A Robustness-Aware Real-Time SFC Routing Update Scheme in Multi-Tenant Clouds

Huaqing Tu, *Gongming Zhao, *Member, IEEE,* *Hongli Xu, *Member, IEEE,* Yangming Zhao, *Member, IEEE,* Yutong Zhai

Abstract—In multi-tenant clouds, requests need to traverse a set of network functions (NFs) in a specific order, referred to as a service function chain (SFC), for security and business logic issues. Due to workload dynamics, the central controller of a multi-tenant cloud needs to frequently update the SFC routing, so as to optimize various network performance, such as load balancing. To achieve effective SFC routing update, we should consider two critical requirements: *system robustness* and *real-time update*. Without considering these two requirements, prior works either result in fragile clouds or suffer from large update delay. In this paper, we propose a robustness-aware real-time SFC routing update (R^3-UA) scheme which takes both requirements into consideration. R^3-UA pursues robustness-aware real-time routing update through two phases: robust NF instance assignment update and real-time SFC routing update. Two algorithms with bounded approximation ratios are proposed for these two phases, respectively. We implement R^3-UA on a real testbed. Both small-scale experimental results and large-scale simulation results show the superior performance of R^3-UA compared with other alternatives.

Index Terms—*Multi-Tenant Clouds, Robustness, Routing Update, Load Balancing.*

1 Introduction

Cloud computing has transformed a large part of the Internet industry and attracted more and more attention from academic and industry communities [2], [3], [4]. In multi-tenant clouds, the cloud vendors (*e.g.*, Amazon Web Services [5] and Google Cloud Platform [6]) lease computing resources to tenants (*e.g.*, enterprises) with virtual machines (VMs). By renting these computing resources, tenants can migrate not only their computation tasks, such as training deep neural networks, but also their network functions (NFs), such as intrusion detection systems and firewalls, to the cloud [7], [8]. It should be noted that for security and business logic issues, a tenant's request should traverse the required NFs in a specific order. For instance, a protected request forwarded to a secure server has to traverse a firewall and then an intrusion detection/prevention system (IDS/IPS) [9]. Usually, such a set of ordered NFs is referred to as a service function chain (SFC) and the problem to manipulate requests to fulfill the SFC requirement is called *SFC routing* [10].

Due to the time-varying workload in a multi-tenant cloud, an SFC routing configuration may be only efficient for a short duration, and an out-of-date SFC routing configuration will result in suboptimal performance in terms of load balancing and end-to-end delay. To pursue high performance in the cloud, we have to periodically update the cloud configuration by the current request/traffic characteristics, which is referred to as *SFC routing update* [10], [11]. With SFC routing update, the cloud will be updated from the out-of-date (or current) routing configuration to the up-to-date one, which will significantly promote the cloud performance, *e.g.*, reduce the NF/link resource utilization. During SFC routing update, we should take two important requirements into considerations for SFC routing update: 1) the up-to-date configuration can satisfy the system robustness requirements when encountering system accidents (*e.g.*, NF failures and attacks from malicious tenants), which is referred to as *system robustness*; 2) the up-to-date configuration should be deployed in a limited time, *i.e.*, *real-time update*.

System robustness requirements mainly come from the widely spread malicious tenants and universal NF failures [12], [13]. On the one hand, malicious tenants may use vicious programs like Bolt [12] to collect information of other tenants as well as the system. With the help of this information, they can launch wide spectrum network attacks, including denial of service (DoS) attacks and co-residency attacks with a high success rate. To alleviate the impact of these attacks, we expect to *limit the number of NF instances traversed by the requests from each tenant*, since a malicious tenant can easily attack all the NF instances carrying its requests. On the other hand, the universal NF failures (incurred by problems such as connectivity errors, hardware faults, and overloads) will also weaken system robustness. It has been shown

- *Some preliminary results of this paper were published in the Proceedings of IEEE/ACM IWQoS 2021 [1]. This paper was supported by the National Science Foundation of China under Grants No. 61822210, 62102392 and 62132019; and in part by the National Science Foundation of Jiangsu Province under Grant BK20210121; and in part by the Anhui Initiative in Quantum Information Technologies under Grant AHY150300. (Corresponding authors: Gongming Zhao; Hongli Xu.)*
- *H. Tu, *G. Zhao, *H. Xu, Y. Zhao and Y. Zhai are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China, 230027, and also with Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou, Jiangsu, China, 215123. E-mail: thq527@mail.ustc.edu.cn, gmzhao@mail.ustc.edu.cn, xuhongli@ustc.edu.cn, zhaoyangming.ustc@gmail.com, zyt1996@mail.ustc.edu.cn.*

that the median time of two consecutive failures is 7.5 hours for firewalls while 5.2 hours for load balancers [13]. For intrusion detection and prevention systems, the median time between two consecutive failures is about 20 minutes [13]. Therefore, NF failures may significantly degrade the robustness of a cloud system. Since the failure of an NF instance will result in an outage to the tenants who have requests being served by it, to mitigate the impact from such universal NF failures and enhance the system robustness, it is also expected to *limit the number of tenants served by each NF instance*.

The real-time update (*i.e.*, complete the update process in a limited time) is required to ensure that the rules are not out-of-date after the update procedure is completed. When there are a large number of requests/flows in the cloud, it will take too much time to deploy the up-to-date configuration. According to [14], it takes at least 0.5 milliseconds (ms) for the central controller to update a routing rule on a commodity switch. Considering that there are usually millions of requests/flows injected into the cloud every minute [15], the routing update may cause a huge overhead to the central controller, and the update delay will be too long to be acceptable. More specifically, in a moderate-size cloud with about $10K$ servers, there may reach $1300K$ flows/min to a server hosting around 15 VMs [15]. Even if there are only 100 flows on each server (less than 1%) that need routing update, it will take the central controller at least $500s$ to update the SFC routing of these flows. Since the workload in the multi-tenant clouds is time-varying, the rules will be out-of-date after the update procedure is completed. Accordingly, we need a real-time SFC routing update scheme in order to complete the SFC routing update in a limited time. Given the time to install one routing rule into a switch, such a scheme can only install a limited number of rules to the switches in the cloud whenever the central controller decides to update the SFC routing.

Previous works of routing update [11], [16], [17], [18] mainly concentrate on improving the performance of the data plane, such as utility maximization or load balancing among links/NF instances. For example, Qu *et al.* [18] study the trade-off problem between network load balancing and route reconfiguration overhead. They formulate this problem as a multi-objective mixed integer optimization and propose a heuristic algorithm to solve this problem. However, these works do not consider the requirements of system robustness and real-time update, thus may lead to weak system robustness and long update delay.

To satisfy these two requirements for SFC routing update in multi-tenant clouds, this paper proposes a robustness-aware real-time SFC routing update (R^3-UA) scheme. R^3-UA pursues robustness-aware real-time routing update through two phases: robust NF instance assignment and real-time SFC routing update. The main contributions of this paper are as follows:

1) We propose a robustness-aware real-time SFC routing update (R^3-UA) scheme for multi-tenant clouds, which includes two phases: robust NF instance assignment update and real-time SFC routing update.
2) For robust NF instance assignment update, we formulate this problem as an integer linear programming. An algorithm with the approximation factor of $O(\log h)$ is designed to solve this problem, where h is the number of NF instances.
3) For real-time SFC routing update, we propose an algorithm with a bounded approximation factor, which can complete the update process under a given delay constraint.
4) We evaluate the proposed algorithms with small-scale testbed experiments and large-scale simulations using real-world topologies and datasets. The results show R^3-UA can decrease update delay by 70% and reduce the impact of malicious tenants and NF failures on the system robustness compared with other alternatives.

2 Related Works

Almost all the previous methods about routing update concentrate on improving the performance of the data plane, and ignore the requirements of system robustness and real-time update. According to whether or not to consider SFC requirement, we discuss the related works in the following two categories.

Routing Update without SFC: Most of the previous works first compute the target routing configuration based on the current workload, and then deploy the target routing configuration in the system. For example, Xu *et al.* [19] study the route update by joint optimization of route selection in the control plane and update scheduling in the data plane. They formulate this problem as a linear program, and propose two algorithms with bounded approximation factors. Zheng *et al.* [20] study how to reroute the updates of multiple network flows in a synchronized software-defined network in a congestion-free manner. Then they proposed their approach based on a time-extended network construction and resource dependency graph, and implemented this method by Openflow 1.5 using the scheduled bundles feature. Diman *et al.* [21] design an efficient rerouting approach in software-defined wide area networks to avoid the overload on the overlay path, then propose a centralized control approach to minimize the network reconfiguration cost.

