A Cost-Effective Shuffling-Based Defense against HTTP DDoS Attacks with SDN/NFV

Yi-Hui Lin*†, Jian-Jhih Kuo*, De-Nian Yang*, and Wen-Tsuen Chen*

*Institute of Information Science, Academia Sinica, Taipei, Taiwan 115
†Information and Communications Research Division, NCSIST, Taoyuan City, Taiwan 325
Email:{yihui1223,lajacky,dnyang,chenwt}@iis.sinica.edu.tw

***Abstract*—Software-Defined Networking and Network Function Virtualisation (SDN/NFV) can provide flexible resource allocation to support innovative security solutions in a central manner. To mitigate HTTP DDoS attacks, shuffling-based moving target defense has been regarded as one of the most effective ways by redirecting user traffic among a group of virtualized service functions. However, previous work did not notice that frequent changes of user traffic will significantly intensify the control overhead of SDN. In this paper, therefore, we first model the effectiveness and cost for shuffling in SDN/NFV networking with Multi-Objective Markov Decision Processes to find the optimal tradeoff between the effectiveness and cost. We then propose a cost-effective approximation algorithm with a guarantee performance bound to solve the problem. Simulation and implementation on an experimental SDN/NFV network manifest that, given 100 attackers among 1000 users and 50 virtualized functions of a web service, our algorithm achieves the approximation ratio of 0.68 and imposes only 2.4s rule modification latency for each shuffle.**

I. Introduction

HTTP(s) DDoS attacks have been regarded as one of the most serious attacks on web services, whereas 43% of the three biggest cyber attacks are HTTP(s) attacks, and 27% of organizations face daily or weekly HTTP(s) attacks in 2015 [23]. Various tactics have been facilitated to realize HTTP(s) attacks for consuming and occupying the computational resources of the web servers. Compared with other kinds of DDoS attacks, defending HTTP DDoS attacks are more challenging due to the following reasons. First, the attackers must generate the traffic with non-spoofed IP addresses, and it is thereby harder to filter the traffic based on the spoofed source IP addresses during the attack. Second, HTTP attacks require a much smaller attack size to exhaust the computational resources of the web services, especially when the web services are very computation-intensive nowadays. For example, a small VPN on a cloud can properly handle the SYNC flood with the size as 200,000 packets per second but is usually unable to withstand under the attack of 500 HTTP requests per second [24].

For traditional networks, the previous solutions to HTTP DDoS attacks either operate in the sole server side [9][16] or require the cooperation with the clients [8][13]. The existing schemes [9][16] detect the attackers by building legitimate client behavior profiles based on statistics model and Hidden Semi-Markov Model, respectively, whereas [8] and [13] require the legitimate clients to identify themselves by passing CAPTCHA and raising the traffic volume, respectively. The former works require extra communication overheads to collect the data if the service is deployed in multiple servers, while the latter is inclined to bother the users.

Recently, network function virtualisation (NFV) [19] has been widely applied for many application paradigms due to its excellent elasticity and efficiency for allocating the computational resource, and several approaches [5][6][12][14] have been proposed to mitigate the DDoS attacks for NFV. Jia et al. [5][6] proposed shuffling-based moving-target defense to generate multiple replicas/proxies[1] on the cloud for guiding the users to the designated replicas through DNS routing. The users connecting to crashed replicas are regrouped and then redirected to the new proxies by repetitive shuffling. Afterward, the attackers pretending as ordinary users are gradually isolated to a fewer replicas, and the number of replicas can be effectively reduced to lower the shuffling cost [14]. However, the above shuffling approaches are inclined to suffer from the proxy harvesting attack [12] because DNS routing usually reveals the proxy addresses to the users, and the attackers in this situation can probe the location and the number of proxies in advance. To address the above issue, a bind/split strategy [12] is designed to limit the number of probing from attackers.

Although the above shuffling-based approaches have been demonstrated very effective to mitigate HTTP attacks, additional admission control usually needs to be involved in traditional networks to ensure that user traffic indeed follows the redirection instruction, because the user traffic is configured by the sources (i.e., users). In this case, the admission control server is inclined to become another target of DDoS attacks [6]. Multiple points of admission control also increase the network operation cost in Content-Centric Networking (CDN) [14]. However, shuffling has not been explored for Software-Defined Networking (SDN) in the literature.

For SDN [4], in contrast, the reconfiguration of routing paths can be achieved on the fly by the SDN controller.[2] Users in this cases are transparent to shuffling, and additional admission control thereby is not required. More specifically,

[1]Replicas and proxies are interchangeable in this paper.

[2]Most of recent SDN-based DDoS defenses tend to handle network-layer attacks [7][10].

when the user traffic is reconfigured to another replica, the SDN controller only needs to install the new forwarding rule in the new path to the replica, and the proxy harvest attack in [12] will not occur here. However, additional route reconfiguration cost is involved for shuffling in SDN because the SDN controller needs to find the new routing path and install the new forwarding rule to every switch in the new path. Additional network latency is usually imposed because the CPU of the SDN switches restricts the bandwidth of the control channel to only 10 Mbit/sec [1]. Therefore, it is very important to consider the overhead of shuffling in SDN, especially when the number of users boosts.[3]

In this paper, therefore, we propose a new shuffling-based defense system, named Software-Defined Shuffling System (SDSS), for SDN/NFV networking. To explore the tradeoff between the effectiveness of shuffling and the SDN reconfiguration cost, we investigate a new Cost-effective Shuffling Problem (CSP) and propose two algorithms, Baseline Shuffling Algorithm (BSA) and Cost-effective Shuffling Algorithm (CSA). The goal of CSP is to find the optimal strategy for a sequence of shuffling decisions under the assumption that the attackers and the legal users are indistinguishable. To solve the problem, BSA is first proposed to maximize the effectiveness but ignore the reconfiguration cost in SDN. In contrast, CSA is an approximation algorithm to find the best trade-off between the effectiveness and reconfiguration in SDN. Simulation results manifest that CSA can effectively shuffle and concentrate the attackers in fewer replicas with limited reconfiguration cost in SDN.

