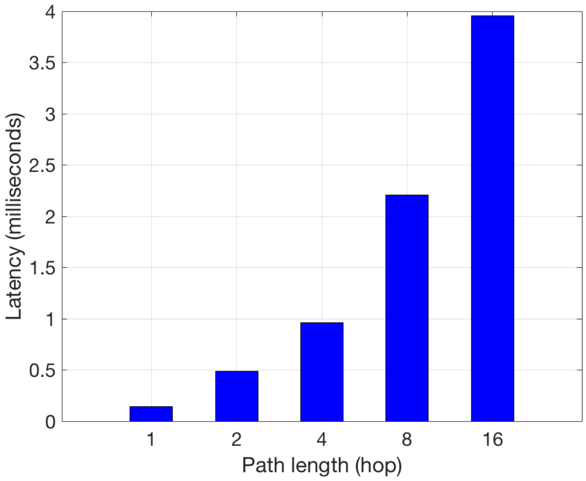

Fig. 8. The packet/path propagation time measurement with respect to the path length. This figure shows the average propagation delay over 10,000 samples.

scheme (which increment corresponds to the randomization-update cost) is small compared to the normal networking cost (without implementing our scheme), and the overhead becomes even relatively smaller as there are more nodes/hops in the path.

Propagation overhead, compared to the processing overhead, has a more of a dramatic increase as the path length increases and more nodes get involved in the packet delivery. Figure 8 studies the propagation delay, which is the time it takes for a packet to travel from the source to the destination. While the latency overhead with randomization grows by 40% for processing from 1 hop to 8 hops in Figure 7, the propagation latency grows by 1522% from 1 hop to 8 hops and by additional 1205% from 8 hops to 16 hops. Also, the variance/randomness of the latency measurements increase as there are more switches involved in the path. From 1 hop to 16 hops, the 95% confidence interval bound (assuming symmetric distribution) monotonically grows from 0.908μs to 3.45μs with 10,000 samples and from 5.9μs to 30.4μs with 100 samples. With 10,000 samples, the confidence intervals are 0.908, 0.945, 1.66, 2.59, and 3.45 (all in μs) with 1 hop, 2 hops, 4 hops, 8 hops, and 16 hops, respectively. Even within our network scenario, which has a single data traffic flow, the randomness increases as more nodes are involved in the forwarding.

In addition to the network topology, the latency performance is highly dependent on the system implementation, such as the SDN implementation [49], [50], and the networking environment and state, e.g., the number of simultaneous traffic flow and the switch queue state. While we analyze the latency dependence of our scheme on the path length, our performance measurements are conservative and the actual overhead of implementing our IP-hopping scheme is even lower relative to the networking delivery costs because there is no other traffic in our setup. In contrast, real-world networking with multiple traffic flow, introducing queuing delays on the switches will increase not only the delivery/propagation latency magnitude itself but also the randomness in the delivery/propagation latencies across different packets within the same flow, while the additional overhead incurred by our scheme is independent to such factors.

E. Synchronization Overhead

Using machines based on Intel Xeon E5530 at 2.4 GHz in CloudLab, we study the overhead of our IP synchronization scheme. More specifically, we measure the computational overhead in latency for pre-computing m synchronization signatures for each node. Our implementation uses SHA-256 hash function because the maturity of its one-way property and the wide use for security applications; as explained in Section VI, our synchronization scheme relies on the hash function's one-way property to prevent the attacker from breaching the synchronization by knowing about the future synchronization signatures before their use. To provide a reference, SHA-256 is used for Bitcoin (cryptocurrency), which mining is based on finding the reverse of a hash function and the miners has a global-scale computing resource (with the crowd-sourced miners competing around the world); successful mining, accepting multiple solutions/collisions, happens once every ten minutes; having such computational resource (equivalent to pooling the global-scale miners) is very difficult and expensive but, even against a hypothetical attacker with such computational resource, breaking the hash chain designed for our synchronization is more difficult than Bitcoin mining because our synchronization chain does not allow collisions (whereas collisions are allowed in Bitcoin mining to control the difficulty of the mining process).