Routing Update with SFC: Zhang *et al.* [16] design Co-Scaler, which focuses on the cooperative scaling of an SFC with multiple NF instance pools. Through the logically centralized control of the software-defined network and the dynamic function provisioning of network function virtualization, Co-Scaler computes the subset of existing flows, then migrates them to new scaled instances when scaling out, which could decrease the number of packets affected by buffering significantly. Qu *et al.* [18] propose a dynamic flow migration problem for embedded services in NFV/SDN-enabled 5G core networks. They first formulate this problem as a multi-objective mixed integer program, then transform it to a tractable mixed-integer quadratically constrained programming problem, and design a heuristic algorithm based on redistribution of hop delay bounds.

All of the previous works ignore two critical requirements: system robustness and real-time update. Thus, these works result in low robustness caused by malicious tenants and NF failures, and the target route configuration can be out-of-date after the update procedure is completed due to the long update delay.

3 Preliminaries

3.1 Multi-Tenant Cloud Model

A typical multi-tenant cloud consists of four components: a service function set, a computing node set, a link set and a central controller.

1) The service function set is composed of different NF instances. Assume that there are b types of NFs, which is denoted as $S = \{s_1, s_2, ..., s_b\}$. For clearly problem formulation, we use $M_s = \{m_1^s, m_2^s, ..., m_{h_s}^s\}$ to represent the set of NF instances of service type $s \in S$, where $h_s = |M_s|$ is the number of NF instances with type s. The total number of NF instances in the cloud is denoted as h, *i.e.*, $h = \sum_{s \in S} h_s$. We also use set $M = M_1 \cup M_2 \cup ... \cup M_s$ to denote the set of all the NF instances. It should be noted that every NF instance can provide services to limited amount of traffic incurred by requests, and such processing capacity is denoted by C_i^s for NF instance $m_i^s \in M$.
2) A set of computing nodes is responsible for providing computing resources to tenants via creating VMs.
3) A set of l links $E = \{e_1, e_2, ..., e_l\}$ is used to realize the data transmission between different NFs/VMs. Let C_e be the capacity of link $e \in E$.
4) The central controller is responsible for managing the whole cloud system, *e.g.*, determining the up-to-date routing configuration and updating the SFC routing.

Tenants rent VMs and buy services from cloud vendors according to their needs in multi-tenant clouds. A set of n tenants is denoted as $T = \{t_1, t_2, ...t_n\}$. Different tenants may generate traffic with various service requirements on different computing nodes. We identify a *request* by four elements <source, destination, SFC requirement, tenant>. It should be noted that every tenant can generate a certain number of requests. The request set is denoted as $\Gamma = \{\gamma_1, \gamma_2, ..., \gamma_d\}$, where $d = |\Gamma|$ is the number of requests in the cloud. Let $f(\gamma)$ be the traffic amount associated with request $\gamma \in \Gamma$. For ease of reference, Table 1 summarizes the key notations used in this paper.

TABLE 1: Key Notations

Parameters	Description
S	a network service type set
M	an NF instance set
M_s	an NF instance set with service type $s \in S$
h	the number of NF instances
h_s	the number of NF instances with type $s \in S$
C_i^s	the processing capacity of NF instance $m_i^s \in M_s$
E	a link set
C_e	the capacity of link $e \in E$
T	a tenant set
$f(t)$	the total traffic demand of tenant t
$I_{t,s}^s$	whether the NF instance m_i^s is assigned to tenant t or not before the update
Γ	a request set
$f(\gamma)$	the traffic amount associated with request γ
$\mathcal{P}_\gamma$	the set of candidate routing paths for request $\gamma \in \Gamma$
k	the maximum number of NF instances traversed by the requests from a tenant
q	the maximum number of tenants that can be served by a NF instance
U	the update delay threshold

3.2 Problem Statement

R^3-UA updates SFC routes with the granularity of requests. Specifically, all the traffic belonging to the same request will be assigned to the same NF instances and go through the same path. However, the packets for different requests, even if they belong to the same tenant, may be delivered to different instances through different paths. In general, the goal of R^3-UA is to pursue load balancing among all links and NF instances as in many previous works [11], [17]. However, there are two specific requirements that should be satisfied when pursuing system load balancing.

1) During the update process, the system robustness requirement should be taken into consideration. A specific tenant will be assigned with only k NF instances, where k is a system-specific parameter. By limiting the number of NF instances that are providing services to an arbitrary tenant, and leveraging the VM isolation techniques [22], we can mitigate the impact of attacks launched by a malicious tenant. For each NF instance, it can provide services to at most q tenants. Again, q is a system-specific parameter. This constraint limits the impact of a single NF failure. By satisfying these two constraints during SFC routing update, we can enhance system robustness.
2) Due to the traffic dynamics in multi-tenant clouds, it is required to complete the update procedure in a pre-defined time threshold U. Otherwise, the routing configuration would be out-of-date when the update process is completed. Assume the central controller can update one rule in time τ, this requirement limits the number of rules that the central controller can refresh in the update procedure.

In summary, R^3-UA is to pursue the system load balancing among all links and NF instances while satisfying the above two requirements, real-time update and system robustness, for SFC routing update in multi-tenant clouds.

3.3 Algorithm Workflow

Inspired by [11], R^3-UA will be invoked periodically or event-driven (*e.g.*, NF failures or link congestion). During the update process, we need to update the assignment between NF instances and tenants for robustness requirements (*i.e.*, update NF instance assignment), and update the SFC routing for each request (*i.e.*, update SFC routing). Regarding the update period, we have the following thoughts: 1) Requests/traffic dynamics are very common in multi-tenant clouds, which requires the SFC routing update in a short duration. 2) Although requests are dynamic, the total traffic of a tenant is relatively stable [23]. From this point of view, we can update NF instances assigned to a tenant at a longer interval. 3) When we update the NF instance set of a tenant, the newly allocated NF instances need to back up the state information of the tenant's requests [24]. The traffic fluctuation during this period will affect tenants' QoS. Therefore, the NF instances assigned to a tenant should not be updated frequently.

In practice, R^3-UA should invoke NF instance assignment and SFC routing update in different frequencies. Thus, similar to [25], R^3-UA pursues robustness-aware real-time routing update through two phases: *robust NF instance assignment* and *real-time SFC routing update*. The first phase is performed in a long-term granularity (*e.g.*, 10 minutes) to assign NF instances to tenants under the system robustness constraints (*i.e.*, the first requirement discussed in Section 3.2). The objective of the first phase is to minimize the total amount of traffic that needs to be migrated. The second phase is performed in a short-term granularity (*e.g.*, 1 minute) to update the SFC routing of each request under delay constraint (*i.e.*, the second requirement discussed in Section 3.2).

4 Robust NF Assignment Update

4.1 Problem Formulation

Now, we give the problem formulation of the robust NF instance assignment update (RNAU) problem. When we update the NF instance assignment, some of the requests have to be migrated to other NF instances, which not only introduces considerable system overhead, but also incurs more routing rule updates. Accordingly, when we consider updating the NF instance assignment, the objective is to minimize the total traffic that needs to be migrated. Let $I^s_{t,i}$ be a binary constant as follows: if NF instance m^s_i is assigned to tenant t before the update, we set $I^s_{t,i} = 1$; otherwise $I^s_{t,i} = 0$. $b^s_{t,i}$ represent the traffic amount of tenant t processed on NF instance m^s_i, $f(t)$ denote the total traffic demand of tenant t. For tenant $t \in T$, we use R^s_t to indicate the proportion of traffic that needs to be served by service of type $s \in S$. The controller can use traffic measurement methods such as sampling or sketch [26] [27] to get each tenants traffic change information for a period of time since the last update. Based on this information, the controller can get relatively stable values of $f(t)$ and R^s_t. It should be noted that R^s_t is a constant value belonging to $[0, 1]$. If tenant t does not need the service of type s, R^s_t is set to 0. With these notations, the RNAU problem can be formulated as follows:

$$\min \sum_{t\in T}\sum_{s\in S}\sum_{i:m^s_i\in M_s} b^s_{t,i}\cdot I^s_{t,i}\cdot(1-y^s_{t,i})$$

$$S.t. \begin{cases} \sum\limits_{i:m^s_i\in M_s} x^s_{t,i} = R^s_t, & \forall t\in T, s\in S\\ x^s_{t,i} \le y^s_{t,i}, & \forall t\in T, s\in S, m^s_i\in M_s\\ \sum\limits_{i:m^s_i\in M_s} y^s_{t,i} \le k, & \forall t\in T, s\in S\\ \sum\limits_{t\in T} y^s_{t,i} \le q, & \forall s\in S, m^s_i\in M_s\\ \sum\limits_{t\in T} x^s_{t,i}\cdot f(t) \le C^s_i, & \forall s\in S, m^s_i\in M_s\\ x^s_{t,i}\in[0,1], & \forall t\in T, s\in S, m^s_i\in M_s\\ y^s_{t,i}\in\{0,1\}, & \forall t\in T, s\in S, m^s_i\in M_s \end{cases} \quad (1)$$

where variable $x^s_{t,i}$ is the traffic proportion of tenant t served by an NF instance $m^s_i \in M_s$, and binary variable $y^s_{t,i}$ denotes that whether the NF instance $m^s_i \in M_s$ is allocated to tenant t or not.