The remainder of this paper is organized as follows. Section II first presents the threat and defense models. Then, we formalize CSP in SDN in Section III. Section IV models the problem with Multi-Objective Markov Decision Processes, and two new shuffling algorithms, BSA and CSA, are proposed in Section V and Section VI, respectively. We derive the approximation ratio and evaluate the proposed algorithms via simulation in Section VII and Section VIII, respectively. Section IX presents the prototype implementation in SDN/NFV. Finally, we conclude the paper in Section X.

II. Attack Model and Defense Model

In this section, we first describe the behavior of HTTP DDoS attacks and then present the defense model in SDN/NFV.

A. Attack Model

The DDoS attack model [17][18] is defined as follows. The attackers consist of a small group of m compromised hosts among a large group of n users, and the HTTP traffic originating from the attackers is indistinguishable from that generated by legitimate users. HTTP DDoS attacks last a period of time for exhausting the server resources, such as CPU, memory, and connection capacity by the following attack types.

- *Session Flooding Attacks*: The attackers send a large number of HTTP requests through overflowing sessions.
- *Request Flooding Attacks*: The attackers send a large number of HTTP requests with only a few sessions.
- *Asymmetric Attacks*: To reduce the packet rate and maintain the high load in the server, the attackers send mostly HTTP requests of heavy workload.
- *Slow Request/Response Attacks*: To hold and prolong HTTP sessions, the attackers slowly sends the HTTP requests/responses by delivering incomplete HTTP headers, diminishing HTTP data rate, or fragmenting HTTP packets.

B. Shuffling-Based Defense Model

When a service is under HTTP attacks, the SDN/NFV network operator creates p VMs as replicas/proxies of the service, and each VM is allocated with sufficient resources for q legitimate users, where $p \times q \geq n$. Then, the SDN controller evenly assigns n users to p VMs by installing the corresponding forward rules in the routing paths.

After monitoring the resource consumptions of VMs for a short period of time, the controller first identifies the set of VMs under attacks. Then, it restarts the crashed VMs and reassigns each user of a crashed VMs to another restarted VMs by reconfiguring the routing. The system repeats the above shuffling process to concentrate the attackers in fewer VMs. The attackers will be gradually located in similar VMs because the corrupted VMs may become sanitized by having the attackers moved out and the legal users moved in, causing more attackers to be distributed in the remain corrupted VMs. Nevertheless, additional computation and signaling overheads are incurred in SDN for finding the new routing paths and reconfiguring the switches. Therefore, our goal is to balance the defense effectiveness and the SDN overhead by maximizing the reduced number of crashed VMs and minimizing the number of user reassignment.

III. Problem Formulation

Specifically, the problem formulation is defined as follows.

Given: a set of n users with m attackers and a group of p VMs of equal resources for q users, where $p \times q = n$

Output: a sequence of matrices $(A_0, A_1, A_2, ..., A_T)$, where $A_t \in \{0,1\}^{p \times n}$, such that

$$\sum_{j=1}^{n} a_{ij}^t = q \quad i = 1, ..., p; \tag{1a}$$

$$\sum_{i=1}^{p} a_{ij}^t = 1 \quad j = 1, ..., n; \tag{1b}$$

The matrix A_0 denotes the initial assignment, and the matrices $\{A_t | t > 0\}$ represent the shuffling decision at time t, where binary variable a_{ij}^t indicates that whether the j-th user is assigned to i-th VM. Hence, Equation 1a states that each VM

[3]The overhead of path reconfiguration is not considered in [11] and the work of [2] focuses on elastic scaling of defense with SDN/NFV rather than a specific attack.

is only allowed to serve q users, and Equation 1b ensures that each user is assigned to only one VM.

Let binary vector $\Omega = (\omega_1, \omega_2, ..., \omega_n)$ indicates whether each user is an attacker, where $\sum_{j=1}^{n} \omega_j = m$. Let ζ_t denote the number of the crashed VMs, and b_i^t is a boolean value that represents whether i-th VM is crashed in t-th time slot (Equation 2b). Equation 2c states that a VM will not crash if no attacker is assigned to the VM. Both ζ_t and b_i^t are available only after A_t is acquired. Since the goal of shuffling is to concentrate the attackers in fewer VMs, we quantify the effectiveness in t-th shuffle by the reduced number of crashed VMs multiplied by a discount value γ^t, where the discounted value ensures that the effectiveness is decayed over time (Equation 2a). The objective function to maximize the effectiveness of shuffling is as follows:

$$f_E = \text{maximize } \sum_{t=1}^{T} \gamma^t(\zeta_{t-1} - \zeta_t), \text{ where} \tag{2a}$$

$$\zeta_t = \sum_{i=1}^{p} b_i^t \text{ and} \tag{2b}$$

$$b_i^t = \begin{cases} 0 & \text{if } \sum_{j=1}^{n} a_{ij}^t \wedge \omega_j = 0 \\ 1 & \text{otherwise} \end{cases} \tag{2c}$$