As is shown in Table I, each hash computation based on the SHA-256 function takes 2.499μs on average, which is smaller by two orders of magnitude than the networking delay in the small-size OVS-networking-prototype in Section VIII-B and smaller by three orders of magnitude than the propagation time of a reasonably sized network (with a hop length of greater

TABLE I
THE COST OVERHEAD (TIME) OF GENERATING SHA-256-BASED SYNCHRONIZATION SIGNATURE

	Mean (μs)	Standard deviation (μs)
Hash computation	2.499	1.889
Hash and IP update	3.309	2.539

than four) in CloudLab in Section VIII-D. The hash computation, along with the IP address field update, takes 3.309μs on average. When the synchronization signatures get exhausted (and a new batch of m synchronization signatures are needed) or there is an update in the path (e.g., by flow-rule or routing change), each node needs 2.499$m\mu$s to compute the m-long hash chain and generate m synchronization signatures. Such computation for synchronization signature generation can also be done off-line or pre-computed, as the remaining hash chain gets shorter and there is an impending need for a new chain.

IX. CONCLUSION

We propose IP-address-hopping-based MTD in SDN to secure the forwarding path access against unauthorized access and network reconnaissance. The defense is implemented at the data plane (making it lightweight in networking and latency overheads) and at the hop level (thus preventing the unauthorized access from the first switch it encounters) and therefore outperforms the prior state of the art (having the controller randomize the endnodes's addresses) in both the defense impact and the processing/networking efficiency. Because our defense involves not only the endnodes but also the intermediate switches, we also build a IP-address-based synchronization scheme that uses a one-way hash chain and piggybacks on the data communications without incurring additional networking overhead. Considering an advanced attacker monitoring and probing the network for reconnaissance, we observe that our scheme increases the attacker's requirement cost for timely reconnaissance by more than an order of magnitude in our controlled lab environment. In CloudLab, we also show that the overhead of our scheme becomes even smaller relative to the data networking cost as the network size and the path length increases.

ACKNOWLEDGMENT

The authors extend their previous work [1] by introducing the IP-based synchronization scheme and implementing a prototype in CloudLab [2] to evaluate the proposed schemes in various networking topologies. The authors would like to thank the anonymous reviewers and the Associate Editor, Mauro Conti for their feedback.

REFERENCES

[1] S. Y. Chang, Y. Park, and A. Muralidharan, "Fast address hopping at the switches: Securing access for packet forwarding in SDN," in *Proc. IEEE/IFIP Netw. Oper. Manag. Symp. NOMS*, Apr. 2016, pp. 454–460.

[2] R. Ricci and E. Eide, "Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications," *USENIX Login*, vol. 39, no. 6, pp. 36–38, Dec. 2014. [Online]. Available: https://www.usenix.org/publications/login/dec14/ricci

[3] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "OpenFlow random host mutation: Transparent moving target defense using software defined networking," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2012, pp. 127–132. [Online]. Available: http://doi.acm.org/10.1145/2342441.2342467

[4] P. Kampanakis, H. Perros, and T. Beyene, "SDN-based solutions for moving target defense network protection," in *Proc. IEEE 15th Int. Symp. World Wireless Mobile Multimedia Netw. (WoWMoM)*, Jun. 2014, pp. 1–6.

[5] B. Awerbuch, A. Richa, and C. Scheideler, "A jamming-resistant MAC protocol for single-hop wireless networks," in *Proc. PODC*, Aug. 2008, pp. 45–54.

[6] S.-Y. Chang, Y.-C. Hu, and N. Laurenti, "SimpleMAC: A jamming-resilient MAC-layer protocol for wireless channel coordination," in *Proc. 18th Annu. Int. Conf. Mobile Comput. Netw. (Mobicom)*, 2012, pp. 77–88. [Online]. Available: http://doi.acm.org/10.1145/2348543.2348556

[7] S. Y. Chang and Y. C. Hu, "SecureMAC: Securing wireless medium access control against insider denial-of-service attacks," *IEEE Trans. Mobile Comput.*, vol. 16, no. 12, pp. 3527–3540, Dec. 2017.