The first set of equations means that the traffic proportion of tenant $t \in T$ which receives service of type $s \in S$ should be equal to R^s_t. The second set of inequalities says that the traffic of tenant t can be processed by NF instance m^s_i if and only if NF instance m^s_i is allocated to tenant t. The third set of inequalities denotes that each tenant's traffic will be processed by at most k NF instances. The fourth set of inequalities represents that each NF instance can process traffic from at most q tenants. The last set of inequalities limits the traffic load on each NF instance.

4.2 Robust NF Insatance Assignment Algorithm

Due to the binary nature of the variables in Eq. (1), it is an NP-Hard problem, which is difficult to solve efficiently [25]. In this section, we propose an NF instance assignment update algorithm (NAUA) to solve Eq. (1) in polynomial time. In this algorithm, there are two steps. The first step is to formulate it as a linear programming which can be efficiently solved, by replacing $\{y^s_{t,i}\}$ with its fractional version. After being relaxed, the variable $\{y^s_{t,i}\}$ means the probability of NF instance m^s_i assigned to tenant t. Suppose the optimal solutions of the relaxed version of Eq. (1) are $\{\widetilde{x}^s_{t,i}\}$ and $\{\widetilde{y}^s_{t,i}\}$, in the second step, we determine how to assign NF instances to each tenant, based on such optimal solutions. For each type of network service $s \in S$, we first calculate $k(t) = \lfloor\sum_{i:m^s_i\in M_s} \widetilde{y}^s_{t,i}\rfloor$, which will be the number of NF instances of service type s that should be assigned to tenant t (we will see in section 4.3 that $k(t)$ must be smaller than k). Then we divide all the NF instances into $k(t)$ groups by pursuing load balancing among groups, *i.e.*, minimize the maximum value of $\sum_{\widetilde{y}^s_{t,i}\in g} \widetilde{y}^s_{t,i}$, where g denote a group. For each group g, we assign NF instance m^s_i to tenant t with probability $\frac{\widetilde{y}^s_{t,i}}{z_g}$, and the traffic proportion of tenant t processed by this instance is set to $\frac{\widetilde{x}^s_{t,i}\cdot z_g}{\widetilde{y}^s_{t,i}}$, where z_g is the sum of values of instances in group g. Till now, we have assigned $k(t)$ NF instances with service $s \in S$ for each tenant $t \in T$. The above algorithm is summarized in Algorithm 1.

4.3 Performance Analysis

This section analyzes the approximation performance of the proposed NAUA algorithm. We first propose the following theorem.

Theorem 1. The proposed NAUA algorithm guarantees that each tenant $t \in T$ will be allocated at most k NF instance with service type $s \in S$.

Proof: For each tenant $t \in T$ and network service $s \in S$, we put all NF instances with value $\widetilde{y}^s_{t,i}$ into $k(t)$ groups in order to minimize the maximum value of $\sum_{\widetilde{y}^s_{t,i}\in g} \widetilde{y}^s_{t,i}$, where g denote a group. Then we choose an NF instance in each group with probability $\frac{\widetilde{y}^s_{t,i}}{z_g}$. Thus, we will get $k(t)$ NF instances in total. According to the third constraints in Eq. (1) and the definition of $k(t)$, we have:

$$k(t) = \lfloor \sum_{i:m^s_i\in M_s} \widetilde{y}^s_{t,i}\rfloor \le \sum_{i:m^s_i\in M_s} \widetilde{y}^s_{t,i} \le k \quad (2)$$

Algorithm 1 NF Instance Assignment Update Algorithm

```
1:  Step 1: Solving the Relaxed Problem
2:  Construct a linear program by replacing with $y^s_{t,i} \in [0,1]$ in Eq. (1)
3:  Obtain the optimal solutions $\{\widetilde{x}^s_{t,i}\}$ and $\{\widetilde{y}^s_{t,i}\}$
4:  Step 2: Instance Assignment to Each Tenant
5:  for each tenant $t \in T$ do
6:    for each network service $s \in S$ do
7:      Let $k(t) = \lfloor \sum_{i:m^s_i \in M_s} \widetilde{y}^s_{t,i} \rfloor$
8:      Put variables in $\{\widetilde{y}^s_{t,i}\}$ into $k(t)$ group by pursuing load balancing among groups
9:      for each group $g$ do
10:       Let $z_g$ be the sum of values of instances in group $g$
11:       Assign NF instance $m^s_i$ in group $g$ to tenant $t$ with probability $\frac{\widetilde{y}^s_{t,i}}{z_g}$
12:       if NF instance $m^s_i$ is assigned to tenant $t$ then
13:         Set the traffic proportion of tenant $t$ processed by instance $m^s_i$ to $\frac{\widetilde{x}^s_{t,i}\cdot z_g}{\widetilde{y}^s_{t,i}}$
14:       end if
15:     end for
16:   end for
17: end for
```

It means that $k(t) \le k$, *i.e.*, we can strictly guarantee that the number of NF instances with service type s assigned to tenant t will not exceed k. □

Lemma 2. For any group g, the lower bound of the sum z_g is greater than 0.5 and not greater than $\frac{k(t)}{2k(t)-1}$.

Proof: First, we prove that the lower bound is greater than 0.5. For each tenant t and network service $s \in S$, according to the definition of $k(t)$, it follows that:

$$\sum_{i:m^s_i \in M_s} \widetilde{y}^s_{t,i} = k(t) + \epsilon, \quad 0 < \epsilon < 1 \tag{3}$$

Now, we give the definition of two sets as follows:

$$\begin{cases} Y_1 = \{\widetilde{y}^s_{t,i} | 0.5 < \widetilde{y}^s_{t,i} < 1, m^s_i \in M_s\} \\ Y_2 = \{\widetilde{y}^s_{t,i} | 0 < \widetilde{y}^s_{t,i} < 0.5, m^s_i \in M_s\} \end{cases} \tag{4}$$

In this way, we can put the values in a group into these two sets according to the definition of these two sets. Then we arbitrarily select two values denoted as y^s_{t,i_1} and y^s_{t,i_2} from set Y_2, and compute their sum denoted as y^s_{t,i_3}. If the value of y^s_{t,i_3} is less than 0.5, we put this new value into Y_2, *i.e.*, $Y_2 = Y_2 - \{y^s_{t,i_1}, y^s_{t,i_2}\} + \{y^s_{t,i_3}\}$. Otherwise we put it into set Y_1 and remove these two values from Y_2, *i.e.*, $Y_1 = Y_1 + \{y^s_{t,i_3}\}$ and $Y_2 = Y_2 - \{y^s_{t,i_1}, y^s_{t,i_2}\}$. We repeat the operations until there is at most one value in Y_2. We make an assumption that there are less than $k(t)$ values in Y_1, which means that the number of values in Y_1 is at most $k(t) - 1$. According to the definition of Y_1, the values in Y_1 are all less than 1. We have:

$$\sum_{i:y^s_{t,i} \in Y_1} \widetilde{y}^s_{t,i} < k(t) - 1 \tag{5}$$

Since there is only one value in Y_2, the sum of values in Y_2 is:

$$\sum_{i:y^s_{t,i} \in Y_2} \widetilde{y}^s_{t,i} < 0.5 \tag{6}$$

Then it follows that:

$$\sum_{i:m^s_i \in M_s} \widetilde{y}^s_{t,i} = \sum_{i:y^s_{t,i} \in Y_1} \widetilde{y}^s_{t,i} + \sum_{i:y^s_{t,i} \in Y_2} \widetilde{y}^s_{t,i} < k(t) - 0.5 \tag{7}$$

Eq. (7) contradicts $\sum_{i:m^s_i \in M_s} \widetilde{y}^s_{t,i} = k(t) + \epsilon, 0 < \epsilon < 1$ in Eq. (3), which means the assumption that there are less than $k(t)$ values in Y_1 does not hold. Thus, there are at least $k(t)$ values in Y_1. Since we use min-max sum strategy to put instances into groups, the total value z_g in any group g must be greater than 0.5.

Now we prove that the lower bound of z_g is not greater than $\frac{k(t)}{2k(t)-1}$. We assume that there are $2k(t) - 1$ instances with identical values $\frac{k(t)}{2k(t)-1}$. When we assign these values to $k(t)$ groups, it is obvious that there exists a group which only has one instance with value $\frac{k(t)}{2k(t)-1}$. Thus, there does not exist an assignment scheme to make sure the sum of values of instances in any group is greater than $\frac{k(t)}{2k(t)-1}$. □

In order to facilitate the description of approximation analysis, we now give two famous lemmas for probability analysis.