Next, we define the SDN control overhead of T times shuffling. Let η_t denote the number of moving users, and c_j^t is a binary variable indicating whether j-th user is moving to another VM in t-th shuffle (Equations 3b and 3c). If j-th user remains in the same VM in t-th shuffle, c_j^t is set to 0; otherwise, c_j^t is set to 1. The objective function to minimize the cost of T times shuffling is as follows:

$$f_C = \text{minimize } \sum_{t=1}^{T} \eta_t, \text{ where} \tag{3a}$$

$$\eta_t = \sum_{j=1}^{n} c_j^t \text{ and} \tag{3b}$$

$$c_j^t = \begin{cases} 0 & \text{if } \sum_{i=1}^{p} a_{ij}^t \oplus a_{ij}^{t-1} = 0 \\ 1 & \text{otherwise} \end{cases} \tag{3c}$$

In our problem, we consider both the effectiveness and cost of shuffling with different weights w_1 and w_2 assigned by the network operator. Our objective function finds the best tradeoff as follows, where the variable of $p - \zeta_0$ represents the number of the survived VMs after the initial assignment.

Objective:

$$\text{maximize } w_1 \times q \times (p - \zeta_0 + f_E) - w_2 \times f_C \tag{4}$$

In the next section, we first tackle our problem as a multi-objective optimization problem.

IV. Problem Modeling

In the following, we first exploit Multi-Objective Markov Decision Processes (MOMDP) [3][15] to model our problem, which is a multi-stage stochastic optimization problem with multiple objectives. A MOMDP for two objectives in our case is a tuple $(\mathcal{S}, \mathcal{X}, \pi, \mathcal{P}^\pi, \mathcal{R})$, where:

- $\mathcal{S}$ represents a finite set of states, and let S_t be the random variable of the state at time t.
- $\mathcal{X}$ denotes a finite set of decisions, and let X_t be the random variable of the decision at time t.
- $\pi : \mathcal{S} \to \mathcal{X}$ is a policy function that maps each state to a decision with the probability $\mathbb{P}[X_t = x | S_t = s]$, where $x \in \mathcal{X}$ and $s \in \mathcal{S}$.
- $\mathcal{P}^\pi$ is a state transition probability matrix with the policy π, where

  $$p_{ij}^\pi = \sum_{x \in \mathcal{X}} \mathbb{P}[X_t = x | S_t = s_i] \mathbb{P}[S_{t+1} = s_j | S_t = s_i, X_t = x].$$

- $\mathcal{R}^\pi : \mathcal{S} \times \mathcal{X}^\pi \to \mathbb{R}^\ell$ is a rewarding function that maps a state and a decision to a rewarding vector of ℓ dimensions, where the decision distribution is π.

A policy is an algorithm or a strategy that decides the probability distribution of the solutions based on the number of the crashed VMs at time t. To find the optimal policy, we solve the following value function iteratively (i.e., value iteration), where w_k is the weighted value of k-th objective that satisfies $\sum_{k=1}^{\ell} w_k = 1$, and $\gamma_k \in [0, 1]$ is the discount value of k-th objective.

$$V_{t+1}^\pi(s) = \max \sum_{k=1}^{\ell} w_k V_{t+1,k}^\pi(s), \text{where} \tag{5a}$$

$$V_{t+1,k}^\pi(s) = R_k^\pi(s, x) + \gamma_k \mathbb{E}[V_{t,k}^\pi(s') | s, x] \tag{5b}$$

By following the policy π after t stages, the above value function of Equation 5b represents the expected cumulative reward for objective k, and Equation 5a sums up the values of all objectives with their weights. For our problem, the value of each state in $\mathcal{S}$ represents the number of crashed VMs, where $\mathcal{S} = \{min(p, m), min(p, m) - 1, min(p, m) - 2, ..., \lceil \frac{m}{q} \rceil\}$. Each decision in $\mathcal{X}$ is a (0,1)-matrix with the constraints the same as Equation 1. After finding the optimal policy, the transition probability matrix with the policy $\mathcal{P}^\pi$ can be induced. Then, the rewarding values of two objectives can be calculated based on Equation 2 and Equation 3.

However, the computation complexity to find the optimal policy is $O(T \times m^2 \times n!)$, where T is the total number of shuffling, m is the number of attackers, and n is the number of users. It is envisaged that directly solving our problem is computation intensive. In the following sections, therefore, we first present a baseline algorithm and then design an efficient approximation algorithm to solve the problem.

V. The Baseline Shuffling Algorithm

In the following, we first present a baseline shuffling algorithm to consider the cost of rerouting in SDN. BSA consists of two steps. The initial assignment step derives the initial state, whereas the t-th shuffling step iteratively reduces the number of the crashed VMs. The routing cost of each shuffle is the sum

of the users that need to change the serving VMs. Specifically, in the initial assignment step (Algorithm 1), n users (including m attackers) are randomly assigned to p VMs through random permutation function ρ. The probability distribution of the initial state is as follows:

$$Pr[S_1 = s] = \frac{(n-m)!}{n!} \times \mathbb{C}_s^p \sum_{k=0}^{s-\lceil \frac{m}{q} \rceil} (-1)^k \mathbb{C}_{s-k}^s \mathbb{P}_m^{q(s-k)}$$

Based on uniform permutation of the users in Algorithm 1, the permutation of m attackers that occupy s VMs is $\mathbb{C}_s^p \sum_{k=0}^{s-\lceil \frac{m}{q} \rceil} (-1)^k \mathbb{C}_{s-k}^s \mathbb{P}_m^{q(s-k)}$ while the permutation of the rest users is $(n-m)!$ among total $n!$ decisions.

