Hindawi
Security and Communication Networks
Volume 2018, Article ID 2916356, 20 pages
https://doi.org/10.1155/2018/2916356

Research Article

A Semistructured Random Identifier Protocol for Anonymous Communication in SDN Network

Yulong Wang, Junjie Yi, Jun Guo, Yanbo Qiao, Mingyue Qi, and Qingyu Chen

State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China

Correspondence should be addressed to Yulong Wang; wyl@bupt.edu.cn

Received 8 December 2017; Accepted 29 March 2018; Published 23 May 2018

Academic Editor: Hamed Okhravi

Copyright © 2018 Yulong Wang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Traffic analysis is an effective mean for gathering intelligence from within a large enterprise's local network. Adversaries are able to monitor all traffic traversing a switch by exploiting just one vulnerability in it and obtain valuable information (e.g., online hosts and ongoing sessions) for further attacking, while administrators have to patch all switches as soon as possible in hope of eliminating the vulnerability in time. Moving Target Defense (MTD) is a new paradigm for reobtaining the upper hand in network defense by dynamically changing attack surfaces of the network. In this paper, we propose U-TRI (unlinkability through random identifier) as a moving target technique for changing the information-leaking identifiers within PDUs for SDN network. U-TRI is based on VIRO protocol and implemented with the help of OpenFlow protocol. U-TRI employs an independent, binary tree-structured, periodically and randomly updating identifier to replace the first part of the static MAC address in PDU, and assigns unstructured random values to the remaining part of the MAC address. U-TRI also obfuscates identifiers in the network layer and transport layer in an unstructured manner. Such a semistructured random identifier enables U-TRI to significantly weaken the linkage between identifiers and end-hosts as well as communication sessions, thus providing anonymous communication in SDN network. The result of analysis and experiments indicates that U-TRI dramatically increases the difficulty of traffic analysis with acceptable burdens on network performance.

1. Introduction

Traffic analysis is a major threat faced by enterprises with large local networks. Through compromised switches [1, 2], adversaries are able to sniff out large amounts of traffic traversing in the enterprise networks. Packet headers of network protocols are originally designed for the benefits of service locating and packets delivery. However, they end up as an important source of information on traffic patterns for attackers, since implicit and explicit identifiers can be found in multiple layers of the network protocol stack. With these identifiers, attackers can gather intelligence, for example, which hosts are online, the vendor of the network adapter used by the host, who is talking to whom, what kind of tasks is probably being carried out, and which are the key hosts [3]. Since packet sniffing is a kind of passive attack, it is very hard to detect. What is worse, when attackers penetrate into one or a few switches, they would be able to perform a large scale reconnaissance in stealth. Therefore, it is critical for the defender to break the linkage between these identifiers and the hosts while maintaining the communication service. In other words, an unlinkability [4] communication protocol is needed.

Existing anonymity networks have already been focusing on the removal or obfuscation of implicit/explicit identifiers. Tor [5] utilizes the onion routing protocol to hide identifiers in the network layer and above, but it aims to solve the unlinkability problem on the Internet instead of in local networks of enterprises. The onion routing protocol cannot be adapted easily to enterprise networks since it routes among hosts and requires multiple encryption/decryption operations. Thus, it has relatively high latency and low robustness, which is acceptable for the Internet but apparently unsuitable for a local network. PHEAR [6] is a low-latency anonymous network solution for enterprise networks. PHEAR creates a randomized packet header called *nonce* for every packet in a network and provides unlinkability by encrypting all data beyond the network layer and exposing only the *nonce* for

routing. However, the nonce represents a pair of source and destination, which results in great exhaustion of the resources of routing devices since they have to store up to n^2 routing entries for a network with n end-hosts in each switch and need more CPU cycles for searching route entries and more control messages for updating route entries. The root cause of this disadvantage is the lack of structure in *nonce*. Besides, encrypting all data beyond the network layer would disturb operations of other SDN applications.

Aimed at narrowing the gap between unlinkability in local networks and existing methods, U-TRI is designed by leveraging a novel virtual id (denoted as *vid*) protocol called VIRO [7], which constructs a tree-like identifier space (called *vid* tree) for packets to be routed effectively in the network. VIRO is a static protocol that does not consider MTD in its design. U-TRI adopts VIRO in such a way that the identifier can be changed unpredictably while the advantages of VIRO are still reserved. This paper builds on our previous work published as a conference paper [8]. In the current version, we extend the basic idea to include a new MTD strategy. We also formally defined the unlinkability problem that U-TRI aims to solve and present mathematical analysis and experimental results on the unlinkability service that U-TRI provides. We proposed three MTD strategies called *tree shifting*, *subtree spinning*, and *hid randomization* to provide unlinkability on the data link layer of the network. By reconstructing the *vid* tree, spinning its subtrees or changing hosts' identifiers randomly, and updating routing tables in switches accordingly, U-TRI manages to put *vid*s in a consistent changing procedure while still supporting effective routing. Meanwhile, U-TRI obfuscates all identifiers in the network and transport layers in order to prevent identifier leakages. However, MAC address obfuscation is the main contribution and focus of this work; thus the obfuscation of other identifiers will only be briefly discussed.

The recently blooming SDN technology provides us with an effective way of implementing U-TRI, which consists of two functional entities: U-TRI server and U-TRI local proxy. The U-TRI server is responsible for *vid* changing as well as updating corresponding routing tables in OpenFlow switches through an SDN controller. The U-TRI local proxy is software running on end-hosts accomplishing the mapping between original packets and changed packets. This way, U-TRI provides an effective unlinkable communication that is transparent to the applications in end-hosts. We evaluate performance and security of a U-TRI network under different configurations using a realistic network with physical switches and end-hosts. The result indicates that U-TRI is a low-latency network system whose performance loss is acceptable in production. We also analyze the security level of a U-TRI network against several well-known attacks. The key contributions of this paper are as follows: (1) it is the first work that provides unlinkability to enterprise networks by using semistructured random virtual identifier; (2) the proposed scheme combines the advantages of VIRO and SDN to support a low-latency, routing resource saving MTD service; (3) the proposed scheme does not rely on payload encryption for identifier hiding so as to support coexistence with other SDN applications; (4) we conduct experimental evaluation of the proposed scheme in a realistic physical network with background traffic.

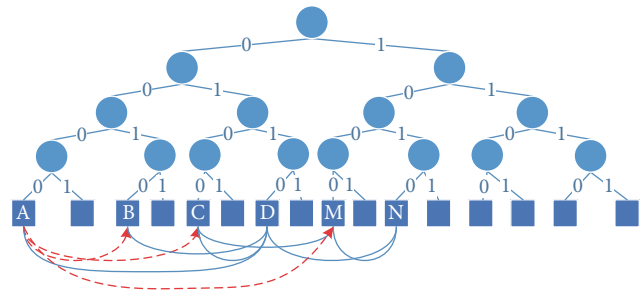


Figure 1: The vid tree in VIRO [8].

The rest of the paper is organized as follows. Section 2 overviews the existing work on which our work is based as well as research works similar to U-TRI. Section 3 presents the threat model and problem definition of our work. Section 4 describes the design details of U-TRI. Then we provide the experimental evaluation of U-TRI in Section 5. In Section 6 we analyzed the security of U-TRI and its effects on VIRO's key properties. Finally, Section 7 concludes the whole paper and describes the next step work.

2. Related Works

Our work's main interest is the design and implementation of a secure and efficient scheme for anonymous communication in local networks. Related research includes the works of VIRO and SDN which are the basis of our work and works that aim to solve similar problems.

2.1. Supporting Works

2.1.1. VIRO. VIRO [7] introduces a virtual identifier space onto which switches' physical identifiers are mapped, as shown in Figure 1. VIRO utilizes a *vid* tree to organize its identifier space, where each VIRO switch, which is represented by a leaf node of the tree, is assigned an L-bit string *vid* corresponding to the bits along the path from the root to that leaf node. When an end-host is initially attached to a VIRO switch, it is assigned an extended *vid* comprised of the L-bit *vid* of the switch plus a switch-chosen l-bit host identifier. Each node in the *vid* tree represents a subnetwork on a different level except for the root node of the *vid* tree, which represents the whole network. The multilevel structure of *vid*s is very helpful in host identifier aggregation and broadcast reduction.

The *vid space* structure of VIRO is critically built under the restrain of two invariant properties: *connectivity* and *closeness* [7]. The *connectivity* property ensures host reachability using *vid*-based routing. And the *closeness* property ensures that logical distance can reflect physical distance (measured in hops). (The *logical distance* δ of two *vid*s in an L-bit *vid space* is defined as $\delta(x, y) = L - lcp(vid(x), vid(y))$, where $lcp(vid(x), vid(y))$ is the length of the longest common prefix for binary strings $vid(x)$ and $vid(y)$.) These two properties

Table 1: An example of a VIRO routing table [8].

Level	Prefix	Nexthop	Gateway
1	00001	-	-
2	0001*	00010 (B)	00000 (A)
3	001**	00100 (C)/00100 (D)	00000 (A)
4	01***	00100 (C)	00100 (C)
5	1****	00010 (B)	00010 (B)

together make sure the VIRO network is an effective switching network. Note that any modification to VIRO should not violate these two properties in order to maintain the advantages of VIRO. In Section 6.2 we will prove that our adoption of VIRO still reserves these properties.

VIRO switches use VIRO routing tables to route packets. A logical instance of a VIRO routing table consists of four columns: level, prefix, nexthop, (nexthop: the neighbor to reach a level-k gateway) and gateway (gateway: a level-k gateway for a VIRO node x is a node y which is less than k logical distance away from x and is connected physically to a node which is at a logical distance of k to x). Table 1 is an example of the constructed routing table of node A in Figure 1. A group of switches that are at level-k of A is those switches that are at a logical distance of k to A. Also, they share the same (longest common) prefix with A. Therefore, prefixes can be seen as a representation of a level. Given the node's *vid*, the prefix of each level can be calculated directly. When routing packets, only the prefix and nexthop columns are actually used. Packets are matched on destination address using the prefix and outputted to the corresponding port of the nexthop. The gateway and level columns are only used when building the routing table.

2.1.2. Software-Defined Networking. Software-Defined Networking (SDN) [10–12] decouples the control plane which manages where traffic is sent to and the data plane which performs the actual packet forwarding, allowing networks to be dynamically initialized, controlled, changed, and managed through programmatic methods from a single point named SDN controller. The SDN controller is capable of gathering and storing all information about the network by controlling SDN switches through the OpenFlow protocol. Meanwhile, applications can access the SDN controller through its northbound API in order to add, modify, or remove flow rules to manipulate the network's behavior. U-TRI is designed to be such an application that can utilize the capabilities of the SDN technology.

2.2. Similar Works. The most similar research work to U-TRI is PHEAR [6], which is a system built on existing SDN protocols and standards. PHEAR is aimed at protecting traffic in enterprise/campus network through removing implicit and explicit identifiers from network traffic. It presented a great idea in providing unlinkability to a network by introducing a new, routable, randomly generated, and periodically changing identifier named *nonce* to the packet header, normalizing other identifiers at network layer two and layer three and encrypting payload in layer three using existing security protocols. PHEAR is a novel solution implemented in an SDN environment. However, *nonce*, which represents a pair of packet source and destination, does not have a hierarchically routing ability and can cause great waste to precious flow table resources on network devices. Since one *nonce* can only represent one pair of end-hosts, central network devices may have to store as many as n^2 *nonce*s in a network with n end-hosts. Meanwhile, the PHEAR server provides no central control on *nonce* refreshing, making it impossible for network administrators to globally and gracefully downgrade security levels when performance is more important than security. The encryption scheme of PHEAR which encrypts all upper layer payload would make other SDN applications running in the same network ineffective since some flow rule matching cannot work on encrypted packets.

Tor [5] is the most well-known anonymizing network on the Internet. Liu et al. [13] take a further step to enhance censorship resistance of Internet by replacing IP with Tor. The basic idea of the proposed scheme for providing unlinkability (or censorship resistance in the terms of the paper) is implementing Tor protocol in the so-called onion routers, through which all traffic is required to travel. A similar system can be implemented in a local network but has two major shortcomings. Firstly, the Tor protocol is required to adapt to the data link layer since most of the intermediate nodes in a local network are switches instead of routers. Secondly, SDN has already provided the infrastructure for such innovative network services to be implemented on the same switches. There is no need to add a new type of single function network device which would add more burden to network administration. And encryption-based solutions such as Tor protocol are apparently hard to cooperate with other network services if not sharing secrets, but sharing secrets with other systems is clearly not allowed by Tor.

RHM [14] is an address mutation approach for disrupting reconnaissance attacks. It replaces the real IP address of packets with a routable short-lived ephemeral IP address when the packets traverse the protected network and restores the addresses before the packets arrive at their destination hosts. RHM depends on a large unused routable IP address space to provide the ephemeral IP addresses, which is not always easily available. In contrast, U-TRI does not have such dependency on existing, sometimes precious, namespace resources. On the other hand, RHM uses edge devices instead of local proxies on end-hosts to conduct the mapping between real addresses and ephemeral addresses. Though such a mechanism can avoid the management overhead on end-hosts, we insist on placing the address mapping function on end-hosts, because it can actually prevent the raw packets containing the destination addresses from exposure to the edge devices, thus providing stronger unlinkability.

