U-TRI: Unlinkability Through Random Identifier for SDN Network

Yulong Wang
State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications
No.10 Xi Tu Cheng Rd., Hai Dian District
Beijing, P.R.China 100876
wyl@bupt.edu.cn

Qingyu Chen
School of Computer Science, Beijing University of Posts and Telecommunications
No.10 Xi Tu Cheng Rd., Hai Dian District
Beijing, P.R.China 100876
chenqy@bupt.edu.cn

Junjie Yi
State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications
No.10 Xi Tu Cheng Rd., Hai Dian District
Beijing, P.R.China 100876
yijunjie@bupt.edu.cn

Jun Guo
State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications
No.10 Xi Tu Cheng Rd., Hai Dian District
Beijing, P.R.China 100876
guoj@bupt.edu.cn

ABSTRACT

Traffic analysis within switches is threatening the security of large enterprise networks built with SDN. Adversaries are able to monitor all traffic traversing a switch by exploiting just one vulnerability in it and obtain linkage information 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 novel theory for re-obtaining 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 identifier, which is one of the most vital attack surfaces of traffic privacy, within packet data units. U-TRI employs an independent, hierarchically-structured, periodically and randomly changing identifier to replace the original static data link layer addresses. It also hides all other identifiers in the network and transport layer by obfuscating them. Such a combination of hierarchical random address and obfuscated identity enables U-TRI to provide unlinkable communications among hosts. The result of experiments indicates that U-TRI is capable of defending traffic analysis with very little burdens on network performance.

CCS CONCEPTS

• **Networks** → **Network privacy and anonymity**; *Logical / virtual topologies*; *Packet-switching networks*; • **Security and privacy** → *Security protocols*;

MTD'17, October 30, 2017, Dallas, TX, USA.
© 2017 ACM. ISBN 978-1-4503-5176-8/17/10…$15.00
DOI: https://doi.org/10.1145/3140549.3140554

KEYWORDS

Moving Target Defense; Software-Defined Networking; Unlinkability

1 INTRODUCTION

Traffic analysis is a major threat faced by enterprises with large local networks. Through wiretap [20, 27] or compromised switches [3, 4], adversaries are able to sniff large amounts of traffic traversing in the enterprise networks. Packet headers of network protocols are originally designed for the benefits of service locating and packet delivery. However, they end up as an important information source of traffic pattern for attackers, since implicit and explicit identifiers can be found in multiple layers of the network stack. With these identifiers, attackers can gather intelligence such as which hosts are online, who is talking to whom, what kind of tasks is probably being carried out and which are the key hosts, etc.[2] Since packet sniffing is a kind of passive attack, it is very hard to detect. What's worse, when attackers are capable of penetrating into a switch through a vulnerability [1], they would be able to perform a large scale reconnaissance, even with only a few compromised switches. Therefore, it is critical to implement unlinkability in the local network. Existing anonymity networks have already been focusing on removal or obfuscation of implicit/explicit identifiers. Tor [10] 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, not 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 a relatively high latency and low robustness, which is acceptable for the Internet but apparently unsuitable for a local network. PHEAR [25] 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 network layer and exposing only the *nonce* for routing. However, the nonce represents a pair of source and destination, which results in great exhaustion to the resources of routing devices since they have to store up to n^2 routing entries for a network with n end-hosts. The root cause to this disadvantage is that the *nonce* is a flat identifier without a hierarchical structure. Besides, encrypting all data beyond network layer would prevent the operation of other SDN applications. We present U-TRI to address these issues.

Our solution. Aiming at meeting the gaps described above, U-TRI is designed leveraging a novel virtual id (denoted as *vid*) protocol called VIRO [12], which constructs a tree-like *vid* space for packets to be routed effectively in the network. Given that VIRO is a static protocol, U-TRI adopts it in such a way that the *vid* can be changed unpredictably while the advantages of VIRO is also reserved. We proposed a technique called *sub-tree spinning* to achieve this goal. By spinning sub-trees of the virtual binary tree guided by a properly designed *pseudo-random* strategy and updating routing tables accordingly, U-TRI manages to put *vids* in a consistent changing procedure while still support effective routing. Meanwhile, U-TRI obfuscates all identifiers with the network and transport layers in order to prevent identifier leakages. However, the main contribution and focus of this work are on MAC address obfuscation, the obfuscation of other identifiers will only be briefly discussed. The recently blooming SDN technology provides us an effective way of implementing U-TRI. An MTD server is responsible for *sub-tree spinning* as well as updating routing tables in OpenFlow switches through an SDN controller. A light-weight proxy running on end-host accomplishes the mapping between original packets and anonymous packets. This way, U-TRI provides an effective unlinkable communication that is transparent to the applications in end-hosts.

Evaluation. We critically evaluate two aspects of a U-TRI network: performance and security. We set up a real network with physical switches and end-hosts, and test the performance of the network with several metrics. 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.

Contributions. The key contributions of this paper are 1. it's the first work that provides unlinkability to enterprise networks by using structured 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 do 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 real physical network with background traffic.

Roadmap. The rest of the paper is organized as follows. Section 2 overviews the background knowledge and the existing work on which our work based. Section 3 describes the design details of U-TRI. Then we provide the experimental evaluation of U-TRI in section 4 as well as analysis of security in section 5. We also present the mathematical analysis in section 6. Section 7 discusses research works related to U-TRI. Finally, Section 8 concludes the whole paper and describe the next step work.

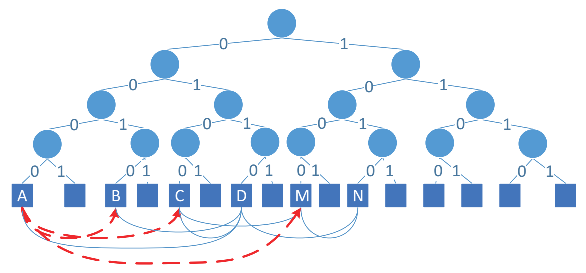
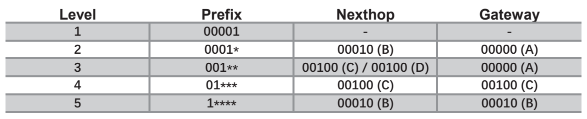


Figure 1: The vid tree in VIRO



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)

Figure 2: An example of a VIRO routing table

2 BACKGROUND

In this section, we briefly describe the vid routing mechanism of VIRO and the SDN technology.