[8] S. Lakshminarayana *et al.*, "Signal jamming attacks against communication-based train control: Attack impact and countermeasure," in *Proc. 11th ACM Conf. Security Privacy Wireless Mobile Netw. (WiSec)*, 2018, pp. 160–171. [Online]. Available: http://doi.acm.org/10.1145/3212480.3212500

[9] B. Muntwyler, V. Lenders, F. Legendre, and B. Plattner, "Obfuscating IEEE 802.15.4 communication using secret spreading codes," in *Proc. 9th Annu. Conf. Wireless On-Demand Netw. Syst. Services (WONS)*, Jan. 2012, pp. 1–8.

[10] S.-Y. Chang, J. Lee, and Y.-C. Hu, "Noah: Keyed noise flooding for wireless confidentiality," in *Proc. 11th ACM Symp. QoS Security Wireless Mobile Netw. (Q2SWinet)*, 2015, pp. 141–148. [Online]. Available: http://doi.acm.org/10.1145/2815317.2815329

[11] R. Pickholtz, D. Schilling, and L. Milstein, "Theory of spread-spectrum communications—A tutorial," *IEEE Trans. Commun.*, vol. COM-30, no. 5, pp. 855–884, May 1982.

[12] M. K. Simon, J. Omura, R. Scholtz, and B. K. Levitt, *Spread Spectrum Communications Handbook*. New York, NY, USA: McGraw-Hill, Mar. 1994.

[13] M. Wilhelm, I. Martinovic, J. B. Schmitt, and V. Lenders, "Short paper: Reactive jamming in wireless networks: How realistic is the threat?" in *Proc. 4th ACM Conf. Wireless Netw. Security (WiSec)*, 2011, pp. 47–52. [Online]. Available: http://doi.acm.org/10.1145/1998412.1998422

[14] M. Atighetchi, P. Pal, F. Webber, and C. Jones, "Adaptive use of network-centric mechanisms in cyber-defense," in *Proc. 6th IEEE Int. Symp. Object Oriented Real Time Distrib. Comput.*, May 2003, pp. 183–192.

[15] E. Al-Shaer, Q. Duan, and J. H. Jafarian, "Random host mutation for moving target defense," in *Security and Privacy in Communication Networks—SecureComm* (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering), vol. 106. A. D. Keromytis and R. D. Pietro, Eds. Heidelberg, Germany: Springer, 2012, pp. 310–327.

[16] P. Mittal, D. Kim, Y.-C. Hu, and M. Caesar, "Mirage: Towards deployable DDoS defense for Web applications," *CoRR*, vol. abs/1110.1060, Oct. 2011. [Online]. Available: http://arxiv.org/abs/1110.1060

[17] S. Antonatos, P. Akritidis, E. P. Markatos, and K. G. Anagnostakis, "Defending against hitlist worms using network address space randomization," in *Proc. ACM Workshop Rapid Malcode (WORM)*, 2005, pp. 30–40. [Online]. Available: http://doi.acm.org/10.1145/1103626.1103633

[18] M. Dunlop, S. Groat, W. Urbanski, R. Marchany, and J. Tront, "MT6D: A moving target IPv6 defense," in *Proc. Mil. Commun. Conf. (MILCOM)*, Nov. 2011, pp. 1321–1326.

[19] M. Albanese, A. De Benedictis, S. Jajodia, and K. Sun, "A moving target defense mechanism for MANETs based on identity virtualization," in *Proc. IEEE Conf. Commun. Netw. Security (CNS)*, Oct. 2013, pp. 278–286.

[20] D. Kewley, R. Fink, J. Lowry, and M. Dean, "Dynamic approaches to thwart adversary intelligence gathering," in *Proc. DARPA Inf. Survivability Conf. Exposit. II (DISCEX)*, vol. 1, 2001, pp. 176–185.

[21] J. Xu, Z. Kalbarczyk, and R. Iyer, "Transparent runtime randomization for security," in *Proc. 22nd Int. Symp. Rel. Distrib. Syst.*, Oct. 2003, pp. 260–269.