Duan et al. [15] proposed a random route mutation (RRM) technique to defend against eavesdropping. Changing the traversing paths of packets indeed disrupts the linkability analysis of packets conducted by adversaries on routers. However, its effectiveness depends on the number of different routes between end-hosts. Wide area networks may be able to provide such a sufficiently large number of different routes,

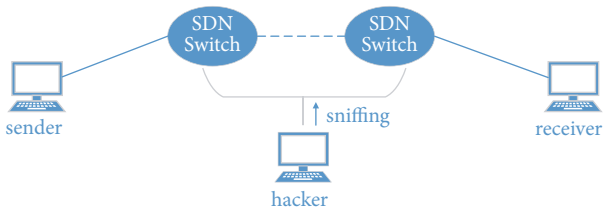


Figure 2: The threat model of U-TRI [8].

but this is clearly not the case in a local network, which usually manages to prevent loops in topology and has much less redundant traversing paths.

Venkatesan et al. [16] proposed an MTD approach utilizing proxy. In their approach, the proxy is not software on end-hosts, but a standalone device in the network acting as a traffic filter for mitigating DDoS attacks. The moving aspect of the approach is mainly implemented on its lookup server, which is responsible for informing authenticated end-hosts of which proxies are available for the moment. The software on the end-hosts that fulfills the authentication and lookup requesting tasks is the counterpart of the U-TRI local proxy. To be effective, an MTD solution usually requires a means to distinguish normal users from adversaries. And local proxy is just one of such means.

3. Problem Description

3.1. Threat Model. U-TRI's threat model is similar to that of PHEAR [6]. In an enterprise network, the adversaries may monitor packets in flight on their compromised section of the network. Also, flow tables on these compromised infrastructures are observable to them. Since SDN controller itself is usually trustworthy, an SDN controller is considered trusted in U-TRI while SDN switches are not.

To the best of our investigation, it seems that no *unlinkability* solution exists that can prevent compromised end-hosts from finding out to whom they are communicating. We believe the reason is that providing *unlinkability* from an end-host violates the basic requirement of networking service provision in that the services cannot be found without an identifier (e.g., network address). As shown in Figure 2, the *sender* inevitably needs the address of the *receiver* to communicate with it. If the *sender* is compromised, the adversary can naturally obtain the address of the *receiver*. Therefore, the compromised end-host is not part of our threat model and there are other solutions for protecting end-hosts [17–19].

Putting all of the above situations into consideration, we assume that the authenticated end-hosts, the SDN controller, the U-TRI server, and the dedicated line between SDN controller and U-TRI server are in the trusted domain, while the switches and end-hosts are all in the untrusted domain.

However, the adversaries sometimes may obtain a copy of the identifier assignment result which exposes the linkage between identifiers and hosts at that time by taking advantage of an unintentional information leak from the U-TRI server or some other unknown sources. We also consider bringing this into our threat model so as to provide a more secure solution.

3.2. Problem Definition. In this section, we define the problem according to the threat model. The problem that U-TRI aims to solve is to increase the difficulty for an adversary who can monitor packets from switches to figure out who is talking to whom. We assume that an adversary does not know the number of end-hosts connected to the switch ports he is monitoring. The rationale for this assumption is due to the fact that adversaries usually do not know the topology of a network. Thus, even if an adversary can penetrate into a core switch, the number of hosts connected to the port is still hard to decide without knowledge of the topology. If the adversary penetrates into an edge switch, he would be able to know that some of the ports are connected to only one host. However, this is usually the knowledge that he already possesses before the penetration, so no new linkage information is obtained by the adversary in this situation. We also assume that if an identifier is unchanged for a period of time t, then the probability for an adversary to determine the linkage between this identifier and some host is increased in proportion to t. This assumption is natural and intuitive because that is where MTD comes in.

Let $\Pr(H(I))$ denote the probability of mapping an observed identifier to an interested host H. A higher $\Pr(H(I))$ means a less workload for deciding the linkage between the identifier I and the host H, thus representing a lower difficulty for linkage recovery. Therefore, the problem is to minimize $\Pr(H(I))$. It is apparent that $\Pr(H(I)) = 1$ when the identifier is in plaintext, since the identifier shows the linkage information directly in this case. The interesting problem is to determine $\Pr(H(I))$ when the identifier is obfuscated and the adversary knows nothing about the mapping between the original identifier and the obfuscated identifier. Similar to cryptanalysis, we also consider two typical cases: ciphertext-only attack (COA) and known-plaintext attack (KPA). In the case of COA, what the adversary can have is the set of obfuscated identifiers appearing in the packets transferring through the switch port he/she is monitoring. What the adversary knows is that one of these identifiers belongs to the host H. Thus we have

$$\Pr\left(H\left(I\right)\right) = \frac{1}{\left|P_m\right|} \tag{1}$$

in which P_m is the anonymous set of the switch port the adversary is monitoring on. The value of P_m is determined by the identifiers that can appear on the port. In the case of KPA, besides the above knowledge, the adversary also knows the mapping between I and H at time $t-1$. We want to know the possibility of he/she finding the correct I that belongs to H at time t. Formally, we have

$$\begin{aligned}&\Pr\left(H\left(I_t\right) \mid H\left(I_{t-1}\right)\right)\\ &\quad = 1 - \Pr\left(I_t \neq I_{t-1}\right)\\ &\qquad + \Pr\left(I_t \neq I_{t-1}\right)\Pr\left(H\left(I_t\right) \mid H\left(I_{t-1}\right), I_t \neq I_{t-1}\right).\end{aligned} \tag{2}$$

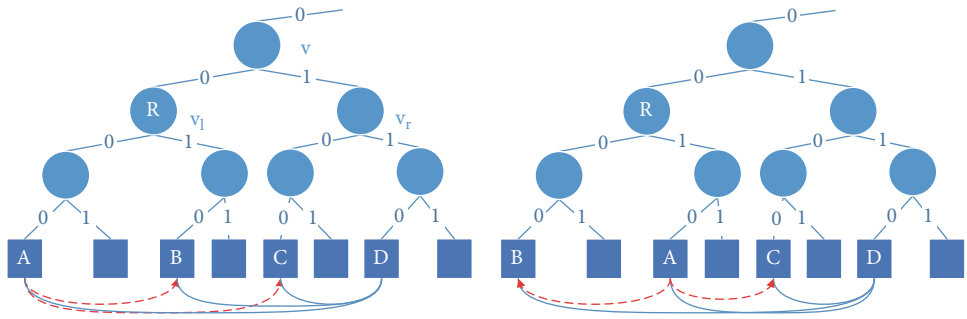


FIGURE 3: Example of subtree swapping of node R (part of the vid tree in Figure 1) [8].

From (2), we can see that $\Pr(H(I_t) \mid H(I_{t-1}))$ is determined directly by the identifier changing rate (reflected by $\Pr(I_t \neq I_{t-1})$). We will prove that it is also determined indirectly by the port anonymous set in Section 6.1.1. Therefore, the problem is to design an MTD protocol so as to minimize both $\Pr(H(I))$ and $\Pr(H(I_t) \mid H(I_{t-1}))$.

4. U-TRI

In this section we present a detailed description of U-TRI.

4.1. Overall Design. The major purpose of U-TRI is to protect the network traffic from linkability attacks even if some switches in the network are compromised. To achieve this goal, we need to provide *unlinkability* [4] of communications from the view of network switches. The degree of *unlinkability* is actually dependent on the size of the anonymity set of each port of a switch. The anonymity set of a switch port is the set of all destination addresses that one packet can possibly reach through this port. Conventionally, packets' source and destination addresses seldom change during communication between two hosts, which means the size of the anonymous set of the port is fixed and small; thus traffic between hosts is easily linked when in-between switches are compromised. To ensure communicating hosts are unlinkable, we need to make linkable identifiers, especially source and destination MAC addresses, change unpredictably. Thus, we need a new type of host identifier possessing the following attributes: (1) *changeable*, the identifier can be changed randomly within a space large enough while preserving global uniqueness; (2) *routable*, the identifier can be used in effective routing; (3) *single-purposed*, the identifier should not have extra semantics except for routing. Fortunately, VIRO provides a good foundation possessing the second and third attributes.

The *vid* of VIRO is a born hierarchical identifier that is purely designed for effective packet routing. It does not contain any information of end-hosts and can naturally aggregate nodes into subnetworks. Our main concern now becomes how to put *vid* into an unpredictable, periodically changing process. This can be done by completely reassigning different *vid*s to the entire network periodically or just updating some *vid*s strategically. Given the fact that VIRO's *vid* space of switches is constructed on a binary tree, we manage to develop two strategies called *tree shifting* and *subtree spinning* to handle switches' identifier moving. The former is to randomly reassign a different virtual identifier tree to the network. The latter basically means swapping the two subtrees of some randomly chosen nodes (excluding leaf nodes, which represent a switch each) in the binary tree in an unpredictable manner. For example, by spinning subtrees of node R in Figure 3, node A's *vid* becomes *0010*, and node B's *vid* becomes *0000*. If the identifier of a switch is changed randomly by either *tree shifting* or *subtree spinning*, every switch in the network is then not fixed to any identifier, and with host identifier (denoted as *hid*) randomization, we would be able to provide traffic source and destination *unlinkability* to the network.

Overall Architecture. The architecture of a U-TRI network consists of five components: U-TRI server, SDN controller, OpenFlow switch, end-host, and U-TRI local proxy. The U-TRI local proxy (ULP) is a lightweight program running on authorized end-hosts whose only function is to map between real and virtual identifiers. The mapping operation mainly involves hash table lookup whose time complexity is a constant value. The ULP communicates with the U-TRI server in a secure channel based on IPsec ESP [20]. The U-TRI server is the MTD center of U-TRI; it distributes *vid*s, answers ARP requests, and utilizes the northbound API of the SDN controller to manage flows in OpenFlow switches. The functions of these components will be explained in detail in Section 4.3.

Protection of Other Identifiers. Note that other identifiers above the data link layer such as IP addresses, TCP/UDP ports, and IP identification can also leak linkage information to the adversaries [21]. Therefore, besides MAC address, U-TRI also needs to obfuscate other identifiers in packets to prevent them from being exploited by linkability attacks.

U-TRI adopted IPsec ESP (transport mode) to encrypt the IP payload of its control traffic, i.e., the messages sent between the ULP and the U-TRI server. Also, marking that IPsec does not cover the IP address field, ULP will simply set the IP addresses of every packet to syntactically valid random values, since the role of IP addresses in packet routing and forwarding is now taken over by U-TRI's *vid*. ULP will

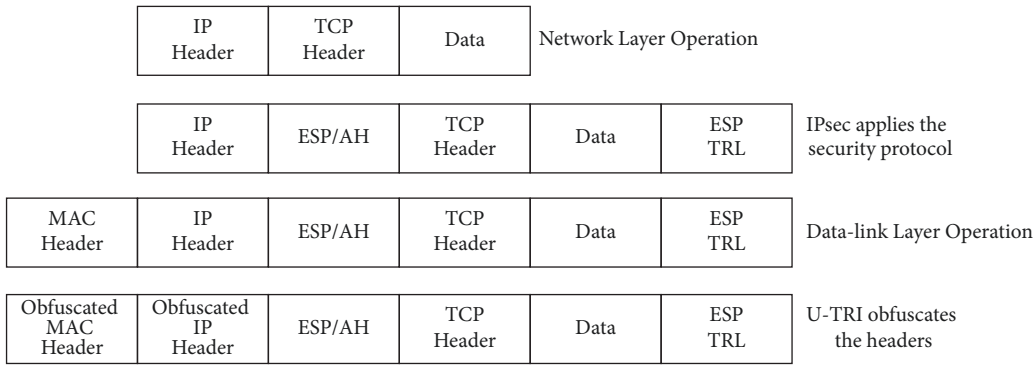


Figure 4: U-TRI's compatibility with IPsec.

restore the randomized IP addresses to their original values at the destination. The obfuscated IP addresses appear to be meaningful values but their values are only used to confuse the adversary. Upon being received by the destination ULP, its real value will be restored so as to provide a transparent implementation of defense with regard to applications running on the hosts. Also, the initial TTL value of a packet may expose the type of OS, so the ULP regulates it to a constant value that does not correspond to any specific OS.

As for user traffic, depending on whether IPsec ESP can be used between the hosts, there exist two strategies. If IPsec ESP can be used, then U-TRI just uses the same mechanism used for control traffic. However, IPsec encryption is not a must because other SDN applications such as deep packet inspection may rely on upper layer contents. In this case, U-TRI will activate a mapping mechanism to rewrite the changeable identifiers of packets (i.e., those not used by other SDN applications) to hide their real values when packets are traveling in the network. The mapping is stored in the U-TRI server and the ULP so that identifiers' real values can be properly restored. The identifier obfuscation involves the network layer (e.g., IP address and IP identification field) and the transport layer (e.g., TCP/UDP ports, TCP initial window size, and TCP timestamps), but not the application layer since it is time-consuming and will incur significant performance downgrade. This mapping mechanism is trivial, so we will not describe it further.