Theorem 3 (Union Bound). Given a countable set of n events: $A_1, A_2, ..., A_n$, each event A_i happens with possibility $\mathbf{Pr}(A_i)$. Then, $\mathbf{Pr}(A_1 \cup A_2 \cup ... \cup A_n) \le \sum_{i=1}^n \mathbf{Pr}(A_i)$.

Theorem 4 (Chernoff Bound). Given n independent variables: $y_1, y_2, ..., y_n$, where $\forall z_i \in [0,1]$. Let $\mu = \mathbb{E}[\sum_{i=1}^n y_i]$. Then, $\mathbf{Pr}[\sum_{i=1}^n y_i \ge (1+\epsilon)\mu] \le e^{\frac{-\epsilon^2 \mu}{2+\epsilon}}$, in which ϵ is an arbitrary value greater than 0.

Assuming that the minimum capacity of NF instances is denoted by C^{min}_m, we define a constant value α as follows:

$$\alpha = \min\left\{\min\left\{\frac{C^{min}_m}{f(t)}, t \in T\right\}, q\right\} \tag{8}$$

Theorem 5. The proposed NAUA algorithm can achieve the approximation factor of $\frac{3 \cdot \log h}{\alpha} + 3$ for the NF instance capacity constraints.

Proof: We first prove that for each tenant $t \in T$ and service type $s \in S$, we have $\mathbb{E}[\hat{x}^s_{t,i}] = \widetilde{x}^s_{t,i}$. Specifically, for any variable $\widetilde{x}^s_{t,i}$, we set $\hat{x}^s_{t,i} = \frac{\widetilde{x}^s_{t,i} \cdot z_g}{\widetilde{y}^s_{t,i}}$ with probability of $\frac{\widetilde{y}^s_{t,i}}{z_g}$. Thus, we have $\mathbb{E}[\hat{x}^s_{t,i}] = \widetilde{x}^s_{t,i}$.

Let variable $\varphi^s_{t,i}$ be the load generated by tenant $t \in T$ on NF instance $m^s_i \in M$ with network service $s \in S$. The expectation of traffic load of NF instance m^s_i is:

$$\begin{aligned} \mathbb{E}\left[\sum_{t \in T} \varphi^s_{t,i}\right] &= \sum_{t \in T} \mathbb{E}[\varphi^s_{t,i}] \\ &= \sum_{t \in T} \widetilde{x}^s_{t,i} \cdot f(t) \le C^s_i \end{aligned} \tag{9}$$

Eq. (9) shows that the NAUA algorithm can guarantee that the expectation of traffic load of NF instance m^s_i is less than the process capacity C_i^s. Combining the definition of α in Eq. (8), we have:

$$\begin{cases} \frac{\varphi_{t,i}^s \cdot \alpha}{C_i^s} \in [0,1] \\ \mathbb{E}\left[\sum_{t\in T} \frac{\varphi_{t,i}^s \cdot \alpha}{C_i^s}\right] \le \alpha \end{cases} \tag{10}$$

By applying Chernoff Bound Theorem, we assume that ρ is an arbitrary positive value. It follows:

$$\textbf{Pr}\left[\sum_{t\in T} \frac{\varphi_{t,i}^s \cdot \alpha}{C_i^s} \ge (1+\rho)\cdot\alpha\right] \le e^{\frac{-\rho^2\cdot\alpha}{2+\rho}} \tag{11}$$

Now, we assume that:

$$\textbf{Pr}\left[\sum_{t\in T} \frac{\varphi_{t,i}^s \cdot \alpha}{C_i^s} \ge (1+\rho)\right] \le e^{\frac{-\rho^2\cdot\alpha}{2+\rho}} \le \frac{1}{h^3} \tag{12}$$

where h is the number of NF instances. The solution for Eq. (12) can be expressed as:

$$\rho \ge \frac{3\cdot\log h + \sqrt{9\cdot\log^2 h + 24\cdot\alpha\cdot\log h}}{2\lambda}$$
$$\Rightarrow \rho \ge \frac{3\log h}{\alpha} + 2, \quad h \ge 2 \tag{13}$$

Then using Union Bound Theorem, we have:

$$\textbf{Pr}\left[\bigvee_{i:m_i^s\in M_s} \sum_{t\in T} \frac{\varphi_{t,i}^s \cdot \alpha}{C_i^s} \ge (1+\rho)\right]$$
$$\le \sum_{i:m_i^s\in M_s} \textbf{Pr}\left[\sum_{t\in T} \frac{\varphi_{t,i}^s \cdot \alpha}{C_i^s} \ge (1+\rho)\right]$$
$$\le h\cdot\frac{1}{h^3} \le \frac{1}{h^2}, \quad \rho \ge \frac{3\log h}{\alpha} + 2 \tag{14}$$

Eq. (14) means that the proposed NAUA algorithm can guarantee that the total load on NF instance $m_i^s \in M_s$ will not exceed the fractional solution by a factor of $1+\rho = \frac{3\cdot\log h}{\alpha} + 3$ with high probability. □

Theorem 6. The proposed NAUA algorithm guarantees that the number of tenants handled by an NF instance with service type s will not exceed q by a factor of $2\cdot(\frac{3\cdot\log h}{\alpha} + 3)$.

Proof: According to the second step of NAUA algorithm, in each group g, NF instance m_i^s is chosen with probability of $\frac{\widetilde{y}_{t,i}^s}{z_g}$. Combining Lemma 2, we have:

$$\textbf{Pr}\left[\hat{y}_{t,i}^s = 1\right] = \frac{\widetilde{y}_{t,i}^s}{z_g} \le 2\cdot\widetilde{y}_{t,i}^s \tag{15}$$

Then we can analyze the approximation ratio performance based on the randomized rounding method. Since the proof process is similar to that of Theorem 5 and the space is limited, we omit it here. □

Theorem 7. After the rounding process, the total traffic migrated from old NF instances into newly allocated NF instances will not exceed the fractional solution.

Proof: According to Lemma 2, we have:

$$\textbf{Pr}\left[\hat{y}_{t,i}^s = 1\right] = \frac{\widetilde{y}_{t,i}^s}{z_g} \ge \frac{2\cdot k(t)-1}{k(t)}\cdot\widetilde{y}_{t,i}^s \ge \widetilde{y}_{t,i}^s \tag{16}$$

Then, with the NAUA algorithm, the total traffic migrated from old NF instances onto newly allocated NF instances is:

$$\sum_{t\in T}\sum_{s\in S}\sum_{i:m_i^s\in M_s} b_{t,i}^s \cdot I_{t,i}^s \cdot (1-\hat{y}_{t,i}^s)$$
$$\le \sum_{t\in T}\sum_{s\in S}\sum_{i:m_i^s\in M_s} b_{t,i}^s \cdot I_{t,i}^s \cdot (1-\widetilde{y}_{t,i}^s) \tag{17}$$

Eq. (17) means that the NAUA algorithm can guarantee that the optimization objective will not exceed the fractional solution after randomized rounding. Similar to Theorem 5 and 6, The NAUA algorithm can also achieve the approximation factor of $O(\log h)$ for the objective value. □

In conclusion, after the NF instance allocation is updated, the capacity of each NF instance and the constraints on the number of tenants served by each NF instance will not be violated by a factor of $O\left(log\ h\right)$, and NAUA can achieve the approximation factor of $O\left(log\ h\right)$ for the objective value. Thus, we can conclude that NAUA can achieve bi-criteria approximation factor [28].

4.4 Discussion

Actually, the robustness constraints proposed in this paper can also be applied to other scenarios. In the following, we give two practically application scenarios.

Virtual Machine (VM) Placement [29]. In data centers, a server usually hosts the VMs of multiple tenants. We can apply the robustness constraints to the problem of VM placement. Specifically, to prevent the failure of a server from affecting too many tenants, we can limit the number of tenants that a server provides resources to. In addition, to limit the range of servers that a malicious tenant can attack, we can restrict the number of servers where a tenant can place VMs.

Task Offloading [30]. In edge clouds, users tasks are offloaded to edge servers to achieve better application responsiveness. We can limit the number of users that an edge server can serve to control the range of tenants affected by an edge device failure. Moreover, to limit the range of edge servers attacked by users malicious requests, we can restrict the number of edge servers assigned to each user.