Algorithm 1 Initial Assignment

Input: a list of users $\mathcal{U} = \{u_1, u_2, ..., u_n\}$ and a list of VMs $\mathcal{M} = \{m_1, m_2, ..., m_p\}$ with size of q users
Output: A binary $p \times n$-matirx A_0

```
Compute ρ(U) = {u_ρ(1), u_ρ(2), ..., u_ρ(n)} with the random permutation function ρ;
for each user u_j do (the user index j is according to ρ)
    k := ⌈ρ(j)/q⌉;
    for each VM m_i do
        if i = k then a^0_ij := 1
        else a^0_ij := 0
        end if
    end for
end for
return all a^0_ij ∈ A_0
```

Afterward, the shuffling step (Algorithm 2) at time t randomly groups and reassigns the users in the crashed VMs through the random permutation function, while the assignment of the other users remains identical. The transition probability matrix in the step is as follows:

$$p_{ij}^{BSA} = \begin{cases} 0 & \text{if } s_i < s_j \\ \frac{(qs_i - m)!}{qs_i!} \times \mathbb{C}_{s_j}^{s_i} \sum_{k=0}^{s_j - \lceil \frac{m}{q} \rceil} (-1)^k \mathbb{C}_{s_j-k}^{s_j} \mathbb{P}_m^{q(s_j-k)} & \text{else} \end{cases}$$

The rewarding values R_1^{BSA} and R_2^{BSA} of each state s_i with policy BSA represents the effectiveness and cost of a shuffle as follows.

$$R_1^{BSA}(s_i) = \sum_{j=1}^{|\mathcal{S}|} p_{ij}^{BSA}(s_i - s_j) \tag{6a}$$

$$R_2^{BSA}(s_i) = n - qs_i(1 - 1/s_i) \tag{6b}$$

Regarding shuffling effectiveness, the rewarding function in Equation 6a includes the expected reduced number of VMs in a shuffle in state s_i. Hence, s_i and s_j are equivalent to ζ_t and ζ_{t+1} in Equation 2b if $S_t = s_i$ and $S_{t+1} = s_j$. In terms of traffic redirection cost in SDN, the rewarding function in Equation 6b represents the number of stationary users, where $qs_i(1 - 1/s_i)$ is the number of the moving users in state s_i.

In BSA, all users in the crashed VMs are redirected with the probability $1 - 1/s_i$, and $R_2^{BSA}(s_i)$ is equivalent to the value of $n - \eta_t$ in Equation 3b if $S_t = s_i$. The value functions of each state with the policy BSA is as follows:

$$V_{t+1,k}^{BSA}(s_i) = R_k^{BSA}(s_i) + \gamma_k \sum_{j=1}^{|\mathcal{S}|} p_{ij}^{BSA} V_{t,k}^{BSA}(s_j), \tag{7}$$

where $V_{1,k}^{BSA} = R_k^{BSA}$, γ_k is a discount value with $\gamma_1 := \gamma$ defined in Equation 2a and $\gamma_2 := 1$.

Algorithm 2 Baseline Shuffling Algorithm (BSA): t-th Shuffling

Input: A binary $p \times n$-matrix A_{t-1}, a list of users $\mathcal{U}$, a list of crashed VMs $\mathcal{M}_t = \{m_{x_1}, m_{x_2}, ..., m_{x_r}\}$, and a list of users in crashed VMs $\mathcal{U}_t = \{u_{y_1}, u_{y_2}, ..., u_{y_{qr}}\}$
Output: A binary $p \times n$-matrix A_t

```
Compute ρ(U_t) = {u_yρ(1), u_yρ(2), ..., u_yρ(qr)} with the random permutation function ρ;
for each u_yj do (the index j is according to ρ)
    k := ρ(j)/q;
    for each crashed VM m_xi do
        if i = k then a^t_xiyj := 1
        else a^t_xiyj := 0
        end if
    end for
    for each VM m_i ∈ M − M_t do
        a^t_iyj := 0
    end for
end for
for each u_j ∈ U − U_t do
    a^t_ij := a^{t-1}_ij ∀i = 1 to p
end for
return all a^t_ij ∈ A_t
```

VI. The Cost-Effective Shuffling Algorithm

BSA relocates almost every user in the crashed VMs and thereby tends to incur redundant movement, especially when the attackers are more sparse in the crashed VMs at the beginning of shuffling. In this section, therefore, we propose a cost-effective shuffling algorithm (CSA) to significantly reduce the number of rule installations in SDN. CSA (Algorithm 3) is executed in each time t after the initial assignment in Algorithm 1. Different from BSA, here the VMs are first randomly paired. Afterward, for each pair of VMs, one VM randomly selects half of the users and moves them to the other VM, whereas the other half stays in the same VMs. The goal of CSP is to generate more distinct groups than the current ones from a random pair. We relocate only half users because moving more/less than half users leads to similar groups and tends to reduce the effectiveness. The transition probability of CSA is represented as the function δ_{ij}, where the value is

correlated to (m, n, p, q).