Compatibility with Other Security Protocols. The relation between U-TRI and IPsec is illustrated in Figure 4. U-TRI does not interfere with IPsec's operation. ULP only rewrites some fields after the IPsec's encryption process and restores them before the IPsec's decryption process. From the IPsec's perspective, ULP does not exist. U-TRI is also compatible with higher layer security protocols such as SSL, TLS, and SSH on transport layer and PGP, as well as S/MIME and HTTPS on the application layer, since U-TRI does not change the data field in Figure 4 whether it is encrypted or not.

Implementation Transparency. Transparency is a desired attribute for a practical MTD solution. U-TRI works from one network entity to another network entity, not from one application process to another application process. Hence, U-TRI can be adopted without requiring changes made to the OS or user applications. Specifically, the ULP does not modify any APIs or libraries of the operating system. It merely adds a kernel module to rewrite packets' headers before they are sent and restore them after they are received. Thus, the existence of the ULP is transparent to any applications running on the operating system. U-TRI is transparent to OpenFlow switches too since it is just yet another SDN application which updates flow tables through a standard SDN controller. U-TRI's transparency to other SDN applications is a complicated issue. U-TRI requires mandatory obfuscation on some fields such as MAC address and IP address and optional obfuscation on other fields such as TCP/UDP ports. For those SDN applications that depend on real MAC address or IP address, U-TRI must inform them with the current obfuscated value of these fields so that they can revise their flow entry accordingly to maintain functionality. However, for those depending on optional obfuscating fields, U-TRI would just keep those fields intact (configured by the administrator) and keep being transparent to them. In Section 5 we used JMeter [22] (acting as client) and Alfresco [23] (acting as server) to perform experiments. All of these applications worked well and did not sense the existence of U-TRI. However, if the updating rates of *vid* and *hid* are set too high, users may experience a noticeable network access delay.

4.2. Virtual Identifiers. In this section, we focus on the principles of the *vid* design of VIRO and how U-TRI is ameliorating it to fit our needs.

Sid. The leaf nodes shown in VIRO's *vid* binary tree only represent switches; thus in U-TRI, the name for the *vid* of switches is redefined as *sid*. The *sid* represents switches and is constructed according to the binary tree. The U-TRI server is responsible for the generation and assignment of the *sid*. In this paper, we use sid_x to represent the *sid* of a switch x or a switch that connects directly to a host x.

Hid. The U-TRI server assigns each end-host another unique identifier: the *hid*. The *hid* does not come in a hierarchical way, because it is not designed for the routing of a packet, but for a larger anonymity identifier set.

Vid. *vid* is the concatenation of *sid* and *hid*. U-TRI repurposed the MAC address field for *vid*. We divide the 48 bits

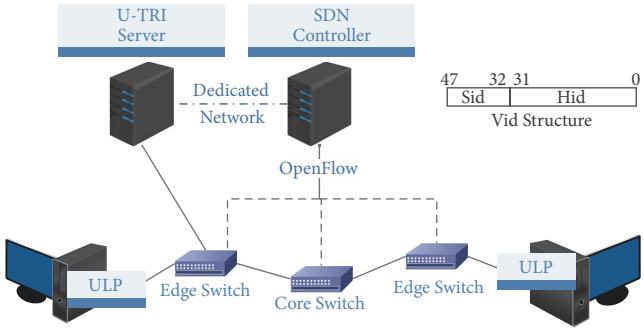


Figure 5: U-TRI architecture [8].

into two subfields: *sid* and *hid* (as shown in Figure 5). Given a host, the *hid* part of its *vid* is its current host identifier, while the *sid* part is the current switch identifier of its edge switch. Given a switch, the *hid* part of its *vid* is zero, while the *sid* part is its identifier. The *sid* subfield is 16 bits long while the *hid* subfield is 32 bits long (the length of both subfields can be adjusted according to network size as long as the sum of the length is still 48 bits).

4.3. Architecture. In this section, we describe functions of each U-TRI component in detail.

4.3.1. OpenFlow Switch and Controller. The SDN controller opens up APIs for the U-TRI server to control OpenFlow switches. It is connected to the U-TRI server through a dedicated line, since the SDN controller, as a network infrastructure, should not be accessible from average end-hosts. U-TRI does not customize the standard implementation of the SDN controller and the OpenFlow switches so that it can coexist with other SDN applications.

In order to get the network to operate, correct flow tables (flow table pipelining is supported since OpenFlow 1.3) must be installed at OpenFlow switches. There are at most four flow tables at each OpenFlow switch, as shown in Table 2.

Table 2(a) (The Entering Table). Every packet would pass through this table, whose main job is to classify packets so that they can be processed next by the corresponding table(s). Meanwhile, the Entering table in the edge switch would have one more rule, which is to match ARP request packets, change their destination MAC addresses to the *vid* of the U-TRI server, and then resubmit them to the routing table so they can be transferred to the U-TRI server.

Table 2(b) (The Host-Sending Table). This table deals with packets sent by local end-hosts; thus only edge switches have this table. Rules in this table match packets with all 0s as their destination *sid* (the destination *sid* of packets sent out by end-hosts is set to zero instead of the correct value to obtain independent updating for *sid* and *hid*), replace the 0s with the proper destination *sid* that is set previously by the U-TRI server through the SDN controller, and resubmit them to the routing table so that they can be transferred to the destination switch.

Table 2: Four types of flow tables in U-TRI (rows surrounded by dashes and dots are meant to be more than one in the table) [8].

(a) The entering table

Match	Action	Priority
ARP_REQUEST	mod_eth_dst($vid_{U\text{-}TRI\ server}$), goto Table (c)	HIGH
$sid == 0$	goto Table (b)	MID
*	goto Table (c)	LOW

(b) The host-sending table

Match	Action
hid_{dst}	mod_eth_dst($sid_{dst} + hid_{dst}$), goto Table (c)

(c) The Routing Table

Match	Action
prefix	output to port
self.*sid*	goto Table (d)

(d) The host-receiving table

Match	Action
local *hid*	output to port

Table 2(c) (The Routing Table). It is a reimplementation of the VIRO routing table (an example is shown in Table 1) with additional operations for U-TRI. OpenFlow switches use prefixes to match on packets' destination *sid*s and send them to the *nexthop* of the matched prefix. The last rule of this table matches on the *sid* of the switch itself, restores the destination *sid* of the matched packets to all 0s, and resubmits them to the host-receiving table so that they can be sent to the corresponding local end-hosts.

Table 2(d) (The Host-Receiving Table). This is also a table that only edge switches possess. Rules in this table match on destination *hid*s and send the matched packets to the *hid*'s corresponding egress port. These *hid*s comprise only the *hid*s of local end-hosts.

The existence of the host-sending table and the host-receiving table ensures that *sid* updating is transparent to end-hosts. End-hosts find their communication peers by the peers' unique *hid*s, while the destination *sid*s have nothing to do with them. The *sid* is only used when transferring packets and thus is not present in the packet before the packet enters the edge switch's host-sending table. The host-receiving table makes sure that the packet is sent to the destination that the *hid* indicates. The U-TRI server does not need to inform any end-hosts when updating *sid*s; it only has to update the host-sending tables as well as the routing tables.

4.3.2. U-TRI Server. The U-TRI server is the MTD center of U-TRI. It has two network interfaces so that it has access to both the enterprise networks we want to protect as well as the dedicated line used to communicate with the SDN controller. It has three functions: building routing tables, answering ULP, and executing *sid* updates.

```
Input: network topology topo
Output: Routing table rt for each switch sw in topo
/* building the routing table with directly connected
neighbors */
(1)  foreach switch sw in topo do
(2)    foreach neighbor nb in sw do
(3)      level ← calcVidDistance(sw, nb);
(4)      port ← sw.getConnectingPort(nb);
(5)      prefix ← getPrefixWithLevel(sw.vid, level);
(6)      sw.rt.addOutputToLevel(prefix, port);
(7)    end
(8)  end
/* find all gateways of each switch on each level */
(9)  foreach switch sw_a in topo do
(10)   foreach switch sw_b in topo and sw_a ≠ sw_b do
(11)     foreach level k in sw_b.rt.levels do
(12)       if sw_a.isGatewayOf(sw_b, k) then
(13)         sw_b.addGateway(k, sw_a);
(14)       end
(15)     end
(16)   end
(17) end
/* find a nexthop for each gateway by recursively
looking up the routing table */
(18)  foreach switch sw in topo do
(19)    foreach level k in sw.rt do
(20)      gw ← sw.rt.getGateway(k);
(21)      nh ← sw.rt.findNexthopRecursively(gw);
(22)      port ← sw.getConnectingPort(nh);
(23)      prefix ← getPrefixWithLevel(sw.vid, k);
(24)      sw.rt.addOutputToLevel(prefix, port);
(25)    end
(26) end
```

Algorithm 1: Building routing table.

Building Routing Tables. When the network initially bootstraps or a tree shift event occurs, the U-TRI server needs to install flow tables with proper flow rules for network switches. These flow rules mainly exist in the routing table, which is built in a centralized manner as opposed to the distributed manner of VIRO [7]. With the help of SDN and OpenFlow Discovery Protocol (OFDP), U-TRI server is able to know the network's full topology. And with the *sid*s of every switch, the U-TRI server can generate the routing table for every switch using Algorithm 1.

Answering ULP. There are three types of requests that a ULP would send to the U-TRI server.

The first one is *registration*. When a newly authorized end-host (denoted as X) with a proper ULP running on it is connecting to the network for the first time, its ULP will send out a *registration* message to the network's U-TRI server to get a new *hid*. On receiving the *registration* message, the U-TRI server extracts and stores X's information together with a newly generated unique hid_X. Before answering the *registration*, the U-TRI server will install a new flow rule into the host-receiving table of the edge switch of X (denoted as E_X) through the SDN controller, so that E_X can now recognize hid_X and send packets to X. Afterward, the U-TRI server answers the ULP of the new hid_X.

The second one is the *ARP request*. Since we repurposed the MAC address field to hold *vid*s, end-hosts' ARP requests are now transferred to the U-TRI server and answered there. Suppose X is requesting the MAC address of end-host Y, whose edge switch is denoted as E_Y. On receiving this ARP request, the U-TRI server will find hid_Y first, but will not answer the request immediately. Instead, the U-TRI server will add a new flow rule to the host-sending table of E_X, matching the packet's destination with hid_Y, and change the destination *sid* to sid_Y. Also, it stores the new communication pairs X and Y in a *communication-peer* table. After all of these are done, the U-TRI server answers the ARP request with hid_Y.

The third one is the *hid update request*. For every random period of time, the ULP would request a new randomly generated unique *hid* from the U-TRI server to increase *unlinkability*. Suppose X is requesting a new hid'_X. Besides answering the *hid update request* and updating the host-receiving table of E_X, the U-TRI server has to inform those end-hosts and edge switches which are currently communicating or used to

communicate with X, so that they can get new destination hid'_X as soon as possible, preventing the communication from being interrupted. The U-TRI server does not have to broadcast this information; instead, it only needs to look up its *communication-peer* table and inform the aforementioned end-hosts. The U-TRI server also needs to update flow rules containing the old hid_X in the host-sending tables of these end-hosts' edge switches.

Executing Sid Update. The execution of *sid* update is totally controlled by the U-TRI server. We will discuss the details in Section 4.4.2.

4.3.3. U-TRI Local Proxy. The ULP acts as the separator between trusted and untrusted domain defined in Section 3.1. Without the ULP, U-TRI would not be able to keep real identifiers unknown to not only core switches but also edge switches.

The ULP have two major functions. The first one is to register itself with its certificate (which is usually stored in a Cryptographic USB Token assigned by the enterprise's IT management department) to the U-TRI server so that the end-host it resides on becomes authenticated. The second one is to be the mapper between original and obfuscated identifiers. Taking MAC address as an example, the ULP rewrites the source MAC address with its end-host's *hid* on sending and restores the destination *hid* to its end-host's MAC address on receiving. Identifiers in the network and transport layers will be processed in a similar manner, except that they will not be used for routing.

Moreover, since the destination addresses stored in end-host's ARP table are actually *hid*s instead of *real* MAC addresses, the ULP can also help to hide communication peers' MAC addresses when the end-host is compromised.

4.4. Vid Update. In this section, we introduce how we update *vid*s in detail. To make things more complicated for the adversaries, we separate the updating of *hid* and *sid*. Their updates are asynchronous and independent.

4.4.1. Hid Update. *hid* update is started by ULP on individual end-hosts. To update its *hid*, the ULP simply sends a *hid update request* to the U-TRI server, and the U-TRI server will do the rest of the job, as we have already described in Section 4.3.2. A *hid* update can happen anytime on any end-host, making the change of *hid*s in the network totally unpredictable.