5 Real-Time SFC Routing Update (RTU)

5.1 Problem Definition for RTU

Using the NAUA algorithm, we can derive the set of NF instances assigned to each tenant. On the basis of this assignment, we first construct a set of candidate routing paths for each request $\gamma \in \Gamma$ (denoted as $\mathcal{P}_\gamma$), such that its SFC requirement and robustness constraints can be satisfied with any path in this candidate set. For each request, the SFC routing path will be recorded in the packet header by the ingress switch. Though there is no path field in classic IP packets, we can use fields like MPLS labels or other unused fields in the packet header as tags to record the SFC routing information [10], [11], [31]. Whenever the route of a request is updated, the central controller updates the corresponding rules in the ingress switch. In general, the time to update a rule is relatively stable (*e.g.*, 0.5 milliseconds according to [14]), which will be denoted as τ in the following.

To update the SFC routing in a real-time manner, we need to select a new path from the candidate path set for each request under a given delay constraint. Suppose we want to complete the SFC routing update in time U, we can only update routes for a limited number of requests (*i.e.*, $\frac{U}{\tau}$ requests). Assuming that the path for request γ before and after the update is p^* and p, respectively, then we use $t(\gamma, p, p^*)$ to indicate whether the updated path p is the same as the path p^* or not. If p and p^* are the same, which means that the routing path of request γ does not need to be updated, we have $t(\gamma, p, p^*) = 0$. Otherwise, $t(\gamma, p, p^*) = 1$, which means the route of request γ needs to be updated. With these notations, we can formulate the RTU problem as follows:

$$\min \quad \psi$$
$$S.t. \begin{cases} \sum\limits_{\gamma\in\Gamma}\sum\limits_{p\in\mathcal{P}_\gamma} z_\gamma^p \cdot t(\gamma,p,p^*)\cdot\tau \le U \\ \sum\limits_{p\in\mathcal{P}_\gamma} z_\gamma^p = 1, & \forall\gamma\in\Gamma \\ \sum\limits_{\gamma\in\Gamma}\sum\limits_{p\in\mathcal{P}_\gamma: m_i^s\in p} z_\gamma^p \cdot f(\gamma) \le C_i^s\cdot\psi, & \forall s\in S, m_i^s\in M_s \\ \sum\limits_{\gamma\in\Gamma}\sum\limits_{p\in\mathcal{P}_\gamma: e\in p} z_\gamma^p \cdot f(\gamma) \le C_e\cdot\psi, & \forall e\in E \\ z_\gamma^p\in\{0,1\}, & \forall\gamma\in\Gamma, p\in\mathcal{P}_\gamma \end{cases} \tag{18}$$

where binary variable z_γ^p denotes whether the request γ selects the feasible path $p \in \mathcal{P}_\gamma$ or not.

The first set of inequalities says the update delay, *i.e.*, the time to update the SFC routing in flow tables, cannot exceed the threshold U. The second set of equations means that every request will be forwarded through a path in the candidate path set. The third and fourth sets of inequalities express that the updated traffic load on each NF instance and link cannot exceed $C_m \cdot \psi$ and $C_e \cdot \psi$, respectively, where ψ is the maximum resource (for both NF instances and links) utilization. Our objective is to achieve the load balance among links and NF instances, *i.e.*, minimize the maximum resource utilization ψ.

It should be noted that in the RTU problem, after updating the route configuration, each NF instance does not need to be equal to the output of the first phase. When solving the RNAU problem, we obtain the traffic demand of each tenant through the long-term observation and statistical collection since the last update. When solving the RTU problem, the rounding-based routing update (RBRU) algorithm first collects the currently existing requests in the system, and then determines the up-to-date configuration according to the NF instance assignment results in the first phase. Therefore, the traffic information input of the first stage comes from long-term statistical collection, and that of the second phase comes from the currently existing requests in the system. That is, the traffic inputs in these two phases are fully different. Hence, after updating the route configuration, each tenants traffic proportion served by each NF instance does not need to be equal to the output of the first phase.

5.2 Algorithm to Solve RTU

Since the variable z_γ^p is binary, it is difficult to solve RTU in a timely manner. Accordingly, in this section, we propose a rounding-based routing update (RBRU) algorithm to efficiently solve the RTU problem. This algorithm consists of two steps. Similar to Algorithm 1, we first relax $z_\gamma^p \in [0, 1]$ in (18) and derive a linear programming problem. In the solution of such a relaxed linear programming problem, each request may be split and routed through multiple paths, which is not feasible to RTU. Accordingly, in the second step, we will choose a unique path for each request and obtain the integer solutions $\{\hat{z}_\gamma^p\}$ based on a rounding method. Assume the optimal solutions of the relaxed linear programming problem are $\{\widetilde{z}_\gamma^p\}$, then for each request $\gamma \in \Gamma$ and feasible path $p \in \mathcal{P}_\gamma$, we set $\hat{z}_\gamma^p = 1$ with probability $\widetilde{z}_\gamma^p$. That is, the request γ will be routed through path p with probability $\widetilde{z}_\gamma^p$. We summarize the RBRU algorithm in Algorithm 2.

Algorithm 2 RBRU: Rounding-Based Routing Update

```
1:  Step 1: Solving the Relaxed Routing Update Problem
2:  Construct a linear programming by replacing with $z_\gamma^p \in [0,1]$
    in Eq. (18)
3:  Obtain the optimal solutions $\{\widetilde{z}_\gamma^p\}$
4:  Step 2: Selecting Routes Using Randomized Rounding
5:  Derive an integer solution $\{\hat{z}_\gamma^p\}$ by randomized rounding
6:  for each request $\gamma \in \Gamma$ do
7:    for each feasible path $p \in \mathcal{P}_\gamma$ do
8:      Set $\hat{z}_\gamma^p = 1$ with probability of $\widetilde{z}_\gamma^p$
9:      if $\hat{z}_\gamma^p = 1$ then
10:       Request $\gamma$ will be routed through path $p$
11:     end if
12:   end for
13: end for
```

5.3 Performance Analysis

In this section, we analyze the approximate performance of the proposed RBRU algorithm. Let the minimum capacity of all links and that of all NF instances be denoted by C_e^{min} and C_m^{min}, respectively. We first define a constant variable β as follows:

$$\beta = \min\{\min\{\frac{C_e^{min}\cdot\widetilde{\psi}}{f(\gamma)}, \frac{C_m^{min}\cdot\widetilde{\psi}}{f(\gamma)}, \gamma\in\Gamma\}, \frac{U}{\tau}\} \tag{19}$$

We present the following theorems to illustrate the approximation ratio of the RBRU algorithm. Since the proof process is similar to that of Theorem 5, we omit the detail of the proofs here.

Theorem 8. After the rounding process, the update delay will not exceed the threshold U by a factor of $\frac{3\cdot\log h}{\beta} + 3$ with high probability.

Theorem 9. The RBRU algorithm guarantees that the total traffic on any NF instance will not exceed the amount according to the fractional solution by a factor of $\frac{3\cdot\log h}{\beta} + 3$.

Theorem 10. The proposed RBRU algorithm guarantees that the total traffic on any link $e \in E$ will not exceed the traffic of the fractional solution by a factor of $\frac{3\cdot\log|E|}{\beta} + 3$.

Approximation Factors: From the above theorems, by updating the routing path of the selected requests on chosen paths, the update delay constraint will hardly be violated by a factor of $\frac{3\cdot\log h}{\beta}+3$, and the processing capacity of NF instances will hardly be violated by a factor of $\frac{3\cdot\log h}{\beta}+3$, and the link capacity will hardly be violated by a factor of $\frac{3\cdot\log|E|}{\beta}+3$. It means that the algorithm can achieve the optimal solution, violating the update delay constraint by at most a factor $\frac{3\cdot\log h}{\beta}+3$, the NF instance capacity by at most a factor $\frac{3\cdot\log h}{\beta}+3$, and the link capacity by at most a factor $\frac{3\cdot\log|E|}{\beta}+3$. By using the traffic controlling method, the intensity of each request can be limited to a specific value, so that the network congestion can be avoided.

It should be noted that RBRU can achieve almost the constant approximation in most practical situations. For example, let $\widetilde{\psi}$ be 0.4. Consider a large-scale clouds with $h=1000$ NF instances and $|E|=1000$ links, the update delay constraint $U=2s$. The central controller needs $0.5ms$ to update a forwarding rule in the data plane [14]. Observing the practical traffic traces, the maximum intensity of a request can reach 10Mbps [32]. The capacity of links and NF instances are both 1Gbps. Under this case, the approximation factors for update delay constraint, NF instance capacity and link capacity are all 3.2. In other words, the RBRU algorithm can achieve almost the constant approximation for the real-time SFC routing update problem in many situations.

6 Performance Evaluation

6.1 Performance Metrics and Benchmarks

6.1.1 *Performance Metrics*

We adopt the following nine performance metrics to evaluate the efficiency of the R^3-UA scheme: (1) the update delay [11]; (2) the maximum link utilization [11]; (3) the maximum NF instance utilization [30]; (4) the maximum number of NF instances assigned to a tenant [30]; (5) the maximum number of tenants served by an instance; (6) the control traffic overhead [31]; (7) the round-trip time (RTT) [33]; (8) the packet loss ratio [34]; (9) the flow completion time (FCT) [11].