2.1 VIRO Vid Routing

VIRO [12] introduces a *structured vid space* onto which physical identifiers are mapped, as shown in Figure 1. VIRO utilizes a *virtual binary tree* to organize its *vid 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 id. Except for the root node of the *vid* tree, which represents the whole network, each node in the *vid* tree represents a subnet on a different level. The multi-level structure of *vids* is very helpful in host aggregation and decreasing broadcast.

The *vid space* structure of VIRO is built critically under the restrain of two invariant properties: *the Connectivity Property* and *the Closeness Property*[12]. The *Connectivity Property* ensures host reachability using *vid*-based routing. And the *Closeness Property* ensures that logical distance[1] can reflect physical distance (measured in hops). These two properties together make sure a VIRO network to be an effective switching network. Note that any modification to VIRO should not violate these two properties in order to keep the advantages of VIRO. In section 6 we will prove our adoption still reserves them.

VIRO switches use VIRO routing tables to route packets. A logical instance of a VIRO routing table consists of four columns: Level, Prefix, Nexthop[2] and Gateway[3]. Figure 2 is an example of the constructed routing table of node A in Figure 1. A group of switches

[1]The *logical distance* δ of two $vids$ 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 $vid(y)$.
[2]Nexthop: The neighbor to reach a level-k gateway.
[3]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.

that are at level-k of A is those switches that are at a logical distance of k to A. Plus, 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 output to the corresponding port of the nexthop. The Gateway and Level columns are only used when building the routing table.

2.2 Software-Defined Networking

Software-Defined Networking (SDN) [7, 17, 24] decouples the control plane which manages where traffic is sent and the data plane which does the real packet forwarding, allowing networks to be dynamically initialized, controlled, changed and managed through programmatic methods from a single point. 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 to the SDN controller through its northbound API to add, modify or remove flow rules to manipulate the network's behavior. U-TRI is designed to be such an application that utilizes the capabilities of the SDN technology.

3 U-TRI

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

3.1 Threat Model

U-TRI's threat model is similar to that of PHEAR[25]. In an enterprise network, the adversaries may monitor, modify, replay, reroute, drop, delay, or inject packets in flight on their compromised fraction of the network. Plus, flow tables on these compromised infrastructures are also observable to them. Since SDN controller itself is 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 it is communicating. We believe the reason is that providing *unlinkability* from an end-host violates the basic requirement of current networking services provision in that the services cannot be found without an identifier (e.g. address). As shown in Figure 3, inevitably the *sender* needs the address of the *receiver* to communicate with it. If the *sender* is compromised, the hacker can naturally gain the address of the *receiver*. Therefore, compromised end-host is not part of our threat model. Thus, we assume that the SDN controller, authenticated end-hosts, U-TRI server(described later in section 3.4.2) are in the trusted domain, while the switches and other end-hosts are all in the untrusted domain.

U-TRI is a dedicated solution for thwarting threats faced by SDN networks. However, its basic theory should be able to be applied in conventional networks.

3.2 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* [22] of communications from the view of network switches. The degree

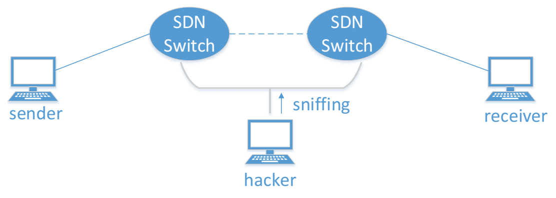


Figure 3: The threat model of U-TRI

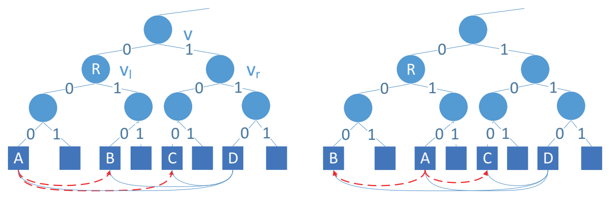


Figure 4: Example of Sub-tree Swapping of node R (Part of the vid tree in Figure 1)

of *unlinkability* is actually dependent on the size of the anonymity set of each port of a switch. The anonymity set of one 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 in the communication of two hosts, thus traffic between hosts are easily linked when in-between switches are compromised. To ensure communicating hosts unlinkable, we need to make linkable identifiers, especially source and destination 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 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 sub-networks. Our main concern now becomes how to put *vid* into an unpredictable, periodically changing process. This can be simply done by completely re-assigning different *vids* to the entire network every period of time, but re-assigning would cause the following problems. Firstly, completely re-assigning costs network resources, especially when re-assigning needs to be done fastly. Secondly, it is not flexible enough in terms of balancing network performance and MTD ability, since multiple sceneries may occur in practice. Thirdly, the exhaustion of *vid* may also be a great problem. Given the fact that VIRO's *vid* space is constructed on a *structured* binary tree, we manage to develop a technique called: *sub-tree spinning*. *Sub-tree spinning* basically means swapping the two sub-trees of some randomly chosen nodes (excluding leaf nodes, which represents a switch each) in the binary tree in an unpredictable manner. By swapping, for example (Figure 4), node A's *vid* becomes *0010*, and node B's *vid* becomes *0000*. If the *vid* space of a VIRO network is *spun* periodically and randomly, every node in the network is then not fixed to any *vid*, thus providing traffic source and destination *unlinkability* to the network.

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 light-weight program running on authorized end-hosts whose only function is the mapping 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[14]. 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 more specifically in section 3.4.

Do notice that other identifiers above the data link layer such as IP addresses and TCP/UDP ports can also leak linkage information to the adversaries [8]. U-TRI adopted IPsec to encrypt the IP payload of its control traffic, i.e., the messages sent between the ULP and the U-TRI server. Plus, mark that IPSec does not cover the IP address field, we simply set the IP address of every packet (both control and user traffic) to a dummy value, since the role of IP addresses in packet routing and forwarding is now taken over by U-TRI's *vid*. For user traffic, however, IPSec encryption is not a must (but can also be well supported) because other SDN applications such as deep packet inspection may rely on the upper layer content. Meanwhile, U-TRI will activate a mapping mechanism to rewrite important identifiers such as the TCP/UDP port field (only restricted to well-known ports since other ports do not relate to a specific type of service) 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. This mapping mechanism is trivial, so we will not discuss it further.

3.3 Virtual Identifiers

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

3.3.1 Sid. The leaf nodes shown in VIRO's *vid* binary tree only represents switches, thus in U-TRI, the name for the *vid* of switches is re-defined as the *sid*. The *sid* represents switches and is constructed according to the binary tree. The U-TRI Server is responsible for the generation and assignments of the *sid*. In this paper, we use *sid* plus subscripts to represent a switch in the network. The subscript letter can both denote a switch or an end-host. When it is an end-host, the whole expression refers to the edge switch of the end-host.