4.4.2. Sid Update. *sid* update is started by the U-TRI server. Each time when a new round of *sid* update is needed, what the U-TRI server has to do is as follows: (1) generating new *sid*s with *subtree spinning* or *tree shifting* and (2) modifying existing host-sending tables and routing tables with new *sid*s to ensure that communications can still proceed. (The ARP rule contains the *sid* of the U-TRI server; thus it also needs to be updated when the *sid* of the U-TRI server is changed.) Note that *no end-host needs to be informed* of a *sid* update, which provides more flexibility in the sense of MTD.

The generation of new *sid*s is controlled alternatively by two different strategies, which we will discuss in depth in Sections 4.4.3 and 4.4.4. Here, we focus on introducing the process of the modification of flow tables. Assume that each switch already has a new *sid*. In the case of the *tree shifting* strategy, all tables with old *sid*s need to be rebuilt. In the case of the *subtree spinning* strategy, modifying the host-sending table is trivial; simply updating old sid_{dst} to the new sid_{dst} would finish the job (Table 2(b)). For modifying the routing table, based on the reasoning in Section 6.2, the U-TRI server only needs to update the prefixes when updating the routing table. With the new *sid* generated, a new group of prefixes can be calculated and be used to replace the old ones by the U-TRI server. The output port in association with these prefixes should remain the same since *subtree spinning* does not change the *nexthop*s in a routing table, according to Theorem 5.

4.4.3. Subtree Spinning Strategy. When designing the *subtree spinning* strategy, a few principles should be met: (1) The execution of *subtree spinning* should be random in terms of spinning subtrees selection. (2) Physical network topology should be taken into consideration when subtrees are chosen; for an instance, any pair of subtrees whose leaf nodes are empty (no corresponding physical nodes) should never be swapped. (3) The strategy should cover every node in the network, making sure their *vid*s will not stay unchanged (for a relatively long time). In fact, the strategy is the answer to the question: *How to choose nodes in the virtual identifier tree periodically so that the spinning of the nodes' subtrees would bring out the best ability in MTD?* We note that *subtree spinning* consists of multiple *swaping* operations. That is, $swap(r)$ is the operation of exchanging the left and right subtrees of tree node r. Let T be the *vid* binary tree, P_r be the spinning possibility of node r, S be the *subtree spinning rate*, and T_r denote a subtree of T whose root node is r; we propose a heuristic *spinning strategy* as shown in Algorithm 2. This algorithm will be executed every S seconds to periodically change *sid*s in the network.

The decay factor ϵ is a small positive number used to slightly adjust the spinning rate of a node so that no nodes would remain unspun for a long time. The key of the algorithm is to decide the initial value of P_r. It is clear that a proper value should balance both security enhancement brought by U-TRI and traffic performance loss caused by U-TRI.

Firstly, we focus only on how to create a spinning strategy that brings out the best ability in MTD. What MTD essentially does is to help the defender get the upper hand by making the attacker operate in an uncertain and unpredictable environment [24, 25]. Thus, our goal is to provide the most unpredictable spinning subtrees. Note that in U-TRI, *vid*s of network nodes are the only variables that stay exposed. Thus, in the case of changing *vid*s in a completely random way, the unpredictability of identifier would be achieved. Furthermore, the uncertainty of a random variable can be measured by its entropy, which is maximized when the probability distribution of the random variable is uniform [26, 27]. Since a *vid* consists of bits, when every bit changes independently and has the same uniform probability distribution in choosing 0

```
Input: original vid binary tree T
Input: decay factor ϵ
Output: updated vid binary tree T
(1)  foreach node r in T do
(2)    if P_r is uninitialized then
(3)      calculate P_r;
(4)    end
(5)    K = Random(0, 1);
(6)    if K < P_r then
(7)      Swap(T_r);
(8)      P_r := P_r − ϵ;
(9)    end
(10)   else
(11)     P_r := P_r + ϵ;
(12)   end
(13) end
(14) return T
```

Algorithm 2: Algorithm for subtree spin.

or 1, the degree of *vid* randomization reaches the maximum level. In order to produce this optimal result from the security perspective, the spinning possibility of node r should be set according to

$$P_r = \frac{1}{2 \times L_r}, \tag{3}$$

where L_r is the level of the *sid* tree that node r resides on. The root node is on level one. Algorithm 2 is an expression of this idea.

Apart from security, the costs for performing the subtree spinning on each node should also be taken into consideration. Since the spinning of each node's subtrees may not affect the same number of switches, the cost of spinning every node's subtrees is usually uneven. To provide a better network performance, we must agree that the more it costs to spin one node's subtrees, the less frequently the node's subtrees should be spun. Thus, we introduce a cost variable c_r to the original P_r, so we have

$$P_r = \alpha \times c_r + \frac{1 - \alpha}{2 \times L_r}, \quad 0 \le \alpha \le 1, \tag{4}$$

where α is the balance factor between security and network performance, which can be set according to current network situation or the administrator's preference. Equation (4) reflects both the considerations (MTD ability and traffic performance). *Subtree spinning* changes *vid*s, resulting in the need for routing table updates. Let *solid leaf node* be the leaf node that represents a network switch in a U-TRI network; then c_r is dependent on the count of *solid leaf nodes* affected by the *subtree spinning* of node r. Therefore, c_r can be calculated as (5), in which S_r denotes the set of *solid leaf nodes* of the tree whose root node is r, and c_{s_i} denotes the count of routing table entries updated at switch s_i.

$$c_r = \frac{\sum_{s_i \in S_r} c_{s_i}}{C}. \tag{5}$$

In (5), C is the count of route table entries in all switches (which can be easily recorded by the U-TRI server when route tables are created and updated).

Therefore, each time a *subtree spinning* is conducted, every node r in the *vid* binary tree has a probability of P_r to be chosen to spin its subtrees. And the *spinning rate* S is used to set the time interval between each *subtree spinning*, which can be easily modified by the network administrator to further balance the security and the performance.

4.4.4. Tree Shifting Strategy. VIRO is essentially a routing planning protocol. It splits the network into balanced subnetworks recursively so as to generate a *vid* tree and assign switches' *vid*s accordingly. The *vid* randomization is constrained by the structure of the current *vid* tree. For an instance, the Hamming distance between *vid*s of switch C and D would always be two bits no matter how the *vid* tree is spun because these two switches are put into the same 3-level subnetwork. This fact can be used by adversaries to recover linkage between the *sid* and the subnetwork after subtree spinning, which in turn lowers the difficulty to figure out the linkage between the *hid* and the host. However, in fact, switches C and D are not necessarily put into the same 3-level subnetwork. According to the physical topology, switch D can also be put into the same 3-level subnetwork with switch N in another *vid* tree, thus making linkage recovery between the *sid* and the subnetwork more difficult. Thus, in order to change the Hamming distances between *sid*s, we present the *tree shifting* strategy to reassign all *sid*s of switches.

For each partition of a network or subnetwork topology, the process is essentially a Maximally Balanced Connected Partition (MBCP) problem in graphs. Also, the costs of edge cuts between the partitioned networks should be lowered so as to obtain better network performance. Therefore, we combined the MBCP enumeration algorithm in [28] with TWO-WAY uniform partition algorithm in [29] and loosened the balance and edge-cut cost restrictions in such a way that more *sid* trees with optimal or suboptimal balance or edge-cut costs can be obtained for *sid* tree shifting.

For topology in Figure 7, the algorithm can generate 452 different *sid* trees (with parameter $b = \langle 1, 2, 1, 1 \rangle$ and $c = 20$), some of which are shown in Figure 6. Though both the MBCP and the TWO-WAY uniform partition problem are NP-hard, the execution of Algorithm 3 is supposed to complete within hours for a typical local network topology. Since it is an off-line algorithm and the result would be stored in the U-TRI server for later use in the network, its execution would not affect U-TRI's online performance.

Since a *sid* tree shifting affects more switches than a *subtree spinning*, its rate should be slower than that of *subtree spinning*.

5. Experiments

In this section, we present the evaluation on the performance of U-TRI through experimentation with our prototype implementation in a realistic network environment. We construct a real enterprise-like network topology as shown in Figure 7, where switches are denoted as circles and each edge switch

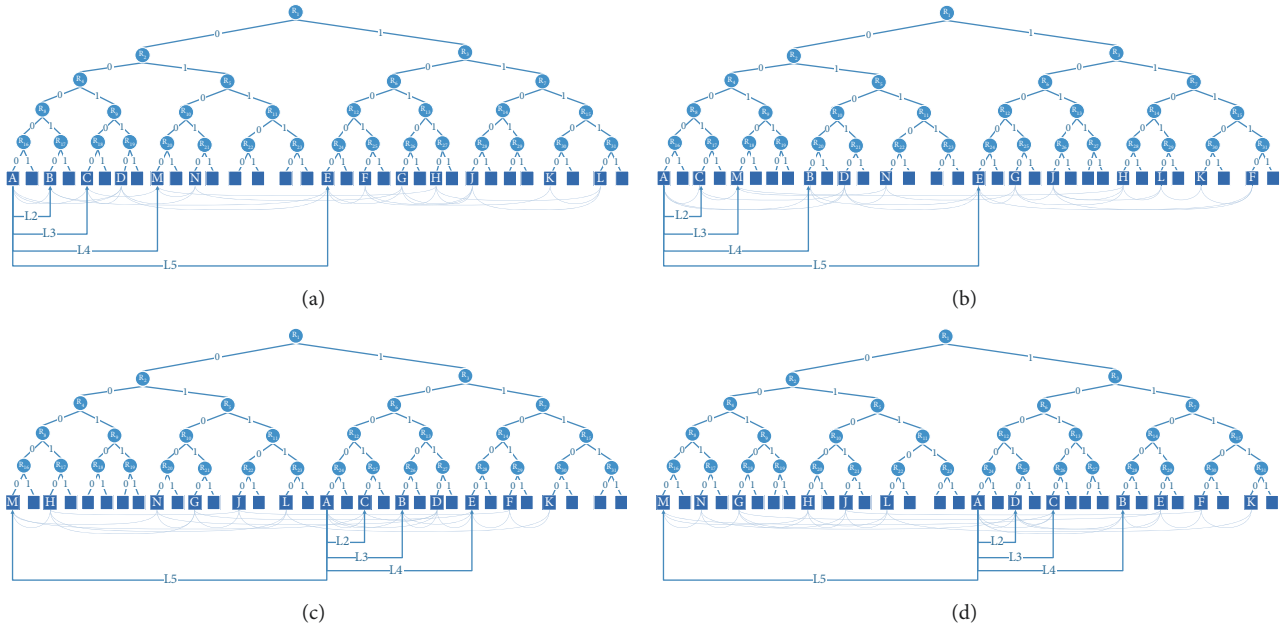


Figure 6: Different sid trees [8].

Input: network topology $G = \langle V, E \rangle$
Input: balance difference restriction $\overrightarrow{b}$
Input: cost restriction c
Input: current level L
(1) $P :=$ all partitions $(V1, V2)$ of V;
(2) **foreach** $p(V1, V2)$ *in* P **do**
(3) **if** $||V1| - |V2|| > b_L$
(4) *or* $G1 = \langle V1, E1 \rangle$ *is disconnected*
(5) *or* $G2 = \langle V2, E2 \rangle$ *is disconnected* **then**
(6) delete p from P;
(7) **end**
(8) **end**
(9) Sort ps in P in the descending order of edge cut cost;
(10) Keep the top c of ps and delete others;
(11) **foreach** $p(V1, V2)$ in P **do**
(12) Output tree node $n\langle V, V1, V2 \rangle$;
(13) Self call with parameter $G1, b, c, L + 1$;
(14) Self call with parameter $G2, b, c, L + 1$;
(15) **end**

Algorithm 3: Generate a *sid* tree set.

has an end-host connected. H_s stands for *server*, H_c stands for *client*, and other end-hosts generating background traffic are denoted as H_b. U stands for the U-TRI server. The SDN controller, which is implemented with RYU SDN Framework [30] running on a machine with Intel i7-3555LE (2.50 GHz), 4 GB memory, and 10 Gbps ethernet port, is connected to every switch through OpenFlow channels (not shown in Figure 7). Before any experiment, the network is first initialized. Firstly, each switch is assigned a *sid* and an initial routing table. Then secure channels between ULPs and the U-TRI server are established and ULPs register their first *hid*s.

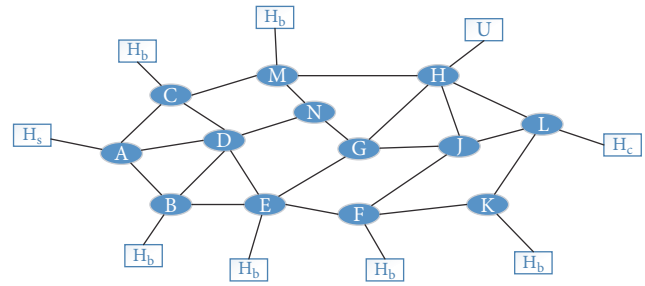


Figure 7: Experiment topology. Each switch is an Open vSwitch (2.7.0) [9] running on a device with Intel i7-4770 (3.40 GHz) CPU, 16 GB memory, and 6 full-duplex 1 Gbps ports [8].