[22] R. Wartell, V. Mohan, K. W. Hamlen, and Z. Lin, "Binary stirring: Self-randomizing instruction addresses of legacy x86 binary code," in *Proc. ACM Conf. Comput. Commun. Security (CCS)*, 2012, pp. 157–168. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382216

[23] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, "FLOWGUARD: Building robust firewalls for software-defined networks," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2014, pp. 97–102. [Online]. Available: http://doi.acm.org/10.1145/2620728.2620749

[24] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments," *Comput. Netw.*, vol. 62, pp. 122–136, Apr. 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128613004003

[25] Y. Park, S.-Y. Chang, and L. M. Krishnamurthy, "Watermarking for detecting freeloader misbehavior in software-defined networks," in *Proc. Int. Conf. Comput. Netw. Commun. (ICNC)*, 2016, pp. 1–6.

[26] R. Jin and B. Wang, "Malware detection for mobile devices using software-defined networking," in *Proc. 2nd GENI Res. Edu. Exp. Workshop (GREE)*, 2013, pp. 81–88, doi: 10.1109/GREE.2013.24.

[27] J. Li, S. Berg, M. Zhang, P. Reiher, and T. Wei, "Drawbridge: Software-defined DDoS-resistant traffic engineering," in *Proc. ACM Conf. SIGCOMM (SIGCOMM)*, 2014, pp. 591–592. [Online]. Available: http://doi.acm.org/10.1145/2619239.2631469

[28] M. Shtern, R. Sandel, M. Litoiu, C. Bachalo, and V. Theodorou, "Towards mitigation of low and slow application DDoS attacks," in *Proc. Cloud IEEE Int. Conf. Eng. (IC2E)*, Boston, MA, USA, Mar. 2014, pp. 604–609.

[29] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch," in *Proc. 10th USENIX Conf. Netw. Syst. Design Implement. (NSDI)*, 2013, pp. 29–42. [Online]. Available: http://dl.acm.org/citation.cfm?id=2482626.2482631

[30] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS)*, 2013, pp. 413–424. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516684

[31] R. Braga, E. de Souza Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in *Proc. IEEE 35th Conf. Local Comput. Netw. (LCN)*, Oct. 2010, pp. 408–415.

[32] S. Matsumoto, S. Hitz, and A. Perrig, "Fleet: Defending SDNs from malicious administrators," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2014, pp. 103–108. [Online]. Available: http://doi.acm.org/10.1145/2620728.2620750

[33] X. Liu, X. Yang, and Y. Lu, "To filter or to authorize: Network-layer DoS defense against multimillion-node Botnets," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 195–206, Aug. 2008. [Online]. Available: http://doi.acm.org/10.1145/1402946.1402981

[34] K. Argyraki and D. R. Cheriton, "Active Internet traffic filtering: Real-time response to denial-of-service attacks," in *Proc. Annu. Conf. USENIX Annu. Tech. Conf. (ATEC)*, 2005, p. 10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1247360.1247370

[35] T. Anderson, T. Roscoe, and D. Wetherall, "Preventing Internet denial-of-service with capabilities," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, pp. 39–44, Jan. 2004. [Online]. Available: http://doi.acm.org/10.1145/972374.972382

[36] A. Yaar, A. Perrig, and D. Song, "SIFF: A stateless Internet flow filter to mitigate DDoS flooding attacks," in *Proc. IEEE Symp. Security Privacy*, Berkeley, CA, USA, May 2004, pp. 130–143.

[37] X. Yang, D. Wetherall, and T. Anderson, "A DoS-limiting network architecture," in *Proc. Conf. Appl. Technol. Archit. Protocols Comput. Commun. (SIGCOMM)*, 2005, pp. 241–252. [Online]. Available: http://doi.acm.org/10.1145/1080091.1080120

[38] B. Parno *et al.*, "Portcullis: Protecting connection setup from denial-of-capability attacks," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 289–300, Aug. 2007. [Online]. Available: http://doi.acm.org/10.1145/1282427.1282413

[39] D. Kim, J. T. Chiang, Y.-C. Hu, A. Perrig, and P. R. Kumar, "CRAFT: A new secure congestion control architecture," in *Proc. 17th ACM Conf. Comput. Commun. Security (CCS)*, 2010, pp. 705–707. [Online]. Available: http://doi.acm.org/10.1145/1866307.1866404

[40] M. S. Kang and V. D. Gligor, "Routing bottlenecks in the Internet: Causes, exploits, and countermeasures," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS)*, 2014, pp. 321–333. [Online]. Available: http://doi.acm.org/10.1145/2660267.2660299

[41] M. S. Kang, S. B. Lee, and V. D. Gligor, "The crossfire attack," in *Proc. IEEE Symp. Security Privacy (SP)*, Berkeley, CA, USA, 2013, pp. 127–141, doi: 10.1109/SP.2013.19.