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

3.3.3 Vid. *vid* is the concatenation of *sid* and *hid*. U-TRI re-purposed the MAC address field for *vid*. We divide the 48 bits into two subfields, one to represent the *sid*, one to represent the *hid* (as shown in Figure 5). The *sid* subfield is 16 bits long while the *hid* subfield is 32 bits long. In this paper, when we talk about the *vid* of

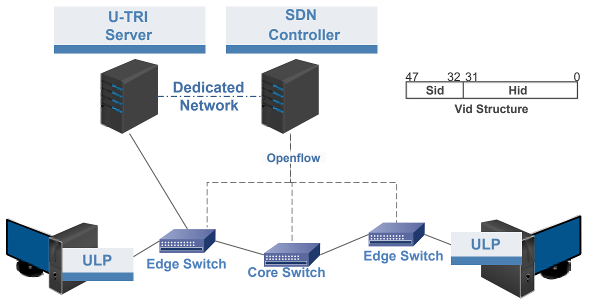


Figure 5: U-TRI architecture

a switch, the *hid* subfield is all 0; when we talk about the *vid* of an end-host, the *sid* subfield should be the *sid* of its edge switch.

3.4 Architecture

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

3.4.1 OpenFlow Switch and Controller. The SDN controller opens up APIs for the U-TRI Server to control OpenFlow switches. It is connected with the U-TRI Server using a dedicated network, 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[4] must be installed at OpenFlow switches. There are at most four flow tables at each OpenFlow switch, as shown in Figure 6.

Table 0: The Entering Table. Every packet would pass 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 at edge switches would have one more rule: to match on ARP request packets, change their destination MAC addresses to the *vid* of the U-TRI Server and re-submit them to the Routing table so they can be transferred to the U-TRI Server.

Table 1: The Host-Forwarding Table. This table deals with packets sent by local end-hosts, thus only edge switches have this table. Rules in this table match on packets with all 0s as their destination *sid* (since these packets are definitely sent out by end-hosts), replace the 0s with the proper destination *sid* and re-submit them to the Routing table so that they can be transferred to the destination switch.

Table 2: The Routing Table. It is the implementation of the VIRO routing table shown in Figure 2. 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, restore the destination *sid* of the matched packets to all 0s and re-submit them to the Host-Receiving table so that they can be sent to the corresponding local end-hosts.

Table 3: 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 to the *hid*'s corresponding egress port. These *hid*s should include only the *hid*s of local end-hosts.

[4]Flow table pipelining is not supported until OpenFlow 1.1[26].

The existence of the Host-Forwarding table and the Host-Receiving table ensures that *sid* update is transparent to end-hosts. End-hosts find their communication peers by peers' unique *hid*s, the destination *sid*s have nothing to do with them. The *sid* is only used when transferring packets, thus is not presented in the packet before the packet enters the edge switch's Host-Forwarding table. The Host-Receiving table makes sure the packet is sent to the destination that the *hid* indicates. The U-TRI Server needs not to inform any end-hosts when updating *sid*s, it only has to update the Host-Forwarding tables as well as the Routing tables.

3.4.2 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 network used to communicate with the SDN controller. It has three functions: building initial Routing tables, answering ULP, and executing *sid* update.

Building Initial Routing Tables. When the network initially bootstraps, no flow tables exist. The U-TRI Server needs to install flow tables with initial flow rules at network switches. These flow rules mainly exist in the Routing table, which is built in a centralized manner as opposed to the distributed mechanism of VIRO. SDN gives the U-TRI Server the ability to know the whole topology of the network. Thus, with the network's full topology and the *sid*s of every switch known, we can generate the Routing table for every switch using Algorithm 1.

Answering ULP. There are 3 types of request that a ULP would send to the U-TRI Server.

The first one is the *registeration.* When a newly authorized end-host (denoted as X) with a proper ULP running on is connecting to the network initially, its ULP will first 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 re-purposed 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 not answer the request immediately. Instead, the U-TRI Server will add a new flow rule to the Host-Forwarding table of E_X, matching packets on hid_Y as the destination and change the destination *sid* of them to sid_Y. Plus, it stores the new communication pairs X and Y in a *compair* table. After all 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 having or used to have communications with X, so that they can get

Algorithm 1: Routing Table Build

```
   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         $level \leftarrow$ calcVidDistance($sw$, $nb$);
 4         $port \leftarrow sw$.getConnectingPort($nb$);
 5         $prefix \leftarrow$ 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 $sw_a \neq sw_b$ 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         $gw \leftarrow sw.rt$.getGateway($k$);
21         $nh \leftarrow sw.rt$.findNexthopRecursively($gw$);
22         $port \leftarrow sw$.getConnectingPort($nh$);
23         $prefix \leftarrow$ getPrefixWithLevel($sw.vid$, $k$);
24         $sw.rt$.addOutputToLevel($prefix$, $port$);
25     end
26 end
```

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 *compair* 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-Forwarding 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 3.5.2.

3.4.3 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 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. Take MAC

The Entering Table (table 0)		
Match	Action	Priority
ARP_REQUEST	mod_eth_dst(vid_{server}), goto table 2	HIGH
sid == 0	goto table 1	MID
*	goto table 2	LOW

(a) The Entering Table

The Routing Table (table 2)	
Match	Action
prefix	output to port
self.*sid*	goto table 3

(c) The Routing Table

The Host-Forwarding Table (table 1)	
Match	Action
hid_{dst}	mod_eth_dst(sid_{dst} + hid_{dst}), goto table 2

(b) The Host-Forwarding Table

The Host-Receiving Table (table 3)	
Match	Action
local *hid*	output to port

(d) The Host-Receiving Table

Figure 6: Four Types of Flowtables in U-TRI (rows surrounded by dashes and dots are meant to be more than one in the table)

address for example, the ULP rewrites the source MAC address with its end-host's *hid* on sending and restore the destination *hid* to its end-host's MAC address on receiving. Identifiers in the network and transport layer will be processed in a similar manner, except that they will not be used for routing.

The function transparency is a desired attribute of a practical MTD solution. The ULP does not modify any APIs or libraries of the operating system but 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. Moreover, since the destination addresses stored in end-host's arp table is actually *hid*s instead of *real* MAC addresses, the ULP also can help to disrupt reconnaissance when end-host is compromised.