End-hosts will start their communication with each other by sending ARP requests.

We carried out similar experiments as PHEAR [6] did. Two types of network users have been emulated: the interactive web browser and the bulk file downloader. The web browser fetches a 500 KB web page from a web server, while the bulk file downloader fetches a file of 500 MB from a file server. H_s acts as both the web server and the file server, and H_c acts as both the web browser and the file downloader. The update rates of identifiers are the key factors in communication performance loss. To evaluate the performance impact of U-TRI, we conducted three groups of experiments with different *vid* updating configurations for both types of typical network users. The results are shown in Figures 8, 9, and 10.

It can be seen from the figures that the download time for 70% of the web pages and bulk files is around 0.135 seconds and 12.5 seconds when the network is configured as a common SDN network, which utilizes STP protocol to construct the routing table. This performance is consistent

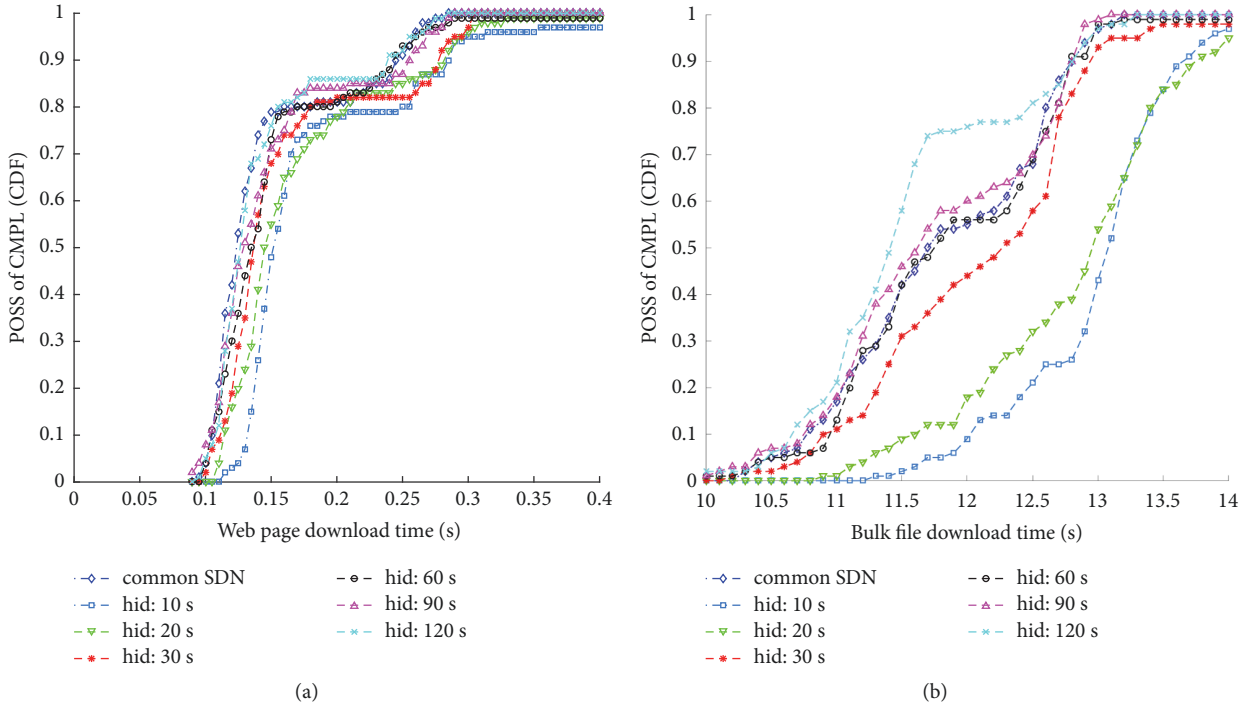


Figure 8: The effect of *hid* updating rates on the network performance.

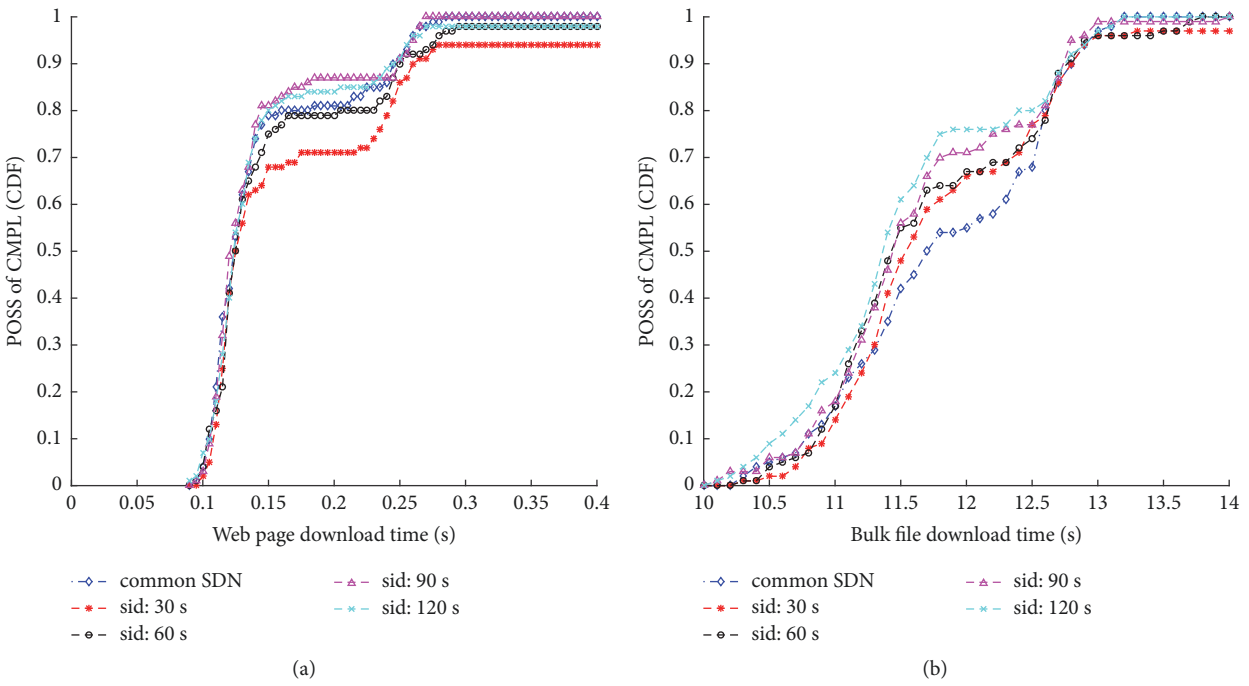


Figure 9: The effect of *sid* updating rates of the subtree spinning strategy on the network performance.

with that of a conventional local network and acts as the baseline of our experiments. In this setting, there is no performance cost introduced by U-TRI.

Figure 8 shows the impact of the update rate of *hid* on download time when we disable the updates of *sid*. The *y*-axis is the possibility of completion of one fetch in terms of the cumulative distribution function (CDF). It can be seen from Figure 8(a) that when the update rate of *hid* is set to per 120 s, 90 s, 60 s, and 30 s, the download time of 70% web page increases from 0.14 seconds to 0.155 seconds. As a result, the percentage of performance downgrade is between 3.7% and 14.8%. For most web users, this is barely noticeable,

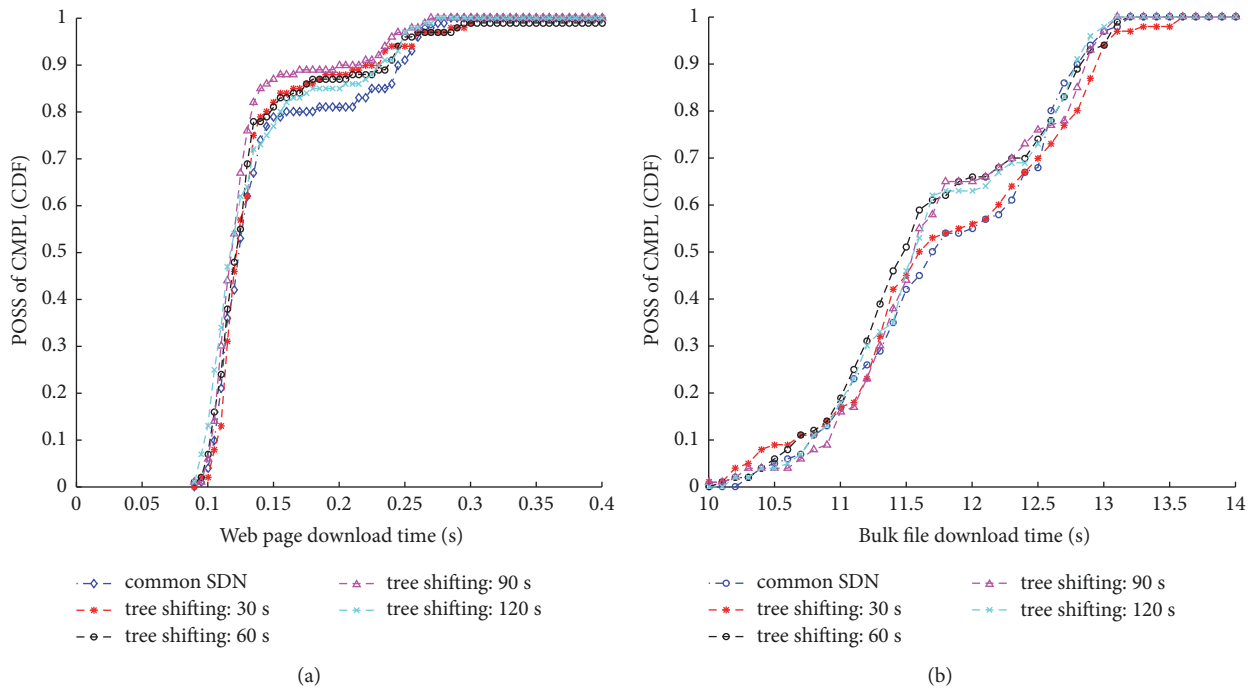


Figure 10: The effect of *sid* updating rates of the tree shifting strategy on the network performance.

considering the widely used Ajax technology for reducing the volume of web page downloading and the client-side cache of static web resources. However, for latency-sensitive web applications such as WebRTC, a short update rate of *hid* may cause lower QoS. For example, when the update rates of *hid* are set to per 20 s and per 10 s, the performance downgrades increase to around 25%, which may affect the performance of real-time web applications. In this case, the update rates of *hid*s should be set lower. However, for bulk file downloaders, the download times for 70% of the fetches are around 11.6 seconds when the update rates of *hid* are per 120 s, which is even less than that of the common SDN. This is due to the fact that the common SDN disabled some links for preventing loops by using the STP protocol while VIRO-based U-TRI can make the best of all network links. When setting the updating rates of *hid* to per 90 s and 60 s, the bulk file downloader can still obtain the same network performance as the common SDN network. When the updating rate of *hid* is increased to per 30 s, the download time reaches 12.7 seconds, with only 1.6% performance downgrade. When we further lower it to per 20 s or 10 s, the performance downgrade for file downloading increases to around 6%, which means a merely 0.75-second increase compared to the 12.5-second downloading time for the common SDN network. This is usually acceptable in a large file transferring scenario. We can see that the performance of bulk file downloading is less affected than that of web page downloading. It is because the processing time of *hid* update is extremely short compared to the total download time of the bulk file. Thus, for those hosts that mostly conduct bulk file downloading, a higher *hid* updating rate is practically acceptable.

Figure 9 shows the impact of the update rate of *sid* on download time when utilizing *subtree spinning*. It can be seen from Figure 9(a) that when the update rate of *sid* is set to per 120 s and 90 s, 70% of the web page downloads consume nearly the same amount of time as that of the common SDN network. When the rate is increased to per 60 s and 30 s, the web page download times are 0.145 seconds and 0.165 seconds, respectively, with performance downgrades from 7.4% to 22.2%. Thus, too quick *subtree spinning* harms the experience of web users. However, for file downloading users, the performance is rather attractive. We can see from Figure 9(b) that when the update rate of *sid* is set to per 120 s, 90 s, 60 s, and 30 s, the file download time increases from 11.7 seconds to 12.3 seconds, which results in 1.6% to 6.4% performance increment compared to the common SDN network.

We hope the ULP on end-hosts is lightweight in that it will not incur significant network accessing delay when mapping the outgoing packets. Therefore, we carried out an experiment measuring the latency between the moment ULP receiving a packet and the moment the packet leaving the end-host. The result is shown in Figure 11. We can see that 80% of the extra time consumed by ULP is within 40 microseconds, which cannot cause a sensible performance downgrade by users.