$$p_{ij}^{CSA} = \begin{cases} \delta_{ij}(n, m, p, q) & \text{if } i + \lfloor s_i/2 \rfloor \geq j \geq i \\ 0 & \text{Else} \end{cases}$$

The rewarding values R_1^{CSA} and R_2^{CSA} of each state s_i with policy CSA represent the effectiveness and cost of each shuffle in state s_i as follows.

$$R_1^{CSA}(s_i) = \sum_{j=1}^{|\mathcal{S}|} p_{ij}^{CSA}(s_i - s_j) \tag{8a}$$

$$R_2^{CSA}(s_i) = n - \frac{1}{2} q s_i \tag{8b}$$

Equation 8b indicates that the moving probability of each user in the crashed VMs is $1/2$. Given the transition probability and the rewarding function, the cumulation of effectiveness and cost with the policy CSA after $t+1$ shuffles is as follows:

$$V_{t+1,k}^{CSA}(s_i) = R_k^{CSA}(s_i) + \gamma_k \sum_{j=1}^{|\mathcal{S}|} p_{ij}^{CSA} V_{t,k}^{CSA}(s_j), \tag{9}$$

where $V_{1,k}^{CSA} = R_k^{CSA}$, γ_k is a discount value with $\gamma_1 = \gamma$ defined in Equation 2a and $\gamma_2 = 1$.

Algorithm 3 Cost-Effective Shuffling Algorithm (CSA): t-th Shuffling

Input: A binary $p \times n$-matrix A_{t-1}, a list of users $\mathcal{U}$ and a list of crashed VMs $\mathcal{M}_t = \{m_{x_1}, m_{x_2}, ..., m_{x_r}\}$
Output: A binary $p \times n$-matrix A_t

1: **for** each pair of $(m_{x_{\rho(i-1)}}, m_{x_{\rho(i)}})$, where $\rho(i) \bmod 2 = 0$, where users $\mathcal{U}_t^0 = \{u_{y_1}, u_{y_2}, ..., u_{y_q}\}$ in $m_{x_{\rho(i-1)}}$ and $\mathcal{U}_t^1 = \{u_{z_1}, u_{z_2}, ..., u_{z_q}\}$ in $m_{x_{\rho(i)}}$ **do** //random pairing
2: Randomly choose $\{u_{y'_1}, u_{y'_2}, ..., u_{y'_{\lceil q/2 \rceil}}\} \subset \mathcal{U}_t^0$ and $\{u_{z'_1}, u_{z'_2}, ..., u_{z'_{\lceil q/2 \rceil}}\} \subset \mathcal{U}_t^1$
3: Set $a_{x_{\rho(i-1)}, z'_j}^t := 1$ and $a_{x_{\rho(i)}, y'_j}^t := 1$
4: Set $a_{i, z'_j}^t := 0$ for all $i \neq x_{\rho(i-1)}$
5: Set $a_{i, y'_j}^t := 0$ for all $i \neq x_{\rho(i)}$
6: **end for**
7: **for** each $u_j \in \mathcal{U} - (\forall\{u_{y'_1}, u_{y'_2}, ..., u_{y'_{\lceil q/2 \rceil}}\} + \forall\{u_{z'_1}, u_{z'_2}, ..., u_{z'_{\lceil q/2 \rceil}}\})$ **do**
8: $a_{ij}^t := a_{ij}^{t-1}$
9: **end for**
10: **return** all $a_{ij}^t \in A_t$

VII. Analysis

In the following, we derive the approximation ratio of the proposed algorithm. Specifically, we first evaluate the speed of the released users. Let $V_{t,1}^{\pi}$ denote the speed of the diminishing crashed VMs after t shuffles with the solution π. Hence, the expected speed of the released users, denoted by $\mathbb{E}[\bar{V}_{t,1}^{\pi}]$ can be derived as follows,

$$\mathbb{E}[\bar{V}_{t,1}^{\pi}] = q \sum_{i=1}^{|\mathcal{S}|} Pr[S_1 = s_i](V_{t,1}^{\pi}(s_i) + (p - s_i))$$

where $(p - s_i)$ is the diminished number of the initial assignment. $\bar{V}_{effect}$ denotes the upper bound of the effectiveness with only one shuffle after the initial assignment.

$$\bar{V}_{effect} = q \sum_{i=1}^{|\mathcal{S}|} Pr[S_1 = s_i](\gamma_1(s_i - s_{|\mathcal{S}|}) + (p - s_i))$$

To analyze the cost, we derive the expected number of stationary users after t shuffles, denoted $\mathbb{E}[\bar{V}_{t,2}^{\pi}]$ as follows.

$$\mathbb{E}[\bar{V}_{t,2}^{\pi}] = \sum_{i=1}^{|\mathcal{S}|} Pr[S_1 = s_i](V_{t,2}^{\pi}(s_i))$$

$\bar{V}_{cost}$ denotes the upper bound of total stationary users with only the attackers and m legal users moved in the first shuffle.

$$\bar{V}_{cost} = n - 2\gamma_2 m$$

Based on the above evaluation, we derive the approximation ratio of the proposed algorithm as follows.