During a simulation running, we use R^3-UA to perform robust NF assignment update and real-time SFC routing update. We use the time to complete the entire update process as the update delay. After the update process is complete, we compute the maximum link utilization, which is defined as $\max\{l(e)/C_e, e\in E\}$, where $l(e)$ is the traffic load on link e. Similar to link utilization, we also obtain the maximum NF utilization, which is defined as $\max\{l(m_i^s)/C_i^s, m_i^s\in M_s, s\in S\}$, where $l(m_i^s)$ is the load on NF instance m_i^s). Moreover, we measure the maximum number of NF instances assigned to a tenant and the maximum number of tenants served by each NF instance, which shows the degree of negative effect caused by malicious tenants and NF failures.

Since the control traffic overhead, RTT, packet loss ratio and FCT can not be obtained in the simulation, we measure these metrics through a small-scale testbed. Specifically, during a testbed run, we use the total traffic sent by the central controller to the data plane during the update process as the control traffic overhead. After the update process is completed, we measure RTT and packet loss rate by executing the command *ping*, and use iPerf3 to obtain the FCT information.

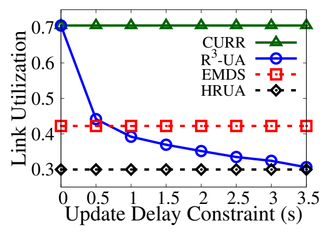
6.1.2 *Benchmarks*

This paper proposes two algorithms, NAUA and RBRU, to solve robust NF instance assignment update and real-time SFC routing update, respectively. These two algorithms constitute the R^3-UA scheme. We compare R^3-UA with the other three benchmarks. The first benchmark is a heuristic routing update algorithm (HRUA), modified form [16] [35]. Specifically, HRUA first selects the NF instances/links whose load is heavier than the average load, called heavy-loaded instances/links. Then, HURA traverses each heavy-loaded instance/link in turn, and updates the routes of requests on the current heavy-loaded NF instance/link until the load of the current heavy-loaded instance/ink is not greater than the average load. Different from the first benchmark, the second benchmark only updates the path of the elephant flows. This kind of method is modified from [11] [36]. Compared to updating routes of all traffic, only updating the routes of elephant flows can effectively reduce the update cost of the centralized controller. By this benchmark, the target routes of elephant flows are determined by the multi-commodity flow (MCF) algorithm, which is classic and widely used multi-path routing algorithm. Then, this benchmark uses a scheduling algorithm (e.g., Dionysus [17]) to update the routing scheme from the current configuration to the target one. In a word, this benchmark only updates elephant flows using the joint MCF and Dionysus and it is denoted as EMDS for simplicity. The last benchmark is the current network configuration (CURR), modified from the classic OSPF protocol and [37]. It does not update the routing of any traffic. This benchmark is used to show that routing update can improve network performance (i.e., reduce resource utilization).

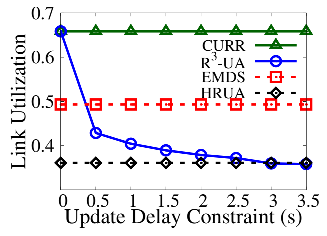
6.2 Simulation Evaluation

6.2.1 *Simulation Settings*

We conduct simulations on two typical topologies. The first topology is the Fat-Tree [38], which contains 80 switches (including 16 core switches, 32 aggregation switches, and 32 edge switches) and 128 servers. The second topology is VL2 [39], which contains 70 switches and 245 servers. The capacity of each link on both topologies is 1 Gbps. The authors in [40] present the throughput results for 9 different types of NFs on a single CPU. We can derive the processing capacity of each type of NF according to [40]. Since these two topologies do not provide NF information, we utilize the virtualization mechanism [41] to deploy nine types of NFs. We use the data traces of Alibaba Cluster [42] to generate requests. Note that, these data traces only contain the information of the traffic size of each request. To be more practical, we use the gravity model to generate the traffic matrix [43]. The SFC requirement of each tenant is randomly generated from the NF set [11]. We construct a layered graph and use the Yen's algorithm [44] to pre-calculate the candidate paths for all requests. According to [14], we set the time to install or modify a forwarding rule as $0.5ms$. The number of tenants is set to 300. Moreover, according to the size of the two topologies, we set the maximum number of instances traversed by the requests from each tenant to be 5, *i.e.*,

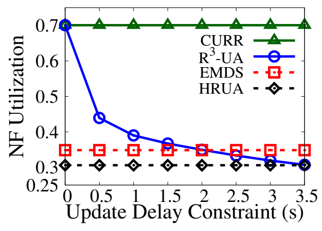


(a) Fat-Tree Topology

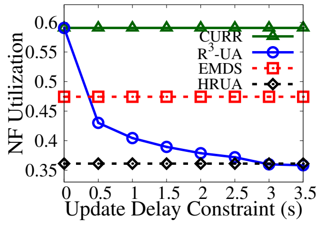


(b) VL2 Topology

Fig. 1: Link Utilization vs. Update Delay Constraint

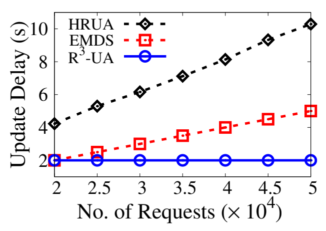


(a) Fat-Tree Topology

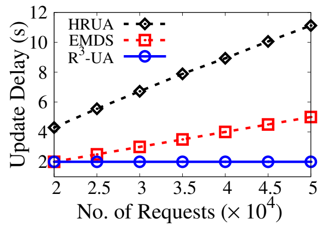


(b) VL2 Topology

Fig. 2: NF Instance Utilization vs. Update Delay Constraint

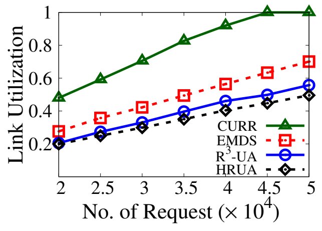


(a) Fat-Tree Topology

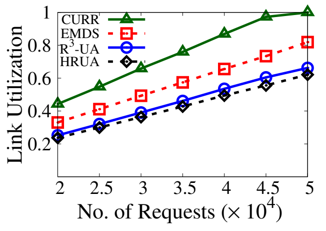


(b) VL2 Topology

Fig. 3: Update Delay vs. Number of Requests



(a) Fat-Tree Topology



(b) VL2 Topology

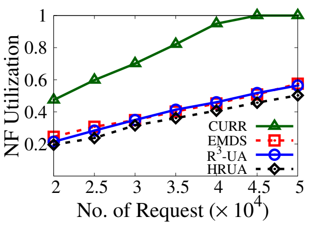
Fig. 4: Link Utilization vs. Number of Requests When $U = 2s$

$k = 5$, and the maximum number of tenants served by each NF instance to be 50, *i.e.*, $q = 50$, by default.

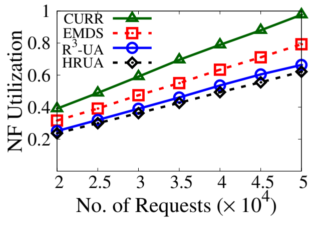
6.2.2 *Simulation Results*

We run five sets of simulations to verify the effectiveness of our algorithm. The first set of simulations mainly investigates how the update delay constraint affects link utilization and NF instance utilization on two topologies. When there are $30K$ requests in the cloud, we change the update delay constraint, and the load balance performance, i.e., the maximum resource utilization, is shown in Figs. 1 and 2. These figures show that link utilization and NF instance utilization are reduced when the update delay constraint becomes loose in R^3-UA. Since EMDS and HRUA do not consider the update delay, the change of update delay constraint does not affect link utilization and NF instance utilization of these two algorithms. Compared with HRUA, when R^3-UA achieves similar performance as HRUA, the update delay in R^3-UA is lower. For instance, when there are $30K$ requests on VL2 topology, HRUA needs about $7s$ to update all the forwarding rules by the right plot of Fig. 3. The right plots of Fig. 1 and 2 show that R^3-UA can achieve the similar route performance with a update delay of $2s$. Compared with EMDS, when the update delay constraint exceeds $2s$, R^3-UA not only has a lower update delay than that of EMDS, but also has better load balancing performance.