We choose four different trees (as shown in Figure 6) to be the candidate trees for evaluating performance impacts of *tree shifting* strategy in the experimental network. The result is shown in Figure 10. We can see that 70% of the bulk file download time increases from 12.4 seconds to 12.5 seconds, with 0 to 1.6% performance increment compared to the common SDN network. However, its impact on web users

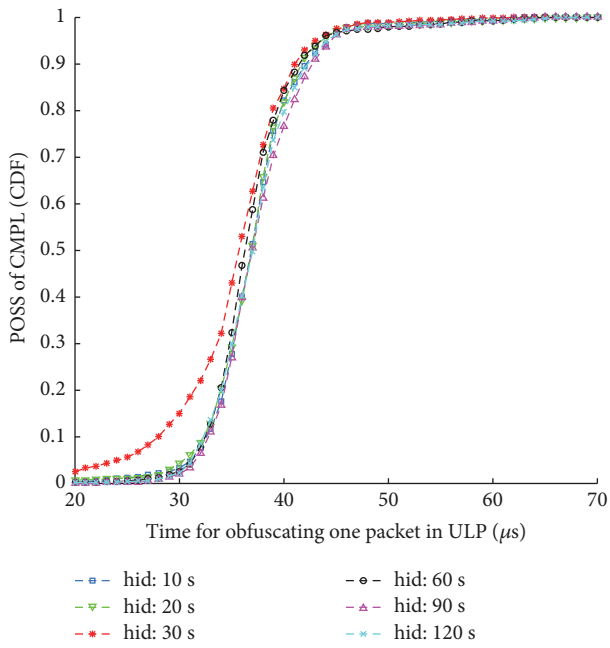


Figure 11: The performance of U-TRI local proxy.

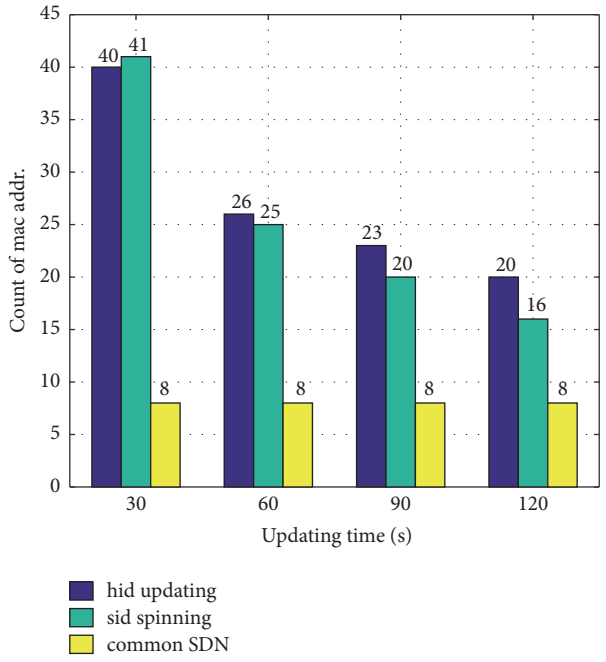


Figure 12: The security levels of different network configurations.

is more subtle; 70% of the web download time is vibrating around 0.13 seconds.

We also evaluate the security levels measured in MAC address count for different configurations, as shown in Figure 12. With the same number of hosts in the network and running the same kind of applications, the common SDN network presents only 8 different MAC addresses on the monitored switch port, while *hid* updating can provide 20 to 40 different MAC addresses, and *sid* updating can provide 16 to 41 different MAC addresses, in the same time windows on the same monitored switch port. Since both *sid* and *hid* will be updated when U-TRI is actually deployed in a real network, they will provide apparently more different identifiers than the common SDN network.

To test how strong U-TRI's defense is in a real attacking scenario, we carried out an initial red-team testing with the well-known traffic analysis tool Wireshark [31]. Firstly, we acted as an attacker that had compromised switch B and sniffed the packets it received. Then, we compared the sniffed packets which have been obfuscated by U-TRI (as shown in Figure 13(b)) with the original packets sent from host H_s (as shown in Figure 13(a)). To make the analysis easier, we used the data field that is intentionally set to a unique value to identify the packets. We can see from Figure 13 that both the MAC addresses and the IP addresses of the source and destination hosts are obfuscated.

Overall, the results of the experiment indicate that with proper parameter selection, U-TRI is capable of providing sufficient performance to support anonymous communication.

6. Analysis

6.1. Security Analysis. In this section, we analyze the security degree provided by U-TRI and the impact of attacks on U-TRI itself.

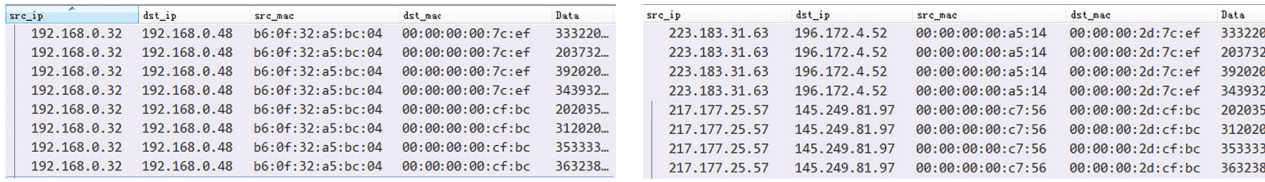
6.1.1. Unpredictability of U-TRI. U-TRI provides *unlinkability* to the network by updating *hid* and *sid*; thus the unpredictability level is determined by the update rates of these two types of identifiers. *hid* and *sid* can be updated independently. To simplify the analysis, we discuss their effects on unpredictability separately.

First we assume that *sid* will change while *hid* is static. From (2), we have

$$\begin{aligned}&\Pr\left(H\left(I_t\right) \mid H\left(I_{t-1}\right)\right)\\&\quad = 1 - \Pr\left(I_t \neq I_{t-1}\right)\\&\qquad + \Pr\left(I_t \neq I_{t-1}\right)\Pr\left(H\left(I_t\right) \mid H\left(I_{t-1}\right), I_t \neq I_{t-1}\right).\end{aligned} \tag{6}$$

If the adversary is in the switch whose identifier is I_{t-1}, he/she would definitely know to which value the identifier has changed, so $\Pr(H(I_t) \mid H(I_{t-1}))$ would be 1. The unlinkability is provided only by *hid* in this case. In other cases, *sid* changing would contribute to the unlinkability. Since the update of *sid* has two strategies, reassigning (denoted as R) through *tree shifting* and spinning (denoted as S) through *subtree spinning*, we have

$$\begin{aligned}&\Pr\left(H\left(I_t\right) \mid H\left(I_{t-1}\right), I_t \neq I_{t-1}\right)\\&\quad = \Pr\left(R\right)\Pr\left(H\left(I_t\right) \mid H\left(I_{t-1}\right),\ I_t \neq I_{t-1}, R\right)\\&\qquad + \Pr\left(S\right)\Pr\left(H\left(I_t\right) \mid H\left(I_{t-1}\right),\ I_t \neq I_{t-1}, S\right).\end{aligned} \tag{7}$$



src_ip	dst_ip	src_mac	dst_mac	Data
192.168.0.32	192.168.0.48	b6:0f:32:a5:bc:04	00:00:00:00:7c:ef	333220…
192.168.0.32	192.168.0.48	b6:0f:32:a5:bc:04	00:00:00:00:7c:ef	203732…
192.168.0.32	192.168.0.48	b6:0f:32:a5:bc:04	00:00:00:00:7c:ef	392020…
192.168.0.32	192.168.0.48	b6:0f:32:a5:bc:04	00:00:00:00:7c:ef	343932…
192.168.0.32	192.168.0.48	b6:0f:32:a5:bc:04	00:00:00:00:cf:bc	202035…
192.168.0.32	192.168.0.48	b6:0f:32:a5:bc:04	00:00:00:00:cf:bc	312020…
192.168.0.32	192.168.0.48	b6:0f:32:a5:bc:04	00:00:00:00:cf:bc	353333…
192.168.0.32	192.168.0.48	b6:0f:32:a5:bc:04	00:00:00:00:cf:bc	363238…

(a) Part of the packets captured at host H_s

src_ip	dst_ip	src_mac	dst_mac	Data
223.183.31.63	196.172.4.52	00:00:00:00:a5:14	00:00:00:2d:7c:ef	333220…
223.183.31.63	196.172.4.52	00:00:00:00:a5:14	00:00:00:2d:7c:ef	203732…
223.183.31.63	196.172.4.52	00:00:00:00:a5:14	00:00:00:2d:7c:ef	392020…
223.183.31.63	196.172.4.52	00:00:00:00:a5:14	00:00:00:2d:7c:ef	343932…
217.177.25.57	145.249.81.97	00:00:00:00:c7:56	00:00:00:2d:cf:bc	202035…
217.177.25.57	145.249.81.97	00:00:00:00:c7:56	00:00:00:2d:cf:bc	312020…
217.177.25.57	145.249.81.97	00:00:00:00:c7:56	00:00:00:2d:cf:bc	353333…
217.177.25.57	145.249.81.97	00:00:00:00:c7:56	00:00:00:2d:cf:bc	363238…

(b) Part of packets sniffed at switch B

Figure 13: The red-team result using Wireshark.

A reassignment of *sid* breaks the correlation between $H(I_t)$ and $H(I_{t-1})$, so we can obtain

$$\begin{aligned}&\Pr\left(H\left(I_t\right) \mid H\left(I_{t-1}\right),\ I_t \neq I_{t-1}\right)\\&\quad = \Pr\left(R\right)\Pr\left(H\left(I_t\right)\right)\\&\qquad + \Pr\left(S\right)\Pr\left(H\left(I_t\right) \mid H\left(I_{t-1}\right),\ I_t \neq I_{t-1}, S\right).\end{aligned} \tag{8}$$

Let $\bar{I}$ denote the conjugate identifier of I. We call switch s_a the conjugate switch of switch s_b if and only if s_a's identifier is the conjugate identifier of s_b's identifier. A conjugate identifier of a switch is the candidate identifier that would be swapped with the switch identifier in the event of subtree spinning. We use indicator function $1\{I\}$ to denote whether identifier I is currently possessed by a switch. With these definitions, we have

$$\begin{aligned}&\Pr\left(H\left(I_t\right) \mid H\left(I_{t-1}\right), I_t \neq I_{t-1}, S\right)\\&\quad = \Pr\left(1\left\{\bar{I}\right\} = 1\right)\\&\qquad \times \Pr\left(H\left(I_t\right) \mid H\left(I_{t-1}\right),\ I_t \neq I_{t-1}, S, 1\left\{\bar{I}\right\} = 1\right)\\&\qquad + \Pr\left(1\left\{\bar{I}\right\} = 0\right)\\&\qquad \times \Pr\left(H\left(I_t\right) \mid H\left(I_{t-1}\right),\ I_t \neq I_{t-1}, S, 1\left\{\bar{I}\right\} = 0\right).\end{aligned} \tag{9}$$

The existence of the conjugate switch for a target switch would affect the possibility for deciding whether a changed identifier still represents the original switch. We assume the packets from/to a switch and its conjugate switch appear on the monitored switch port with even possibility. The adversary cannot decide whether an identifier has been swapped to represent another switch when it has the conjugate switch, but would know that a newly appeared identifier actually represents the switch whose original identifier has suddenly disappeared on the monitored port. Thus, we have $\Pr(H(I_t) \mid H(I_{t-1}),\ I_t \neq I_{t-1}, S,\ 1\{I\} = 1) = \Pr(H(I_t))$ and $\Pr(H(I_t) \mid H(I_{t-1}),\ I_t \neq I_{t-1}, S,\ 1\{I\} = 0) = 1$. Substituting this result into (9), we have

$$\begin{aligned}&\Pr\left(H\left(I_t\right) \mid H\left(I_{t-1}\right),\ I_t \neq I_{t-1}, S\right)\\&\quad = 1 - \left(1 - \Pr\left(H\left(I_t\right)\right)\right)\Pr\left(1\left\{\bar{I}\right\} = 1\right).\end{aligned} \tag{10}$$

Given a network topology, there is no guaranteed solution to ensure that every switch has a conjugate switch. However, we can add dummy packets with conjugate identifiers to the network, so that from the view of the adversaries, every switch does seem to have a conjugate switch. In this case, $\Pr(1\{\bar{I}\} = 1) = 1$. Therefore, we have

$$\Pr\left(H\left(I_t\right) \mid H\left(I_{t-1}\right),\ I_t \neq I_{t-1}, S\right) = \Pr\left(H\left(I_t\right)\right). \tag{11}$$

Combining (11) with (8), we have

$$\begin{aligned}&\Pr\left(H\left(I_t\right) \mid H\left(I_{t-1}\right),\ I_t \neq I_{t-1}\right)\\&\quad = \Pr\left(R\right)\Pr\left(H\left(I_t\right)\right) + \Pr\left(S\right)\Pr\left(H\left(I_t\right)\right)\\&\quad = \Pr\left(H\left(I_t\right)\right).\end{aligned} \tag{12}$$

Substituting this result into (6), we obtain

$$\begin{aligned}\Pr\left(H\left(I_t\right) \mid H\left(I_{t-1}\right)\right) = \Pr\left(H\left(I_t\right)\right)\Pr\left(I_t \neq I_{t-1}\right) + 1\\ - \Pr\left(I_t \neq I_{t-1}\right).\end{aligned} \tag{13}$$

Combining (13) with (1), we can obtain the final result:

$$\begin{aligned}\Pr\left(H\left(I_t\right) \mid H\left(I_{t-1}\right)\right) = 1\\ - \left(1 - \frac{1}{|P_m|}\right)\Pr\left(I_t \neq I_{t-1}\right).\end{aligned} \tag{14}$$

Therefore, the probability of broken unlinkability is determine by the size of the port anonymous set (reflected by $|P_m|$) and the identifier changing rate (reflected by $\Pr(I_t \neq I_{t-1})$).