3.5 Vid Update

In this section, we introduce how we update *vid*s in detail. To get things more complicated for the adversaries, we separate the update of *hid* and *sid*. Their update is asynchronous and independent.

3.5.1 Hid Update. *hid* update is started by individual end-host (or its ULP to be exact). To update its *hid*, the ULP simply sends an *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 3.4.2. An *hid* update can happen any time at any end-host, making the change of *hid*s in the network totally unpredictable.

3.5.2 Sid Update. *sid* update is started by the U-TRI Server. Each time when a new round of *sid* update is needed, all the U-TRI Server has to do are: (1) generating new *sid*s with *sub-tree spinning*. (2) modifying existing Host-Forwarding tables and Routing tables with new *sid*s to ensure communications can still proceed[5]. Notice that **no end-hosts need to be informed** of an *sid* update, therefore if we ignore the delays of flow tables updating, theoretically *sid* update can be of no end-to-end communication performance penalty.

The generation of new *sid*s is strictly controlled by the *spinning strategy*, which we will discuss in depth in section 3.6. Here, we focus on introducing the process of the modification of flow tables. Assume that each switch already has a new *sid*. Modifying the Host-Forwarding table is trivial, simply updating old sid_{dst} to the new sid_{dst} would finish the job (Figure 6(b)). Modifying the Routing table, however, needs a little mathematical reasoning in advance. We shall provide the conclusion here, a detailed process of the reasoning is included in section 6.

[5]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.

The conclusion is: 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 (as described in section 2.1) 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 *sub-tree spinning* does not change the nexthops in a Routing table physically, according to Theorem 6.5.

3.6 Spinning Strategy

When designing the *spinning strategy*, there are a few principles should be met: (1) The conduction of *sub-tree spinning* should be random in terms of spinning nodes selection; (2) Physical topology should be taken into consideration when sub-trees are chosen, for an instance, any pair of sub-trees whose leaf nodes is empty (no corresponding physical nodes) should never be swapped. (3) The strategy should cover every node in the network, making sure their *vid*s won't stay unchanged (for a relatively long time). In fact, the strategy is the answer to the question: *How to choose nodes in the binary tree periodically so that the spinning of their sub-trees makes the best of MTD ability?* We note that *sub-tree spinning* consists of multiple *swap* operations. That is, $swap(a)$ is the operation of exchanging the left and right sub-trees of binary tree a. Let T be the *vid* binary tree, p_t be the spinning possibility of node t, R be *sub-tree spinning rate* and T_t denote a sub-tree of T whose root node is t, we propose an heuristic *spinning strategy* as in Algorithm 2. This algorithm will be executed every R seconds to periodically change *vid*s in the network.

Now we will discuss the justness of this strategy. Before all, we must make it clear that an ideal spinning strategy 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 makes the best of MTD ability. What MTD essentially does is to help the defender get the upper hand by making the attacker operate in an uncertain and unpredictable environment [9]. Thus, our goal is to provide the most unpredictable system by using our spinning strategy. Notice that in U-TRI, *vid* of network nodes is the only variable that stays exposed. Thus, if the spinning strategy changes *vid*s completely randomly, the unpredictable goal is

Algorithm 2: Sub-tree Spin

```
Input  : original vid binary tree $T$
Output : updated vid binary tree $T$
1 foreach node $t$ in $T$ do
2     calculate $p_t$;
3     $K$ = Random(0, 1);
4     if $K < p_t$ then
5         Swap($T_t$)
6     end
7 end
8 return $T$
```

reached. 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 [5, 16]. Since a *vid* consists of bits, when every bit changes independently and has the same uniform probability distribution in choosing 0 or 1, the degree of *vid* randomization reaches the maximum level. Therefore, we arrive at the conclusion that a spinning strategy that provides the best MTD ability is a strategy in which every node t in the vid binary tree has a probability p_t of 1/2 to switch its two sub-trees. Algorithm 2 is just an expression of this idea.

Now we take traffic performance into consideration. We have already concluded that a p_t of 1/2 for every node is the best for MTD ability, however, this conclusion is based on the hypothesis that the cost of spinning every node is equal. Nevertheless, when considering traffic performance, given the fact that any action of spin on one node has a cost corresponding to the count of affected switches, the cost of spinning every node is usually unequal. To provide a better network environment, we must agree that the more cost the spinning of one node takes, the less frequently the node should be spun. Thus, we introduce a tuning variable α_t to the original p_t (whose value is 1/2) so we have:

$$p_t = \alpha_t + \frac{1}{2}, \quad -\frac{1}{2} \le \alpha_t \le \frac{1}{2} \tag{1}$$

Equation (1) reflects the two considerations (MTD ability and traffic performance) together. Before deciding α_t, we must know how to calculate the cost of one spinning of any node t (which will be denoted as C_t later).

Sub-tree spinning changes *vid*s, resulting in the need for routing table update. Let *solid leaf node* be "leaf node that represents a network switch in a U-TRI network", then C_t is dependent on the count of *solid leaf nodes* affected by the *sub-tree spinning* of node *t*. Therefore, C_t can be calculated as Equation (2), in which T denotes the set of *solid leaf node* of the tree whose root node is t, and C_{s_i} denotes the count of routing table entries updated at switches.

$$C_t = \sum_{s_i \in T} C_{s_i} \tag{2}$$

Now that we have assigned C_t to t as the cost of one *sub-tree spinning*, α_t can be obtained with equation (3),

$$\alpha_t = \left(\frac{1}{2} - \frac{C_t}{C}\right) * \beta \tag{3}$$

where C is the count of route table entries in all switches (which can be easily recorded by U-TRI server when route tables are created and updated). β ($0 \le \beta \le 1$) is a scale factor used to adjust the weight of traffic performance in strategy constructing.β can be set by network administrators according to his preference on security or performance.

Therefore, each time a *sub-tree spinning* is conducted, every node t in the vid binary tree has a probability of(p_t to be chosen to spin its sub-trees. And the *spinning rate R* is used to set the time interval of every *sub-tree spinning*, which can be easily modified by the network administrator to further balance the security and the performance.

4 EXPERIMENTAL EVALUATION

In this section, we present the evaluation on the performance of U-TRI through experimentation with our prototype implementation in a realistic network environment.