The second set of simulations studies the update delay by changing the number of requests in the cloud. Fig. 3 indicates that the update delay of HRUA and EMDS almost linearly increases with the number of requests in the network, while the update delay in R^3-UA is always maintaining at a low level. For example, when there are $40K$ requests on fat-tree topology, HRUA takes more than $8s$ to update the SFC routing, while R^3-UA only takes $2s$ to update the SFC routing, which means that R^3-UA can reduce the update delay more than 75% compared with HRUA. This result shows that the update delay in R^3-UA is less than that of the comparison algorithms. This is because R^3-UA takes the update delay constraint into account. With such a constraint, it will select those requests that can significantly improve the system performance, *i.e.*, reduce the maximum resource utilization, and update their SFC routing.



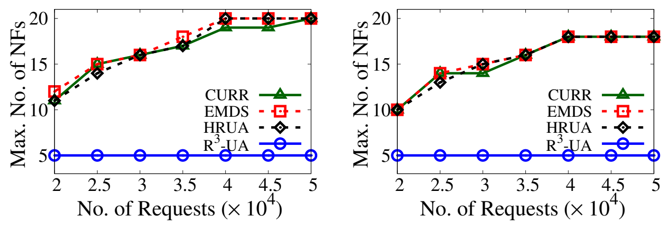
(a) Fat-Tree Topology



(b) VL2 Topology

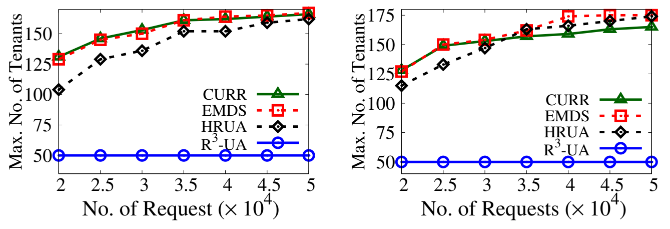
Fig. 5: NF Instance Utilization vs. Number of Requests When $U = 2s$

The third set of simulations shows how the number of requests affects the maximum resource utilization, including link utilization and NF instance utilization, when the update delay threshold U is set to $2s$. Figs. 4 and 5 show that the maximum resource utilization in R^3-UA is closer to that of HRUA as the number of requests increases, while the update delay in R^3-UA is much lower. Besides, compared with CURR and EMDS, R^3-UA can achieve lower resource utilization and update delay. For example, in the left plot of Fig. 4, when there are $50K$ requests on fat-tree topology, the gap between R^3-UA and HRUA is around 6%, and the update delay in R^3-UA and HRUA are $2s$ and $11s$, respectively. Also, R^3-UA can reduce link utilization by 14% and 44% compared with CURR and EMDS, respectively. This is because that CURR does not take update operation, and EMDS only updates the elephant using the MCF algorithm instead of selecting the requests that have the most significant impact on the system performance to update their SFC routing.



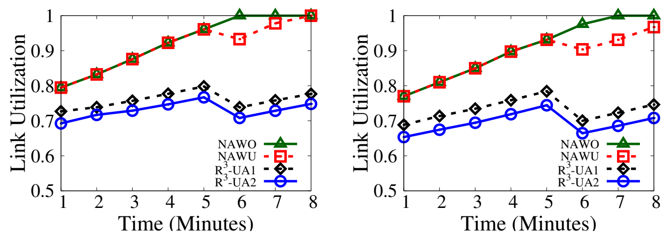
(a) Fat-Tree Topology (b) VL2 Topology

Fig. 6: Maximum Number of NFs vs. Number of Requests



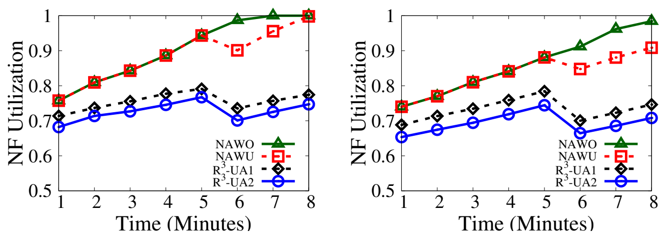
(a) Fat-Tree Topology (b) VL2 Topology

Fig. 7: Maximum Number of Tenants vs. Number of Requests



(a) Fat-Tree Topology (b) VL2 Topology

Fig. 8: Link Utilization vs. Time



(a) Fat-Tree Topology (b) VL2 Topology

Fig. 9: NF Instance Utilization vs. Time

The fourth set of simulations shows how the number of requests affects the maximum number of NF instances allocated to a tenant and the maximum number of tenants that an NF instance provides service to. Figs. 6 and 7 show that R^3-UA always performs better on these two metrics compared with other benchmarks. For example, in the left plot of Fig. 6, when there are $50K$ requests, the maximum number of NF instances allocated to a tenant through CURR, EMDS, HRUA and R^3-UA are all 20, while the value of R^3-UA is 5. In the left plot of Fig. 6, when there are $50K$ requests, an NF instance should provide services for at most 166, 167, 162 and 50 tenants, according to CURR, EMDS, HRUA and R^3-UA, respectively. Thus, R^3-UA performs better on these two metrics and has better system robustness. This is because R^3-UA considers the robustness constraints.

The fifth set of simulations illustrates the effectiveness of R^3-UA which consists of the robust NF instance assignment algorithm called NAUA and the real-time SFC routing update algorithm called RBRU. We design the first comparison algorithm called NAWO that determines the NF instance set for each tenant at the beginning of the system without NF instance assignment update and real-time SFC routing update. Each request in NAWO is routed through the shortest path by default. We also devise the second comparison algorithm called NAWU that updates NF instance assignment using the NAUA algorithm every 6 minutes, but it does not use RBRU to conduct the real-time SFC routing update. Then, the algorithms that run NAUA every 6 minutes and run RBRU every 1 minute with update delay constraint $2s$ and $4s$ as R^3-UA1 and R^3-UA2, respectively. As shown in Figs. 8 and 9, though it can reduce the maximum utilization of links and NF instances to a certain extent by running NAUA without RBRU, its performance is not as good as that of R^3-UA. Specifically, when the cloud runs for 6 minutes, the performance gap between NAWU and R^3-UA is at least 15%. This set of simulations shows the superiority of combined algorithms (*i.e.*, R^3-UA) of NAUA and RBRU.

From the above simulation results, we can make the following conclusions. First, in Figs. 1-2, with the increase of the update delay constraint, our algorithm can update more requests, and thus achieves better resource utilization. Second, Figs 3-5 show that, with the increase of request number, R^3-UA can achieve much lower update delay, and the resource utilization performance between R^3-UA and that of HRUA is within 6%. Third, Figs. 4-7 show that, our proposed algorithm can obtain much better system robustness, and its impact on resource utilization performance can be ignored. Fourth, Figs. 8-9 show the importance of combining NAUA and RBRU for R^3-UA compared with running only NAUA without RBRU.

6.3 Testbed Evaluation

6.3.1 *Testbed Implementation*

We implement CURR, EMDS, HRUA and R^3-UA on a small-scale topology Telstra, which contains eight switches [45]. Since the topology does not provide NF information, we utilize the virtualization mechanism [41] to deploy three types of NFs (*i.e.*, Firewall, NAT, Proxy) and place five instances of each NF type. Each NF instance and switch (implemented by Open vSwitches, version 2.9.2 [46]) runs on a single server with a Core i5-7500 processor and 8G of RAM. Besides, we use RYU [47] as the controller software running on another server with a Core i7-8700k and 32GB of RAM. We implement our tests with the data trace of Alibaba Cluster [48]. All requests need to go through *Firewall-NAT-Proxy*. There are 20 tenants in the testbed and each tenant generates 100 requests. Moreover, we set $k = 2, q = 8$ and 1500 requests by default. We adopt the command ping to measure the RTT and the packet loss ratio. Besides, we use the iPerf3 tool [49]