[42] L. Lamport, "Password authentication with insecure communication," *Commun. ACM*, vol. 24, no. 11, pp. 770–772, Nov. 1981. [Online]. Available: http://doi.acm.org/10.1145/358790.358797

[43] N. Haller, "The S/KEY one-time password system," in *Proc. Internet Soc. Symp. Netw. Distrib. Syst.*, 1994, pp. 151–157.

[44] H. Padekar, Y. Park, H. Hu, and S.-Y. Chang, "Enabling dynamic access control for controller applications in software-defined networks," in *Proc. 21st ACM Symp. Access Control Mod. Technol. (SACMAT)*, 2016, pp. 51–61. [Online]. Available: http://doi.acm.org/10.1145/2914642.2914647

[45] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 147–163, Dec. 2002. [Online]. Available: http://doi.acm.org/10.1145/844128.844143

[46] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "Efficient authentication and signing of multicast streams over lossy channels," in *Proc. IEEE Symp. Security Privacy (S&P)*, Berkeley, CA, USA, 2000, pp. 56–73.

[47] D. Liu and P. Ning, "Multilevel μTESLA: Broadcast authentication for distributed sensor networks," *ACM Trans. Embedded Comput.*, vol. 3, pp. 800–836, Jan. 2003.

[48] B. Pfaff *et al.*, "The design and implementation of Open vSwitch," in *Proc. 12th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2015, pp. 117–130. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff

[49] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc. 2nd USENIX Conf. Hot Topics Manag. Internet Cloud Enterprise Netw. Services (Hot-ICE)*, 2012, p. 10. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228283.2228297

[50] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of SDN/OpenFlow controllers," in *Proc. 9th Central Eastern Eur. Softw. Eng. Conf. Russia (CEE-SECR)*, 2013, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/2556610.2556621

Sang-Yoon Chang received the B.S. and Ph.D. degrees from the Department of Electrical and Computer Engineering, University of Illinois at Urbana–Champaign in 2007 and 2013, respectively. He was a Post-Doctoral Fellow with the Advanced Digital Sciences Center. He is an Assistant Professor with the Computer Science Department, University of Colorado Colorado Springs. His research is in networking, wireless/mobile, systems security, and applied cryptography with focuses on cyber-physical systems, software-defined networking, and distributed consensus protocols/blockchain.

Younghee Park received the Ph.D. degree in computer science from North Carolina State University in 2010. She is an Assistant Professor of computer engineering with San Jose State University. She has served as the Kordestani Endowed Chair with the College of Engineering with SJSU in 2016 and 2017 as a Distinguished Research Professor. She is a Coordinator for the Cybersecurity Certificates Program with the Computer Engineering Department, SJSU, supported by the National Information Assurance Education and Training Program. Since 2016, she has been serving as the Center Executive at the Center for Smart Technology, Computing, and Complex Systems, a multidisciplinary research group of faculty and researchers in the area of the Smart City. Her main research is network and system security with an emphasis on malware detection, insider attacks, botnets, and traceback to detect attacks. She was a recipient of the Best Paper Award at ACM SIGCSE 2018, the Best Paper Award-Honorable Mention Award at the 21st ACM SACMAT in 2016, and the Faculty Excellence Award in Scholarship in College of Engineering at SJSU in 2018.

Bhavana Babu Ashok Babu received the bachelor's degree from Visveswaraiah Technological University, India. She is currently pursuing the master's degree in computer engineering with San Jose State University, where she is a Research Assistant. Her research is based on security in software-defined networking. She has worked on projects, such as moving target defense using IP randomization and currently working on queue configuration and bandwidth control using Ryu controller. She was also with Societe Generale Global Solution Center, a software company in India.