4.1 Experiment Design

We construct a real enterprise-like network topology as shown in Figure 7, where switches are denoted as circles and each edge switch 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 [23] running on a machine with Intel i7-3555LE (2.50 GHz), 4 GB memory and 10Gbps ethernet port, is connected to every switch through OpenFlow channels (not shown in Figure 7). Before any experiment, the network is firstly initialized. Firstly, each switch is assigned an *sid* and an initial Routing table. Then secure channels between ULPs and the U-TRI Server are established and ULPs registered their first *hid*s. End-hosts will start their communication with each other from sending ARP requests.

We carried out similar experiments as PHEAR [25] did. Two types of network users are emulated: the interactive web browser and the bulk file downloader. The web browser fetches a 500KB web page from a web server, while the bulk file downloader fetches a file of 500MB from a file server. Each fetch is conducted 500 times without overlapping. 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 rate of *sid* and *hid* is the key factor of communication performance loss. To evaluate the performance impact of U-TRI, we conducted two groups of experiments with different *vid* updating configuration for each of the aforementioned two types of typical network users. The results are shown in Figure 8.

4.2 Experiment Analysis

It can be seen from figure 8 that the download time of 80% web pages and bulk files are around 0.027 seconds and 7.3 seconds respectively when *hid* and *sid* are never updated. This performance is consistent to that of a conventional local network and acts as the baseline of our experiments. In this setting, the only extra performance cost is the mapping operation of ULP in the end-host and it is neglectable. The ARP processing mechanism in U-TRI can potentially increase performance since it avoids ARP broadcast.

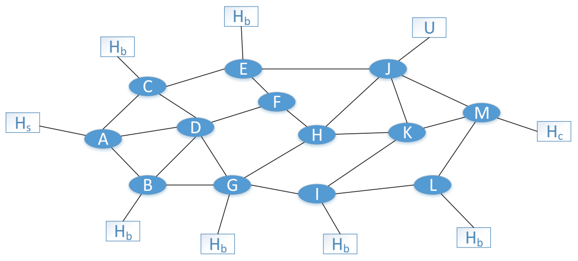


Figure 7: Experiment topology. Each switch is an Open vSwitch (2.7.0) [21] running on a physical device with Intel i7-4770 (3.40GHz) CPU, 16 GB memory and 6 full-duplex 1Gbps ports.

Figure 8(a) and Figure 8(b) show the impact of the update rate of *hid* on download time when we fix the update rate of *sid*. The y-axis is the possibility of completion of one fetch in terms of cumulative distribution function (CDF). *hid* is updated every 45s, 105s and 165s.

It can be seen from figure 8(a) that when the update rate of *hid* is set to per 165s, 105s and 45s (the update rate of *sid* set to 90s), the web page download time increased from 0.027 seconds to 0.028, 0.033 and 0.041 seconds respectively. As a result, the performance downgrade is between 3.7% to 51.9%. Therefore, for latency-sensitive users such as interactive web user, the update rate of *hid* should be set lower. However, for bulk file downloaders, the download time of 80% of fetches increased from 7.3 seconds to 7.35 seconds(figure 8(b)), even when the rate of *hid* update is increased to per 45s (the update rate of *sid* set to 90s), with only 0.7% performance downgrade. This is due to the fact that the processing time of *hid* update is extremely short compared to the total download time of the bulk file. Thus, for those hosts mostly conducting bulk file downloading such as hosts in CDN networks, a higher *hid* updating rate is practically acceptable.

Figure 8(c) and Figure 8(d) show the impact of the update rate of *sid* on download time. It can be seen from figure 8(c) that when the update rate of *sid* is set to per 120s, 90s, 60s and 30s, the increments of web page download time are almost the same, i.e. from 0.027 seconds to around 0.032 seconds. Thus, the performance downgrade is about 18.5%. Note that this doesn't mean this performance loss is caused by the update of *sid* since *hid* is also updating simultaneously. If we look closely at the line with setting *hid:105s, sid:90s*, we will find that the performance loss is solely caused by the update rate of *hid*. This result confirms our predicate in section 3.5.2, that *sid* update **does not harm the ULP's efficiency**. Figure 8(d) shows that the update rate of *sid* has little impact on the performance of bulk file downloading, i.e. 1.8% downgrade in the worst case. Combined with the result of figure 8(b), we can conclude that the update of *vid* has little negative influence on the download time of a bulk file. Overall, the results of the experiment indicate that U-TRI is capable of providing sufficient performance to support typical network traffic.

5 SECURITY ANALYSIS

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

5.1 Unpredictability of U-TRI

The unpredictability level of U-TRI is flexible. U-TRI provides *unlinkability* to the network by updating *hid* and *sid*, thus the unpredictability level is determined by the update rates of the two types of identifiers. The *sid* update rate, i.e. the *spinning rate* can be set by the U-TRI Server centrally to be relatively fast to present more distinct *sids* to the network (it does not affect communications between end-hosts anyway, as confirmed in section 4.2). The *hid* update rate can be totally configured by the ULP independently, depending on the security requirements of end-hosts and its performance constraints. Both rates are not required to be fixed, configurations can be made to apply random update rates to both types of identifiers.

The moving space of *sid* is determined by the *virtual binary tree*, which is defined to be a full binary tree or else *sids* 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 sub-tree represents a subnet of the network. In order to gain a maximum transmit performance, the network needs to be divided along the minimum edge cut set recursively. If in some recursion, more than one minimum edge cut sets exits, the minimum height virtual binary tree of the network would not be unique (the assignment of a subnet to be a left or right sub-tree is not seen as two different trees). For a minimum height virtual binary tree of the network, when there are no null non-leaf tree node (whose sub-trees don't contain any switches), the different number of assignments of *sids* can reach to

$$N_a = 2^{2^{h-1}-1} \tag{4}$$

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 *vid*. Since the upper bound of the height of 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 *vid*. In our experiments, we actually generated a virtual binary tree whose height is 6, which supports 2^{27} different assignments of *vids* (excluding the *vids* that involves 4 null non-leaf tree nodes). It's a large space for 13 switches to change their *sids*.

The length of *hid* is 32 bits, which provides 2^{32} different identities for hosts. Together with the fact that the updates of *hid* and *sid* are asynchronous (as described in section 3.5), U-TRI is able to provide sufficient unpredictability.