Theorem 1: Given the weighted values of the objectives (w_1, w_2) and t-time shuffling operations, the solution π guarantees that $\mathbb{E}[\bar{V}_t^{\pi}] \geq \alpha_{\pi} \mathbb{E}[\bar{V}_t^{opt}]$, where $\alpha_{\pi} = \frac{w_1 \mathbb{E}[\bar{V}_{t,1}^{\pi}] + w_2 \mathbb{E}[\bar{V}_{t,2}^{\pi}]}{w_1 \cdot \bar{V}_{effect} + w_2 \cdot \bar{V}_{cost}}$ and opt is the optimal solution.

Proof: Since

$$\mathbb{E}[\bar{V}_t^{\pi}] = w_1 \mathbb{E}[\bar{V}_{t,1}^{\pi}] + w_2 \mathbb{E}[\bar{V}_{t,2}^{\pi}]$$

and

$$\mathbb{E}[\bar{V}_t^{opt}] \leq w_1 \cdot \bar{V}_{effect} + w_2 \cdot \bar{V}_{cost},$$

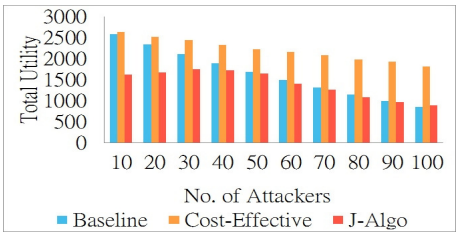
$$\mathbb{E}[\bar{V}_t^{\pi}] \geq \frac{w_1 \mathbb{E}[\bar{V}_{t,1}^{\pi}] + w_2 \mathbb{E}[\bar{V}_{t,2}^{\pi}]}{w_1 \cdot \bar{V}_{effect} + w_2 \cdot \bar{V}_{cost}} \mathbb{E}[\bar{V}_t^{opt}] \text{ holds.}$$

■

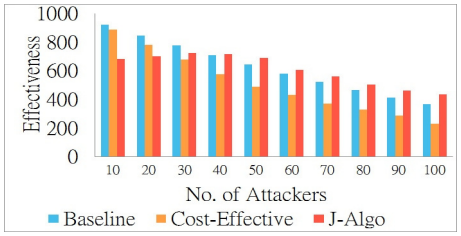
Theorem 1 finds the approximation ratio of the speed for the shuffling solutions. Later in the next section, we show that α_{BSA} and α_{CSA} are both close to 1 for different numbers of the attackers and VMs.

VIII. Simulation

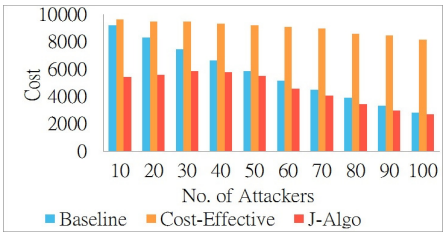
In the following, we compare the proposed algorithms with J-Algo [5] to evaluate the effectiveness and cost of shuffling. To find the transition probability matrix of CSA, p_{ij}^{CSA}, we first implement Algorithm 3 and execute it 10000 times with pre-defined parameters (n, m, p, q). We also find the induced transition probability matrix of BSA, p_{ij}^{BSA} and the distribution of the initial state $Pr[S_1 = s]$, similarly. Afterward, we compare the expected value functions of CSA with that of BSA and J-Algo in terms of effectiveness, cost, and total utilities (i.e., weighted sum of effectiveness and cost). Note that J-Algo [5] is to decide the size of the VMs in order to optimize the expectation of saving users in a single shuffle. Hence, in the simulation, the parameter q is not fixed and depends on the output of J-Algo.



(a) Sum of effectiveness and cost, $w_1 = 0.8$ and $w_2 = 0.2$

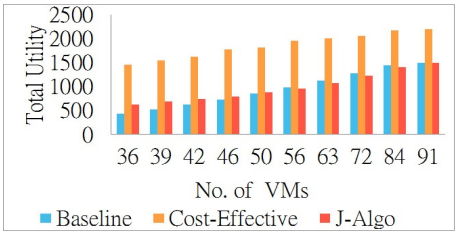


(b) Effectiveness for the speed of the saving users, $\gamma_1 = 0.9$

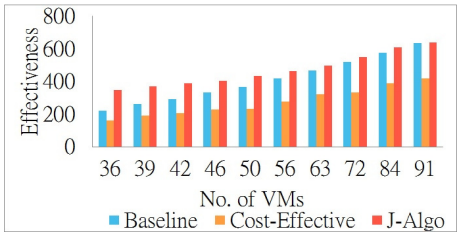


(c) Cost for the No. of stationary users, $\gamma_2 = 1$

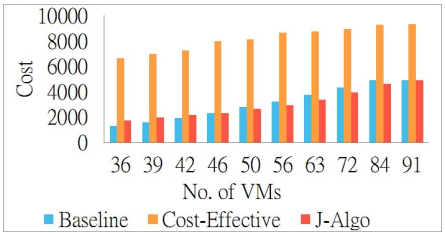
Fig. 1: Comparison of BSA, CSA, and J-Algo [5] with different No. of attackers, $n = 1000$, $p = 50$, $q = 20$, and $t = 10$



(a) Sum of effectiveness and cost, $w_1 = 0.8$ and $w_2 = 0.2$



(b) Effectiveness for the speed of the saving users, $\gamma_1 = 0.9$



(c) Cost for the No. of stationary users, $\gamma_2 = 1$

Fig. 2: Comparison of BSA, CSA, and J-Algo [5] with different No. of VMs, $n = 1000$, $m = 100$, and $t = 10$