The *sid* changing rate is determined by the *tree shifting* rate and the *subtree spinning* rate; both can be set by the U-TRI server centrally. The *tree shifting* rate provides larger moving space for identifiers, while *subtree spinning* brings less updates to routing tables. Therefore, it is reasonable to set a lower value for the former and a higher value for the latter so as to obtain a better combination of moving space and moving speed.

The moving space of *sid* is determined by the *virtual binary tree*, which is defined to be a full binary tree or else *sid*s will have different lengths. A specific network may correspond to more than one such trees, whose quantity is determined by both the topology of the network and the length of the *sid* field (which is 16 bits in our current design, as depicted in Figure 5). Each subtree represents a subnetwork of the network. In order to gain a maximum transmit performance, the network needs to be divided recursively according to the constraints of both MBCP problem and TWO-WAY uniform partition problem as discussed in

Section 4.4.4. If in some recursion, more than one division exists, the minimum height virtual binary tree of the network would not be unique (to assign the subnet to be a left or right subtree does not make them two different trees). For any minimum height virtual binary tree of the network, when there is no null non-leaf tree node (whose subtrees do not contain any switches), the different number of assignments for *sid*s can reach up to

$$N_a = 2^{2^{h-1}-1} \tag{15}$$

in which h is the height of the virtual binary tree. For the topology in Figure 7, the minimum height of its virtual binary tree is 5; thus U-TRI can provide theoretically 2^{15} different assignments of *sid*s for a specific virtual binary tree. Since the upper bound of the height of the virtual binary tree is 16 bits (which is equal to the length of the *sid* field), we can also generate a higher virtual binary tree for the same network to enlarge the size of moving space for *sid*s (but with more dummy packets to show up IDs for leaf nodes without switches). For the example topology in Figure 7, we actually generated 452 different virtual binary trees whose height is 6; each supports up to 2^{31} different assignments of *sid*s. It is a large space for 13 switches to change their *sid*s.

The *hid* update rate can be totally configured by the ULP independently, depending on the security requirements of end-hosts and its performance constraints. The length of the *hid* is 32 bits, which provides 2^{32} different identities for hosts. Also with the fact that updates of the *hid* and the *sid* are asynchronous (as described in Section 4.4), U-TRI is able to provide sufficient unpredictability.

6.1.2. Attacks Thwarted by U-TRI

Traffic Analysis Attacks. U-TRI obfuscates identifiers by mapping real values to virtual ones, preventing adversaries from easily linking hosts to packets. Meanwhile, the unpredictable nature of the U-TRI routing identifier *vid* makes the task of correlating captured packets to up layer sessions complicated, especially when the changing rates of the *sid* and *hid* are high.

However, U-TRI cannot guarantee the impossibility of completing this task through statistical analysis. Some side channel information still may be leaked by U-TRI that could allow attackers to learn more about the network. This is due to the following facts which mostly concerned with network performance.

Firstly, U-TRI does not want to search the application layer payload for identifiers that may appear in any location since such a search and the corresponding hiding and recovering processes would dramatically hurt switching performance. Therefore, U-TRI only hides identifiers which appear at fixed places so there is no need for location searching. Thus, U-TRI chooses to hide the identifiers in the data link layer (MAC addresses), network layer (IP addresses), and parts of the transport layer (TCP/UDP ports). However, we should note that IP address and TCP/UDP port may also appear in the application layer. For an instance, the From, To, Via, and Path fields of SIP packets could all have IP addresses and TCP/UDP ports as part of their values. This is the compromise U-TRI has to make to maintain network performance

Secondly, U-TRI does not change the lengths of packets. Packet length randomization can be achieved by padding extra random bits at the end of the packet prior to sending it to an egress port. However, such an operation would not only incur additional checksum computing but may also exceed the limitation of MTU for some links along its path and incur unnecessary IP fragmentation, thus causing great performance penalty on network performance. On the other hand, some protocols such as BitTorrent, HTTP, and PPStream do have distinguishable distributions on packet length [32]. Although U-TRI makes it hard to decide whether a packet belongs to a well-known protocol or not, if the majority of traffic in the network belongs to such a protocol, the attackers can still recognize it once collecting a large enough amount of packets and analyzing its packet length distribution.

Thirdly, U-TRI manages to maintain the routing cache mechanism of the OpenFlow switch, which in the case of OVS is the *fast path* mechanism. Fast path works in the kernel space which makes switching very efficient. Normally, the first packet of a flow would undergo a *flow path* in the OVS switch, which triggers the process of adding a new flow entry in the switch's flow table. This process is relatively slow partly because it happens in the user space of the OS. However, once the flow entry is added, the ensuing packets of the same flow will go through the *fast path* in the kernel space of the OS, which is much faster. However, if we want to send out packets to randomly chosen egress ports, we break the prerequisite of *fast path* and can only choose *slow path* whose operations are carried out in the user space, thus being much slower. Therefore, U-TRI basically will not change the packets' routes except when the *nexthop* values in route tables were changed due to *tree shifting*. U-TRI is based on VIRO which aims to effectively switch packets by selecting the shortest path possible between end-nodes. Though the IDs in the packets are changed by U-TRI, the packets are usually still the same collection of packets traversing the switch. When the adversary sniffs on the port of an edge switch and finds that it is much busier than most other ports, then he can infer that it is very likely that the sniffed port is connected to a server. Combined with packet length distribution analysis, he may figure out the type of the server (e.g., email server or web server).

Fourthly, unlike onion routing in which packet identifiers change each hop, U-TRI still reserves a time window between two consecutive identifier updates in which packet identifiers remain unchanged. With a larger time window, it is easier to link packets to an ongoing end-to-end session by using unchanged identifiers. This is the common issue of MTD solutions. MTD aims to *decrease* the staticity of a system, not to get rid of it. The static nature is bad for network security but will contribute to other network properties such as high performance and easy management. U-TRI provides tunable parameters for users to choose. It is up to the network administrator to balance between security and other metrics.

Finally, since periodical control messages always exist in an U-TRI network, when user network traffic is inactive, these messages are relatively noticeable. Although their content

is encrypted by IPsec, identifying characteristics such as packet length and connection volume can still be helpful to adversaries when recognizing these control messages and obtaining system configurations such as the rates of *subtree spinning* or *tree shifting*.

Fortunately, all of the above problems can be mitigated to a large extent when configuring U-TRI carefully and combining it with other countermeasures. The updating rate of *hid* and *sid* should be set as high as possible within the tolerance of network performance so as to leave a minimum time window for static identifiers. Combined with IPsec ESP for protecting payload from the network layer up to the application layer, no original IP addresses or TCP/UDP ports would be leaked accidentally. Dummy packets with a length distribution that is the reverse of that of normal packets in the protected network can be injected into various links between OpenFlow switches, in order to disturb the normal packet length distribution. With some randomly embedded features (e.g., port pattern or MAC address pattern), dummy packets can be filtered out before leaving the protected network if they cause any malfunction beyond the protected network.

Linkability Attacks. In this section, we analyze the impact on end-host unlinkability when different components of the U-TRI network are compromised.

Among all distinct types of components in a U-TRI network, the end-host, which is usually poorly protected by average users due to lack of security expertise, is most vulnerable to attacks. Once an end-host is compromised, since the U-TRI protocol is completely transparent to end-hosts, identifiers of both the compromised end-host and all of its communicating peers will be exposed to the adversary. What is more, if the ULP is running on the compromised end-host, the adversary can also learn the local-stored mapping from the original identifiers to *vid*s as well as the identifier updating rate.

Another vulnerable component in U-TRI is the switch. There are more and more vulnerabilities being found in switches including SDN switches (e.g., CVE-2016-2074, CVE-2017-1000357). Adversaries may compromise network switches and monitor traffic transmitted in them. U-TRI limits linkability to ingress/egress port anonymity sets by changing *sid*s and *hid*s periodically. Therefore the possibility of the adversary identifying a link between two end-hosts is determined by the size of the anonymity sets of the respective switch ports connected directly/indirectly to these two end-hosts. Usually, edge switches provide weaker linkability protection than core switches since one end of the link can be easily identified due to the fact that most of their ports are connected to only one end-host. However, the other end of the link is more difficult to identify since the size of its corresponding anonymity set is usually very large. In the extreme case, if the network consists of only ONE switch, no linkability is provided.

The SDN controller is supposed to be the most secure component in U-TRI. Under most circumstances, it should be always trusted and not be compromised. However, if it is compromised, it will be a catastrophic security event because mappings from identifiers to *vid*s are entirely exposed to adversaries. Unlinkability is therefore totally lost. What is worse, malicious control messages can also be purposely sent to switches, interfering with normal network activities.

6.1.3. Attacks on U-TRI Mechanism. Since we do not conceal U-TRI to adversaries, attacks targeting U-TRI mechanism could happen. In this section, we analyze three possible attack surfaces of the U-TRI mechanism and provide countermeasures.

Identifier Updating Rate. Knowing only the rate of *subtree spinning* or *tree shifting* does not actually break unlinkability, but this information can assist further attacks such as traffic analysis. That is, the adversary can capture and group network traffic according to different time windows determined by the rate. Configuring the rate nonconstant is a solution. A capricious identifier updating rate can obfuscate the moments of *subtree spinning* or *tree shifting*.

Logical Distance of Vids. According to Theorem 1, logical distances between any pairs of *vid*s remain unchanged after *subtree spinning*. This property may be utilized by attackers to group packets by different logical distances. However, with a higher rate of *tree shifting*, which usually changes the logical distance of *vid*s, this threat can be eliminated.

Vid Exhaustion. The essence of *subtree spinning* is exchanging existing *sid*s instead of assigning new ones. Thus, *subtree spinning* does not have a *vid* exhaustion problem. Also, the length of *sid* can accommodate at most 2^{16} switches, which is sufficient for a large real local network to obtain a new *sid* reassignment through *tree shifting*. *hid* needs to be unique in the network, but its length (32 bits) can support *hid* updating for hundreds of thousands of end-hosts, which is enough for practical use and will not easily lead to vid exhaustion in a real scenario.

6.2. Property Maintenance Analysis. In this section, we focus on proving that *subtree spinning* keeps the property of VIRO, thus maintaining VIRO's merit on routing efficient and failure recovery.

Theorem 1. *The logical distance for any pair of sid remains unchanged after a swap.*

Proof. Given a *binary tree* T representing an L-bit *sid* space, let V be the set of all nodes in T excluding leaf nodes and M be the set of all leaf nodes (VIRO switches). The theorem is true if and only if the following predicate is true:

$$\delta_T(x, y) \equiv \delta_S(x, y) \quad \forall v \in V \wedge \forall x,\ y \in M, \tag{16}$$

in which S is the tree transformed from T by spinning and $\delta_T(x, y)$ and $\delta_S(x, y)$ are the *logical distance* of nodes x and y in the tree T and S, respectively.

Let v_l be the left child of v and v_r the right child, in terms of the virtual tree T_v (as shown in Figure 3(a)); the location of

x and y in T can and only can be in one of the four following conditions.

Both in the Same Subtree (T_{v_l} or T_{v_r}). Without loss of generality, we assume x and y are both in subtree T_{v_l}. Since $\delta(x, y) = L - lcp(x, y)$ and L is constant, we only need to prove that $swap(T_v)$ does not change $lcp(x, y)$. Since the only one bit $swap(T_v)$ that has changed is the bit between v_l and v which is the common bit of $sid(x)$ and $sid(y)$ (as shown in Figure 3), $lcp(x, y)$ remains the same as it was after $swap(T_v)$.

One in T_{v_l} and the Other in T_{v_r}. Since x and y are under different subtrees of T_v, their *longest common prefix* must stop at v from the root of T. $swap(T_v)$ cannot affect any bit alongside the path from *root* to v; thus $lcp(x, y)$ remains unchanged.

One in T_v and the Other out of T_v. $lcp(x, y)$ should be the length from *root* to the parent node of v in this situation. $swap(T_v)$ does not change anything neither.

Both out of T_v. It goes without saying that $swap(T_v)$ will not affect $lcp(x, y)$ under such circumstance.

Therefore, Theorem 1 is proved. □

Since a *subtree spinning* is composed of finite times of *swap* operations on different nodes, given Theorem 1, the following can be concluded.

Corollary 2. *The logical distance for any pair of sids remains unchanged after a subtree spinning.*

Now given Theorem 1 as well as the fact that an original VIRO *sid* space binary tree T was built strictly according to the *closeness property* and the *connectivity property*, we will prove that these two properties will not be violated by *swap*.