5.2 Attacks Thwarted by U-TRI

5.2.1 Traffic Analysis Attacks. U-TRI obfuscated identifiers by mapping real values to fake ones, preventing adversaries from trivially linking endpoints to messages. Meanwhile, the unpredictable nature of U-TRI routing identifier *vid* makes the task of correlating captured packets to up layer sessions complicated, especially when *spinning rate* is high. U-TRI cannot guarantee the impossibility of completing this task through statistical analysis, but dramatically complicates it by randomly updating the vid binary tree. In different *spinning periods* the same *vids* may belong to different sessions,

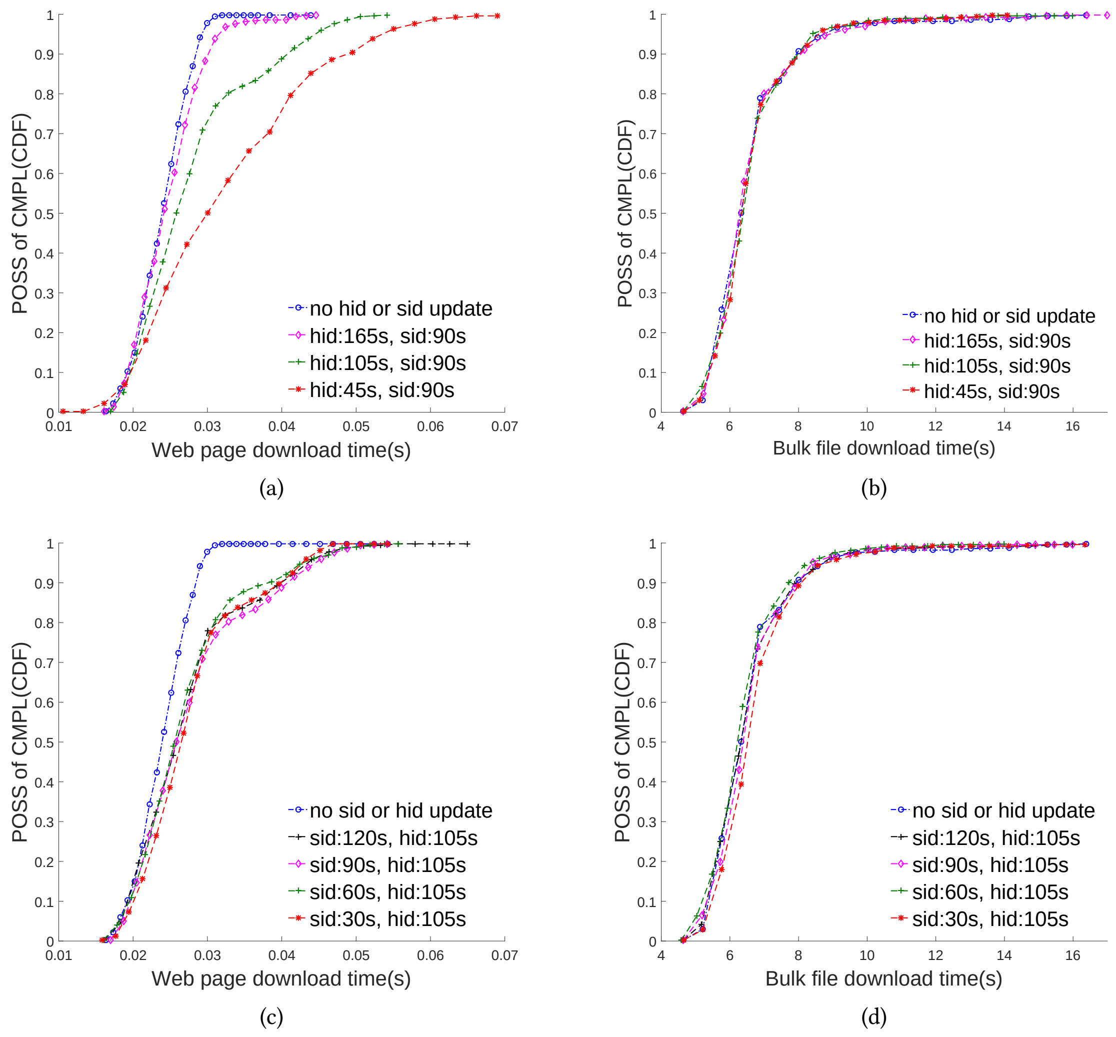


Figure 8: Experiment results. (a) and (b) are conducted with an *sid* update of 90s, (c) and (d) are conducted with an *hid* update of 90-120s

with a high *spinning rate*, the difficulty of traffic analysis can be greatly increased.

However, unlike onion routing in which packet identifiers change each hop, U-TRI still reserves a time window between two adjacent *sub-tree spinning* in which packet identifiers remain unchanged. With a larger time window, it is easier to link packets to an ongoing end-to-end connection by using unchanged identifiers.

What's more, 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 size and connection volume can still be used by adversaries to recognize these control messages and obtain system configurations such as *spinning rate*.

The statistical analysis based on traffic volume cannot be defeated solely by U-TRI. Nevertheless, dummy packets with appropriate sizes can be injected by the SDN controller into various links between OpenFlow switches, in order to change the original volume distribution of the protected network traffic (e.g. email or web). In this way, this problem can be further mitigated.

5.2.2 Linkability Attacks. In this section, we analyze the impact on source and destination unlinkability when different components of 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 the most vulnerable to be attacked. 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's more, if the ULP is running on the compromised end-host, the adversary can also learn the local-stored mapping from original identifiers to *vid*s as well as the *spinning 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 interested in the linkability of networks can compromise network switches and monitor traffic transmitted in them. U-TRI limited linkability to ingress/egress port anonymity sets by changing *sid*s and *hid*s periodically. Therefore the possibility that the adversary can identify a link between two end-hosts is determined by the size of 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 posts 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.

Controllers are supposed to be the most secure component in U-TRI. Under most circumstances, they should be always trusted and not be compromised. However, if they are 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's worse, malicious control messages can also be purposely sent to switches, interfering normal network activities.

5.3 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 U-TRI mechanism and provide countermeasures.

Spinning Rate. Knowing only *spinning rate* 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 spinning periods. Configuring *spinning rate* not to be constant is a solution. A capricious *spinning rate* can obfuscate the moment of *sub-tree spinning*.

Logical Distance of Vids. According to theorem 6.1, logical distance between any pair of *vid*s remain unchanged after *sub-tree spinning*. This property may be utilized by attackers to group packets by different logical distances.

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

6 MATHEMATICAL ANALYSIS

In this section, we focus on proving why *sub-tree spinning* does not affect VIRO's original network design.