Fig. 1 and Fig. 2 first compare different algorithms with varied number of attackers and VMs, respectively. In Fig. 1, 1000 users are involved in the shuffling scheme, and the system is able to allocate at most 50 VMs for shuffling (except J-Algo). In Fig. 2, there are 100 attackers among 1000 users. Fig. 1 demonstrates that the advantage of the shuffling approach linearly decreases when the number of the attackers grows, whereas Fig. 2 manifests that the shuffling approach performs better when the number of VMs increases. For the effectiveness of shuffling, Fig. 1b and Fig. 2b present the speed of the saved users, (i.e., $\mathbb{E}[\bar{V}^{\pi}_{t=10,1}]$) with the discount value set to 0.9. For the cost of shuffling, Fig. 1c and Fig. 2c show the number of stationary users, (i.e., $\mathbb{E}[\bar{V}^{\pi}_{t=10,2}]$), with no discount value (i.e., 1). The simulation results in Fig. 1a and Fig. 2a demonstrate that the total utility (i.e., $\mathbb{E}[\bar{V}^{\pi}_{t=10}]$) of CSA significantly outperforms those of BSA and J-Algo, where the weights of effectiveness and cost are set to 0.8 and 0.2 here.

The result of J-Algo in Fig. 1 does not climb when the number of attackers is smaller than 50, because when the attackers are fewer than the VMs, J-Algo will generate a large q (i.e.,$q > 20$) and lead to a few empty VMs. The effectiveness of shuffling in BSA slightly outperforms the the one in CSA due to the following two reasons. First, the transition probability of CSA decreases to 0 when $j > i + s_i/2$, because with$\lfloor s_i/2 \rfloor$, more crashed VMs in a shuffle can be sanitized at best. Second, BSA improves when the attackers are dense in the VMs. For the shuffling cost, Fig. 1c and Fig. 2c indicate that more users remain stationary in 10 shuffles for CSA, and the rule modification cost of CSA is thereby much smaller than that of BSA. To consider both the effectiveness and cost,

No. of Attackers	20	40	60	80	100
BSA	0.84	0.69	0.55	0.42	0.32
CSA	0.91	0.84	0.79	0.73	0.68

(a) Comparison with different No. of attackers

No. of VMs	39	46	56	72	91
BSA	0.20	0.27	0.37	0.48	0.55
CSA	0.58	0.66	0.73	0.77	0.81

(b) Comparison with different No. of VMs

TABLE I: The approximation ratios $(\alpha_{BSA}, \alpha_{CSA})$ of the baseline and the cost-effective solutions, $w_1 = 0.8$ and $w_2 = 0.2$

our proposed algorithm still outperforms BSA, respectively in Fig. 1a and Fig. 2a. Table. I also presents the corresponding approximation ratios derived from Theorem 1, and the ratios are lose to 1 for CSA in most cases.

IX. Implementation

We implement the shuffling scheme in an experimental SDN/NFV network. The network is composed of 5 HP DL320e Gen8 servers and an HP 5406zl OpenFlow switch. One server is employed to construct the control platform, and the others are installed with the hosts connected to the OpenFlow switch. For the virtual network deployment, we utilize Docker [21] as the VM generator and Open vSwitch [22] to bridge the docker VMs in the physical hosts. We also exploit Ryu [20] and the remote API of Docker to build the control platform of the SDN/NFV network.

In the implementation, we start 50 VMs and assign the memory and CPU resources to each VM for 20 users. The VMs are equally allocated to four servers. Then, we arrange 100 attackers with different IP and TCP ports, where the

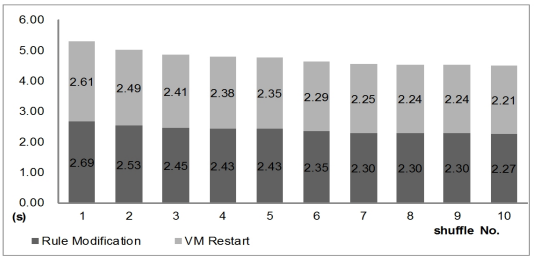


Fig. 3: Time consumption in each shuffle of CSA, $n = 1000$, $m = 100$, $p = 50$, $q = 20$

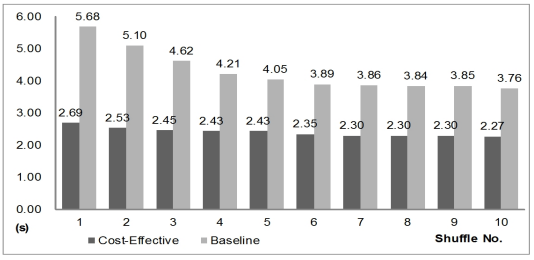


Fig. 4: Time consumption of rule modification in each shuffle of BSA and CSA, $n = 1000$, $m = 100$, $p = 50$, $q = 20$

attackers overload the VMs through the HTTP attack tool. Once a shuffle is performed, some of the users are relocated through rule modification. If a user is assigned to a VM that shares the same server with the previous VM, only two rules (i.e., ingress and egress flows) in the open vSwitch need to be modified. Otherwise, two more rules in the HP OpenFlow switch are required to be reconfigured.