Theorem 3. *The closeness and connectivity properties hold after swap.*

Proof. We will prove these two properties one by one.

Closeness Property. Since T is a binary tree which meets the requirement of *closeness property* and a *subtree spinning* does not change the *logical distance* of any two nodes according to Theorem 1, apparently the *closeness property* will not be violated.

Connectivity Property. Since T is a binary tree which meets the requirement of *connectivity property*, it must meet the *connectivity* equation represented in [7]

$$\exists z \in B_k(x) \wedge \exists y \in S_{k-1}(x) : (y, z) \in E \tag{17}$$

in which $B_k(x)$ is the set of all nodes which are at logical distance of k from node x, and $S_k(x)$ represents the set of all nodes which are at no more than logical distance of k from node x. That is, $S_k(x) = B_1(x) \cup B_2(x) \cup \cdots \cup B_k(x)$.

Based on Theorem 1, it is clear that $B_k(x)$ will remain the same after an action of *swap* for $1 \le k \le L$. Thus, $S_k(x)$ will also remain the same. Therefore, if before *swap*, y and z, which are physically connected, belong to $S_{k-1}(x)$ and $B_k(x)$, respectively, they would still be in those sets after *swap*. As a result, (17) still holds for the network after *swap*. Thus the *connectivity property* will not be violated. □

Notice that now we have proved the consistency of the two properties restraining VIRO *sid* space before and after a *swap*; for the same reason mentioned above, the following is concluded.

Corollary 4. *The* closeness property *and the* connectivity property *will keep inviolated after a* subtree spinning.

Theorem 5. subtree spinning *does not change the* nexthops *in a routing table physically.*

Proof. We shall prove this theorem by mathematical induction. Note that in the routing table of one switch A, a *nexthop* at level-k is the neighbor of A through which A can reach its level-k gateway while this gateway is at a logical distance no more than k to A.

Let G_x be the gateway that is x logical distance away from A.

With Corollary 2 given and thus the logical distances of A and its neighbors are not changed, it is trivial to prove that when the gateway is G_0, i.e., the gateway is A itself, *subtree spinning* does not change the gateway's corresponding *nexthop*s.

Now suppose that when the gateway is G_m ($0 \le m \le k$, $k \ge 0$), *subtree spinning* does not change the gateway's corresponding *nexthop*s. We shall prove that the corresponding *nexthop*s of G_{k+1} will still not be changed by *subtree spinning*. In fact, to reach G_{k+1}, we first need to find the level-$k + 1$ gateway, and the *nexthop* to reach the level-$k + 1$ gateway is the *nexthop* to reach G_{k+1}. (Do make it clear that "level-k gateway" does not mean that the gateway is k logical distance away, but the gateway is physically connected to a node that is k logical distance away.) Recalling the definition of the gateway, we know that a level-$k + 1$ gateway is at a logical distance no more than k logical distance away from A. From the assumption, *subtree spinning* does not change the corresponding *nexthop*s of G_m ($0 \le m \le k$, $k \ge 0$) and this level-$k+1$ gateway is surely included in G_m. Thus, the *nexthop* to reach the level-$k + 1$ gateway is not changed, resulting in the invariability of the *nexthop* reaching G_{k+1}. Therefore the theorem is proved. □

Theorem 5 gives us the mathematical confidence when updating *sid*s that we only need to recalculate prefixes of each level, without concern about the *nexthop*s since they proved to be invariable physically. That is to say, no matter how subtrees are spun, the output port of each level in the routing table should remain the same. Or rather, *subtree spinning* does not change the path through which packets travel the network, just the identifiers.

7. Conclusion

The moving target approach is promising for rebalancing the cyber landscape in favor of the defense. By changing the

network attack surface randomly while sustaining network services for normal users, defenders can drastically increase attacking costs for adversaries. In this paper, we proposed a random namespace scheme U-TRI that is able to hide the majority of the identifiers in packets. We chose a hierarchical virtual namespace to ensure packet switching efficiency and added randomization to provide attack surface moving capability. We also utilized the centralized software-based network control ability of SDN to implement U-TRI. U-TRI can provide both unlinkability and efficient network service in SDN networks, while its basic theory should be able to be applied in conventional networks. Our next step work would be refining the moving strategy, taking network traffic trending into consideration, so as to obtain a better balance between security and performance.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The work was supported by the National High-tech R&D Program of China (863 Program) (2015AA017201). The authors are very grateful for the VIRO source code sharing by Zhi-Li Zhang and Braulio Dumba from University of Minnesota Twin Cities.

References

[1] P. W. Chi, C. T. Kuo, J. W. Guo, and C. L. Lei, "How to detect a compromised sdn switch," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft '15)*, pp. 1–6, London, UK, April 2015.

[2] Y. C. Chiu and P. C. Lin, "Rapid detection of disobedient forwarding on compromised OpenFlow switches," in *Proceedings of the 2017 International Conference on Computing, Networking and Communications (ICNC '17)*, pp. 672–677, Silicon Valley, Calif, USA, January 2017.

[3] A. V. Arzhakov and I. F. Babalova, "Analysis of current internet wide scan effectiveness," in *Proceedings of the 2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus '17)*, pp. 96–99, Moscow, Russia, Feburary 2017.

[4] A. Pfitzmann and M. Hansen, "Anonymity, unobservability, and pseudonymity—a proposal for terminology," 2008, https://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.31.pdf.

[5] R. A. Haraty and B. Zantout, "The TOR data communication system," *Journal of Communications and Networks*, vol. 16, no. 4, pp. 415–420, 2014.

[6] R. Skowyra, K. Bauer, V. Dedhia, and H. Okhravi, "Have no phear: networks without identifiers," in *Proceedings of the 2016 ACM Workshop on Moving Target Defense (MTD '16)*, pp. 3–14, ACM, Vienna, Austria, October 2016.

[7] S. Jain, Y. Chen, Z.-L. Zhang, and S. Jain, "VIRO: a scalable, robust and namespace independent virtual Id routing for future networks," in *Proceedings of the IEEE INFOCOM 2011*, pp. 2381–2389, April 2011.

[8] W. Yulong, C. Qingyu, Y. Junjie, and G. Jun, "U-tri: unlinkability through random identifier for sdn network," in *Proceedings of the 2017 Workshop on Moving Target Defense (MTD '17)*, pp. 3–15, ACM, Dallas, Tex, USA, October 2017.

[9] B. Pfaff, J. Pettit, T. Koponen et al., "The design and implementation of open vSwitch," in *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI '15)*, pp. 117–130, May 2015.

[10] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow: enabling innovation in campus networks," *SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[11] S. Sezer, S. Scott-Hayward, P. Chouhan et al., "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[12] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: an intellectual history of programmable networks," *SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[13] V. Liu, S. Han, A. Krishnamurthy, and T. Anderson, "Tor instead of IP," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks (HotNets-X '11)*, pp. 14:1–14:6, ACM, Cambridge, Mass, USA, November 2011.

[14] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "An effective address mutation approach for disrupting reconnaissance attacks," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 12, pp. 2562–2577, 2015.

[15] Q. Duan, E. Al-Shaer, and H. Jafarian, "Efficient random route mutation considering flow and network constraints," in *Proceedings of the 2013 IEEE Conference on Communications and Network Security (CNS '13)*, pp. 260–268, IEEE, National Harbor, Md, USA, October 2013.

[16] S. Venkatesan, M. Albanese, K. Amin, S. Jajodia, and M. Wright, "A moving target defense approach to mitigate DDoS attacks against proxy-based architectures," in *Proceedings of the 2016 IEEE Conference on Communications and Network Security (CNS '16)*, pp. 198–206, October 2016.

[17] Y. Song and B. D. Fleisch, "Utilizing binary rewriting for improving end-host security," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 12, pp. 1687–1699, 2007.

[18] Y. Chen, "Aptshield: a fine-grained detection system for remote access trojan in the apt attacks," http://free.eol.cn/edu_net/edudown/spkt/chenyan.pdf.

[19] A. Niakanlahiji and J. H. Jafarian, "Webmtd: defeating web code injection attacks using web element attribute mutation," in *Proceedings of the 2017 Workshop on Moving Target Defense (MTD '17)*, pp. 17–26, ACM, Dallas, Tex, USA, October 2017.

[20] S. Kent and K. Seo, "Security architecture for the internet protocol," RFC 4301, 2005, https://tools.ietf.org/html/rfc4301.

[21] Y. Gilad and A. Herzberg, "Spying in the dark: Tcp and tor traffic analysis," in *Privacy Enhancing Technologies*, pp. 100–119, Springer, 2012.

[22] Apache, "Apache jmeter application," https://jmeter.apache.org/.

[23] Alfresco, "Alfresco community edition," https://www.alfresco.com/alfresco-community-editions.

[24] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, Eds., *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, Springer, 2011.

[25] S. Jajodia, A. K. Ghosh, V. Subrahmanian, V. Swarup, C. Wang, and X. S. Wang, Eds., *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*, Springer, 2012.

[26] I. Csiszar, "Maximum entropy and related methods," in *Proceedings of the 1994 Workshop on Information Theory and Statistics*, pp. 11–15, 1994.

[27] D. MacKay, *Information Theory, Inference and Learning Algorithms*, Cambridge University Press, 2003.

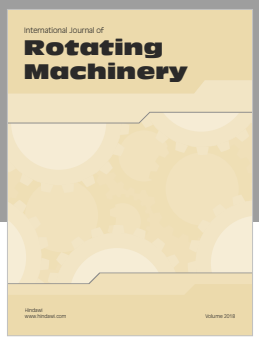
[28] D. Matic and M. Bozic, "Maximally balanced connected partition problem in graphs: application in education," *The Teaching of Mathematics*, vol. 15, no. 2, pp. 121–132, 2012.

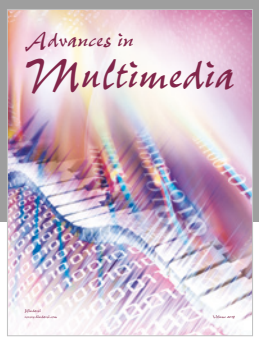
[29] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Labs Technical Journal*, vol. 49, no. 1, pp. 291–307, 1970.

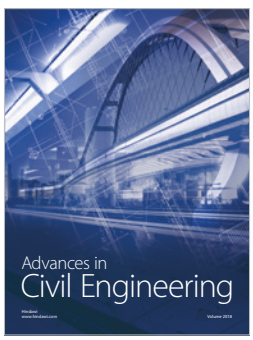
[30] R. Kubo, T. Fujita, Y. Agawa, and H. Suzuki, "Ryu SDN framework-open-source SDN platform software," *NTT Technical Review*, vol. 12, no. 8, pp. 1–5, 2014.

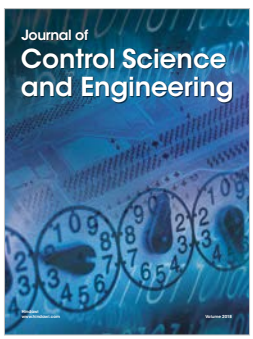
[31] Wireshark, https://www.wireshark.org/.

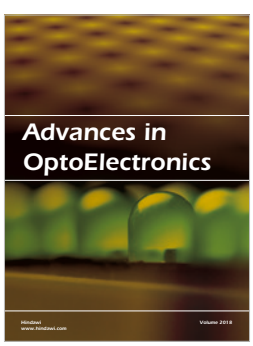
[32] X. L. Wu, W. M. Li, F. Liu, and H. Yu, "Packet size distribution of typical internet applications," in *Proceedings of the International Conference on Wavelet Active Media Technology and Information Processing (ICWAMTIP '12)*, pp. 276–281, Chengdu, China, December 2012.

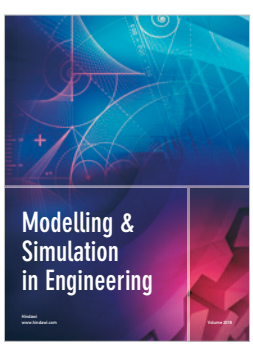


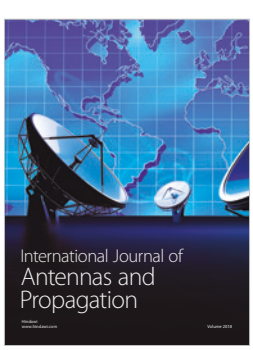


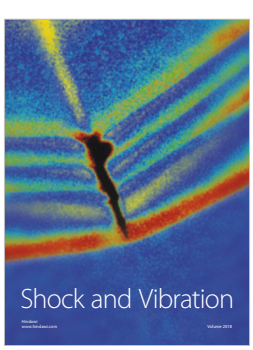


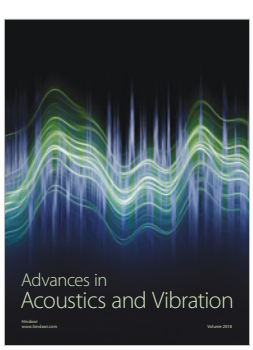












Hindawi

Submit your manuscripts at
www.hindawi.com