Theorem 6.1. *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:

$$\forall v \in V \wedge \forall x, y \in M, \delta_T(x,y) \equiv \delta_S(x,y),$$

in which, S is the tree transformed from T by spinning, $\delta_T(x,y)$ 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 4(a)), the location of x and y in T can and only can be in one of four conditions:

Both in the same sub-tree (T_{v_l} or T_{v_r}). Without loss of generality, we assume x and y are both in sub-tree 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 $swap(T_v)$ 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 4), $lcp(x,y)$ remains the same as it was after $swap(T_v)$.

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

One in T_v, one 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 $swap(T_v)$ won't affect $lcp(x,y)$ under such circumstance.

Therefore, theorem 6.1 is proved. □

Since a *sub-tree spinning* is composed of finite times of *swap* operations on different nodes, given theorem 6.1, it can be concluded that:

Corollary 6.2. *The logical distance for any pair of sids remains unchanged after a sub-tree spinning.*

Now given theorem 6.1 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 6.3. *The closeness and connectivity properties hold after swap.*

Proof. *Closeness Property.* Since T is a binary tree which meets the requirement of *closeness property* and a *sub-tree spinning* does not change the *logical distance* of any two nodes according to theorem 6.1, 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 [12]:

$$\exists z \in B_k(x) \wedge \exists y \in S_{k-1}(x) : (y,z) \in E \tag{5}$$

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. i.e., $S_k(x) = B_1(x) \cup B_2(x) \cup \ldots \cup B_k(x)$.

Based on theorem 6.1, it is clear that $B_k(x)$ will remain the same after an action of *swap* for $1 \leq k \leq L$. 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, equation (5) 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, it is concluded that:

Corollary 6.4. *The closeness property and the connectivity property will keep inviolated after a sub-tree spinning.*

Theorem 6.5. *sub-tree 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 6.2 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, *sub-tree spinning* does not change the gateway's corresponding nexthops.

Now suppose when the gateway is G_m $(0 \leq m \leq k, k \geq 0)$, *sub-tree spinning* does not change the gateway's corresponding nexthops. We shall prove that the corresponding nexthops of G_{k+1} will still not be changed by *sub-tree 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). Recall 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, *sub-tree spinning* does not change the corresponding nexthops of G_m $(0 \leq m \leq k, k \geq 0)$ 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 to reach G_{k+1}. Therefore the theorem is proved. □

Theorem 6.5 gives us the mathematical confidence when updating *sid*s, that we only need to re-calculate prefixes of each level, without concerning about the nexthops since they are proven to be invariable physically. That is to say, no matter how sub-trees are spun, the output port of each level in the Routing table should remain the same. Or rather, *sub-tree spinning* does not change the path that packets travel the network, just the identifiers.

7 RELATED WORKS

To the best of our knowledge, the most similar existing work to U-TRI is PHEAR, which is a system built atop existing SDN protocols and standards. PHEAR aims 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 identification free, randomly generated and periodically changing nonce to the packet header, normalizing other identifiers found at layer 2 and 3, and encrypting payload from layer 3 above using existing security protocols. PHEAR is a novel solution implemented in an SDN environment. However, the so-called 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. while one nonce can only represent one pair of end-hosts, central network devices may have to store as many as n^2 nonces 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 down-grade security levels under a situation when performance is needed more than security. The hiding scheme of PHEAR by encrypting all upper layer payload would make other SDN application running in the same network hardly work since flow rule matching cannot work on encrypted packets.

Tor [10] is the most well-known anonymizing network on the Internet. Vincent Liu *et al.* [15] 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 required to travel. A similar system can be implemented in a local network but has two major shortcomings. Firstly, the Tor protocol requires being adapted to 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 sole function network device to increase the network administration burden. And encryption-based solution such as Tor protocol apparently is hard to cooperate with other network services without sharing secrets, and sharing secrets with other system is clearly not allowed by Tor.

RHM [11] 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 traversing the protected network, and restore the address before the packet arriving its destination host. RHM depends on a large unused IP address space to provide the ephemeral IP address, which is not always easily satisfiable. In contrast, U-TRI doesn't 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 real address and ephemeral address mapping. 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 does prevent the raw packets containing the destination address from presenting in the edge device, thus provides stronger protection on unlinkability.

Qi Duan *et al.* [6] proposed a random route mutation(RRM) technique to defend against eavesdropping. Changing the path of packets traversing indeed disrupts the linkability analysis on packets conducted by adversaries on routers. However, its effectiveness depends on the number of different routes between end-hosts. The wide area network may be able to provide such a sufficiently large number of different routes, but this is clearly not the case in a local network, which usually utilizes star-like topology and lack of redundant traversing paths.

Sridhar Venkatesan *et al.* [28] proposed an MTD approach utilizing proxy. In their approach, the proxy is not a software on end-hosts, but a standalone device in the network and 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 look up requesting tasks is the counterpart of local proxy in U-TRI. The effect of MTD requires a way of distinguishing normal users and adversaries. And local proxy is one of such ways.

The Host Identity Protocol (HIP) [18] is an approach that separates the identity of a host from its location to provide secure end-to-end mobility and multihoming. In a HIP architecture, the location of the host is bound to IP addresses and used for routing packets to the host in the same way as in the current Internet architecture. But the identity of the host is defined by the *host identity* which is a long-term identity and is used by transport and application layers to preserve connections when the host is moving from one network to another. Although HIP utilized IPsec ESP *Bound End-to-End Mode* (BEET) [13, 19] to provide data encryption and integrity protection, it cannot provide *unlinkability* protection since the *host identity* is a long-term identity which can easily be used to gain linkage information between communication peers by the adversaries. U-TRI also has a host identity *hid*. But the *hid* is a short-term identity that is randomly and periodically changed, which is helpful to hide the linkage information.

8 CONCLUSION

Moving target approach is hopeful for re-balancing the cyber landscape in favor of defense. By changing the network attack surface randomly while sustaining network services for normal users, we can drastically increase attacking costs of adversaries. In this paper, we proposed a random namespace scheme that is able to hide majority identifiers in packets. We chose a hierarchical virtual namespace to ensure packet switching efficiency and add randomization to provide attack surface moving capability. We also utilized the centralized software-based network control ability of SDN to implement the scheme. Our scheme can provide both unlinkability and efficient network service. Our next step work would be refining the moving strategy, taking network traffic trending into consideration, so as to remove unnecessary moving actions. We will also try to find a new method to increase the size of the anonymous set of vids under certain circumstances in order to improve the overall protection capability of our scheme.