After implementing BSA and CSA in Python and executing the program as an application on the SDN/NFV control platform, Fig. 3 and Fig. 4 present the latency of our approach caused by rule modification and VM restart in each shuffle of different algorithms. The results indicate that restarting a crashed VM takes 0.25s with parallel programming, and modifying 1000 rules takes 0.5s in Open vSwitch and 3.6s in the HP switch. In Fig. 3, our approach requires 4.5-5s in each shuffle, including VM restart and rule modification. In Fig. 4, our approach improves the rule modification latency around 1.5-2.5s in each shuffle, because CSP is more inclined to meet the chance that random pairs of the VMs are colocated. In this case, very little rule modification latency is imposed. In total, CSA outperforms BSA around 18.8s in rule modification latency after all shuffles.

X. Conclusion

In this paper, we explore user traffic redirection in the shuffling-based moving target defense scheme against HTTP DDoS attacks in SDN/NFV. We model the defense effectiveness and rule modification cost with Multi-Objective Markov Decision Processes and design an approximation algorithm to solve it. Simulation results indicate that our proposed CSA effectively balances the tradeoff between the two objectives, and implementation results manifest that our approach imposes much smaller rule modification latency, and each shuffle only takes around 4.5s for redirecting the traffic of 1000 users to 50 VMs.

Acknowledgement

This work is supported by Ministry of Science and Technology of Taiwan under contract No. MOST105-2622-8-009-008 and by Academia Sinica Thematic Research Program.

References

[1] A. R. Curtis, J. C. Mogul, J. Tourrilhes, and P. Yalagandula, "DevoFlow: scaling flow management for high-performance networks," ACM SIGCOMM Computer Communication Review 41.4, pp.254-265, 2011.
[2] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: flexible and elastic DDoS defense," In Proc. USENIX Security, 2015.
[3] N. Furukawa, "Vector valued Markov chain decision processes within countable state space," In R. Hartley, L. C. Thomas, and D. J. White, editors, Recent Developments in Markov Decision Processes, pp. 205-223. Academic Press, New York, 1980.
[4] N. McKeown et al., "OpenFlow: enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review 38(2), pp. 69-74, 2008.
[5] Q. Jia, K. Sun, and A. Stavrou, "Motag: moving target defense against internet denial of service attacks," In Proc. ICCCN, 2013.
[6] Q. Jia, H. Wang, D. Fleck, F. Li, A. Stavrou, and W. Powell, "Catch me if you can: a cloud-enabled ddos defense," 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2014.
[7] M. S. Kang, V. D. Gligor, and V. Sekar, "SPIFFY: inducing cost-detectability tradeoffs for persistent link-flooding attacks." NDSS, 2016.
[8] S. Kandula, D. Katabi, M. Jacob, and A. W. Berger, "Botz-4-Sale: surviving organized DDoS attacks that mimic flash crowds," NSDI, 2005.
[9] S. Ranjan, R. Swaminathan, M. Uysal, A, Nucci, and E. Knightly, "DDoS-shield: DDoS-resilient scheduling to counter application layer attacks," IEEE/ACM Transactions on Networking (TON), 17(1), pp. 26-39, 2008.
[10] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks," In Proc. ACM CCS, 2013.
[11] M. Shtern, R. Sandel, M. Litoiu, C. Bachalo, and V, Theodorou, "Towards mitigation of low and slow application ddos attacks," IEEE International Conference on Cloud Engineering (IC2E), 2014.
[12] S. Venkatesan, M. Albanese, K. Amin, S. Jajodia, and M. Wright, "A moving target defense approach to mitigate DDoS attacks against proxy-based architectures," IEEE Conference on Communications and Network Security, 2016.
[13] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker, "DDoS defense by offense," SIGCOMM Computer Communications Review, Vol. 36, no. 4, pp. 303-314, 2006.
[14] P. Wood, C. Gutierrez, and S. Bagchi, "Denial of Service Elusion (DoSE): Keeping Clients Connected for Less," IEEE 34th Symposium on Reliable Distributed Systems (SRDS), 2015.
[15] L. C. Thomas, "Constrained Markov decision processes as multi-objective problems," In S. French, L. C. Thomas, R. Hartley, and D. J. White, editors, Multi-Objective Decision Making, pp. 77-94. Academic Press, 1983.
[16] Y. Xie, and S. Z. Yu, "A large-scale hidden semi-Markov model for anomaly detection on user browsing behaviors," IEEE/ACM Transactions on Networking (TON), Vol. 17, no. 1, pp. 54-65, 2009.
[17] Y. Xie and S. Z. Yu, "Monitoring the application-layer DDoS attacks for popular websites." IEEE/ACM Transactions on Networking (TON), Vol. 17, no. 1, pp. 15-25, 2009.
[18] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks," IEEE Communications Surveys & Tutorials Vol. 15, no. 4, pp. 2046-2069, 2013.
[19] Network Functions Virtualisation, https://portal.etsi.org/NFV/
[20] Ryu SDN Framework, https://osrg.github.io/ryu/
[21] Docker, https://www.docker.com
[22] Open vSwitch, http://www.openvswitch.org
[23] 2015-2016 RADWARE GLOBAL APPLICATION & NETWORK SECURITY REPORT https://www.radware.com/PleaseRegister.aspx?returnUrl=6442457234
[24] SUCURI BLOG: Analyzing Popular Layer 7 Application DDoS Attacks, https://blog.sucuri.net/2015/09/analyzing-popular-layer-7-application-ddos-attacks.html