ACKNOWLEDGMENT

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] S. Ansari, S. G. Rajeev, and H. S. Chandrashekar. 2002. Packet sniffing: a brief introduction. *IEEE Potentials* 21, 5 (Dec 2002), 17–19. DOI:https://doi.org/10.1109/MP.2002.1166620

[2] A. V. Arzhakov and I. F. Babalova. 2017. Analysis of current internet wide scan effectiveness. In *Proceedings of 2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus).* 96–99. DOI:https://doi.org/10.1109/EIConRus.2017.7910503

[3] Po-Wen Chi, Chien-Ting Kuo, Jing-Wei Guo, and Chin-Laung Lei. 2015. How to detect a compromised SDN switch. In *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft).* 1–6. DOI:https://doi.org/10.1109/NETSOFT.2015.7116184

[4] Y. C. Chiu and P. C. Lin. 2017. Rapid detection of disobedient forwarding on compromised OpenFlow switches. In *Proceedings of 2017 International Conference on Computing, Networking and Communications (ICNC).* 672–677. DOI:https://doi.org/10.1109/ICCNC.2017.7876210

[5] I. Csiszar. 1994. Maximum entropy and related methods. In *Proceedings of 1994 Workshop on Information Theory and Statistics.* 11–. DOI:https://doi.org/10.1109/WITS.1994.513853

[6] Qi Duan, E. Al-Shaer, and H. Jafarian. 2013. Efficient Random Route Mutation considering flow and network constraints. In *Proceedings of 2013 IEEE Conference on Communications and Network Security (CNS).* 260–268. DOI:https://doi.org/10.1109/CNS.2013.6682715

[7] Nick Feamster, Jennifer Rexford, and Ellen Zegura. 2014. The Road to SDN: An Intellectual History of Programmable Networks. *SIGCOMM Comput. Commun. Rev.* 44, 2 (April 2014), 87–98. DOI:https://doi.org/10.1145/2602204.2602219

[8] Yossi Gilad and Amir Herzberg. 2012. Spying in the dark: TCP and tor traffic analysis. In *Proceedings of the 12th International Symposium on Privacy Enhancing Technologies Symposium.* Springer, 100–119.

[9] Mordechai Guri. 2016. Deception and Counter Deception: Moving Target Attacks vs. Moving Target Defense. (2016). http://engage.morphisec.com/e-book-counter-deception-moving-target-defense

[10] R. A. Haraty and B. Zantout. 2014. The TOR data communication system. *Journal of Communications and Networks* 16, 4 (Aug 2014), 415–420. DOI:https://doi.org/10.1109/JCN.2014.000071

[11] J. H. Jafarian, E. Al-Shaer, and Q. Duan. 2015. An Effective Address Mutation Approach for Disrupting Reconnaissance Attacks. *IEEE Transactions on Information Forensics and Security* 10, 12 (Dec 2015), 2562–2577. DOI:https://doi.org/10.1109/TIFS.2015.2467358

[12] S. Jain, Y. Chen, Z. L. Zhang, and S. Jain. 2011. VIRO: A scalable, robust and namespace independent virtual Id routing for future networks. In *Proceedings of 2011 IEEE INFOCOM.* 2381–2389. DOI:https://doi.org/10.1109/INFCOM.2011.5935058

[13] Petri Jokela, Jan Melen, and Robert Moskowitz. 2008. Using the encapsulating security payload (ESP) transport format with the host identity protocol (HIP) (RFC 5202). (2008). https://datatracker.ietf.org/doc/rfc5202/

[14] S. Kent and K. Seo. 2005. Security Architecture for the Internet Protocol (RFC 4301). https://tools.ietf.org/html/rfc4301. (December 2005).

[15] Vincent Liu, Seungyeop Han, Arvind Krishnamurthy, and Thomas Anderson. 2011. Tor Instead of IP. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks (HotNets-X).* ACM, New York, NY, USA, Article 14, 6 pages. DOI:https://doi.org/10.1145/2070562.2070576

[16] D.J.C. MacKay. 2003. *Information Theory, Inference and Learning Algorithms.* Cambridge University Press. https://books.google.com.hk/books?id=AKuMj4PN_EMC

[17] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.* 38, 2 (March 2008), 69–74. DOI:https://doi.org/10.1145/1355734.1355746

[18] Robert Moskowitz, Pekka Nikander, Petri Jokela, and Thomas Henderson. 2008. Host Identity Protocol (RFC 5201). (2008). https://datatracker.ietf.org/doc/rfc5201/

[19] PEKKA Nikander and J Melen. 2007. A Bound End-to-End Tunnel (BEET) mode for ESP (IETF draft). (2007). https://tools.ietf.org/html/draft-nikander-esp-beet-mode-09

[20] F. Oggier and B. Hassibi. 2011. The Secrecy Capacity of the MIMO Wiretap Channel. *IEEE Transactions on Information Theory* 57, 8 (Aug 2011), 4961–4972. DOI:https://doi.org/10.1109/TIT.2011.2158487

[21] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, and others. 2015. The Design and Implementation of Open vSwitch. In *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation.* 117–130.

[22] Andreas Pfitzmann and Marit Hansen. 2010. A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management. (April 2010). http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.33.pdf v0.33.

[23] Yuji Agawa Rui Kubo, Tomonori Fujita and Hikaru Suzuki. 2014. Ryu SDN framework-open-source SDN platform software. *NTT Technical Review* 12, 8 (Aug 2014), 1–5.

[24] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao. 2013. Are we ready for SDN? Implementation challenges for software-defined networks. *IEEE Communications Magazine* 51, 7 (July 2013), 36–43. DOI: https://doi.org/10.1109/MCOM.2013.6553676

[25] Richard Skowyra, Kevin Bauer, Veer Dedhia, and Hamed Okhravi. 2016. Have No PHEAR: Networks Without Identifiers. In *Proceedings of the 2016 ACM Workshop on Moving Target Defense (MTD ’16)*. ACM, New York, NY, USA, 3–14. DOI: https://doi.org/10.1145/2995272.2995276

[26] The Open Networking Foundation. 2011. OpenFlow Switch Specification. (Feb. 2011). https://www.opennetworking.org/software-defined-standards/specifications

[27] Baylor University. 2017. Sniffing (network wiretap, sniffer) FAQ. (2017). http://cs.baylor.edu/~donahoo/tools/sniffer/sniffingFAQ.htm

[28] S. Venkatesan, M. Albanese, K. Amin, S. Jajodia, and M. Wright. 2016. A moving target defense approach to mitigate DDoS attacks against proxy-based architectures. In *Proceedings of 2016 IEEE Conference on Communications and Network Security (CNS)*. 198–206. DOI: https://doi.org/10.1109/CNS.2016.7860486