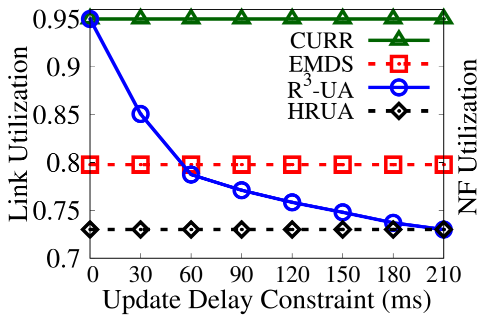


Fig. 10: Link Utilization vs. Update Delay Constraint When there are 2K requests

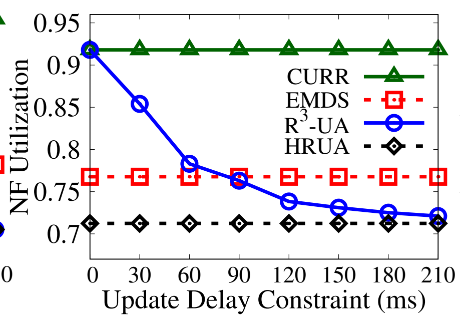


Fig. 11: NF Utilization vs. Update Delay Constraint When there are 2K requests

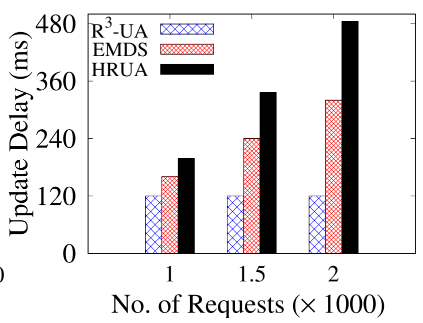


Fig. 12: Update Delay vs. No. of Requests When $U = 120ms$

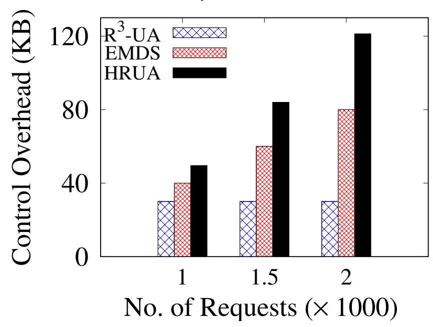


Fig. 13: Control Traffic Overhead vs. No. of Requests When $U = 120ms$

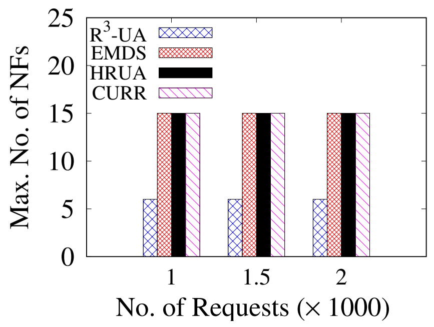


Fig. 14: Maximum Number of NFs vs. Number of Requests When $U = 120ms$

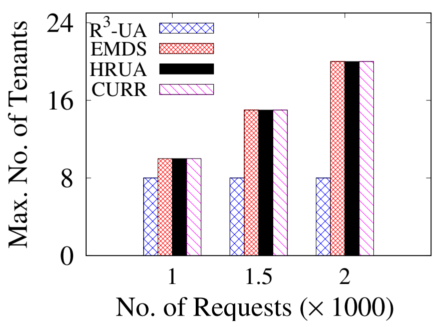


Fig. 15: Maximum Number of Tenants vs. Number of Requests When $U = 120ms$

to generate tenants' traffic and the vnStat tool [50] to monitor and collect the traffic information.

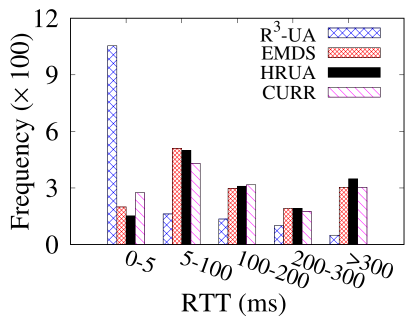
6.3.2 *Update Performance*

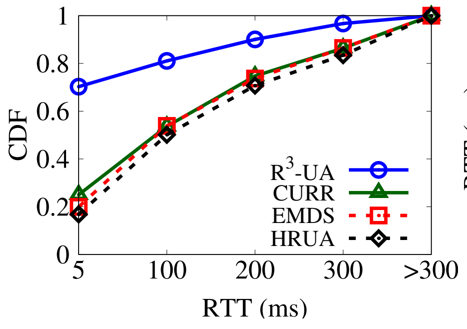
Figs. 10-15 show our experimental results of the update performance on the testbed. First, Figs. 10-11 mainly show how the route update delay constraint affects the link utilization and NF utilization on two topologies. Given $2K$ requests in the testbed, we change the update delay constraint, and measure the link utilization and NF utilization. It can be observed that the maximum link utilization and NF instance utilization in R^3-UA are reduced. Moreover, the resource utilization gap between R^3-UA and the comparison algorithms is also reduced when the update delay threshold becomes larger. Specifically, when update delay constraint is set to $120ms$, the resource utilization gap between R^3-UA and HRUA is less than 5%. Second, Figs. 12-13 evaluate the update delay and control traffic overhead by changing the number of requests with update delay constraint of $120ms$. It shows that R^3-UA can reduce the update delay and control traffic overhead by 60% and 59%, respectively, compared with HRUA. Third, Figs. 14-15 show the robustness performance of CURR, EMDS, HRUA and R^3-UA by changing the number of requests in the testbed. Fig. 14 shows that the requests of a tenant are processed by 15 NF instances in clouds through EMDS, HRUA and CURR, while only 8 NF instances (including Firewall, NAT, proxy) will be assigned to a tenant through R^3-UA. It means that R^3-UA can reduce the maximum number of NFs allocated to a tenant by 60% compared with the comparison algorithms when there are $2K$ requests. Fig. 15 shows that R^3-UA can also reduce the maximum number of tenants that an NF instance serves by 47% on average compared with HRUA, CURR and EMDS. Thus, R^3-UA can achieve better robustness performance when encountering NF failures and malicious tenants.

From these results, we can make the following conclusions. First, in Figs. 10-11, with the increase of the update delay constraint, R^3-UA can update more requests, and thus achieve better resource utilization. Second, in Figs. 12-13, with the increase number of requests, R^3-UA can reduce the update delay and control traffic overhead by 60% and 59%, respectively, compared with the comparison algorithms. Third, in Figs. 14-15, R^3-UA can reduce the number of affected NF instances by 60% if encountering malicious tenants, and decrease the number of affected tenants by around 47% if encountering NF failures.

6.3.3 *Performance Comparison with a Malicious Tenant*

In this section, we conduct the testbed under the scenario with a malicious tenant to fully explain that limiting the number of NF instances traversed by the requests from each tenant can enhance the system robustness. Specifically, we randomly choose a tenant as a malicious one, and the malicious tenant uses the hping tool [51] to launch a DDoS attack on its assigned NF instances. Then we measure RTT, packet loss ratio and FCT for performance comparison. To measure RTT more fairly, we test the RTT of each NF instance 100 times. Since the testbed contains 15 NF instances, there are totally 1500 tests. Figs. 16(a)-16(b) show that around 75% of the RTT results are less than $5ms$ through R^3-UA,


(a) Frequency vs. Round-trip Time


(b) CDF vs. Round-trip Time

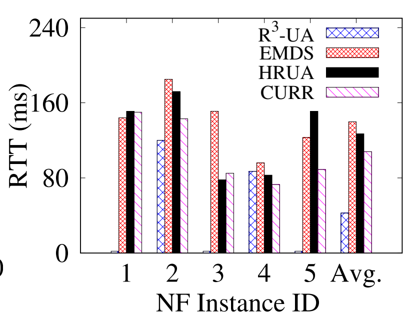
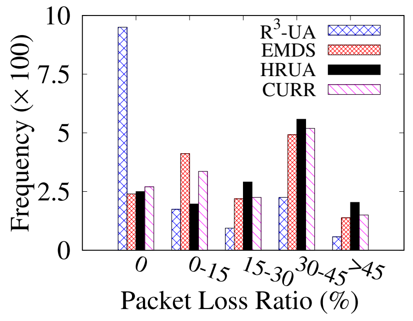
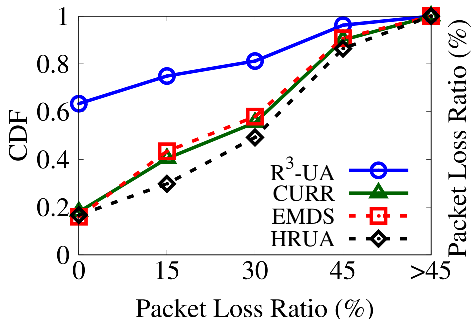

(c) Round-Trip Time of Instances

Fig. 16: The Round-trip Time Performance with a Mailicous Tenant


(a) Frequency vs. Packet Loss Ratio


(b) CDF vs. Packet Loss Ratio

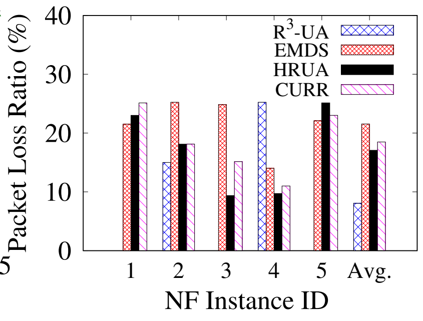
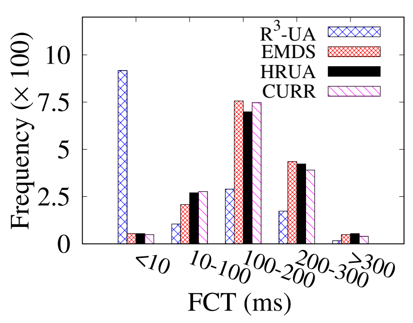
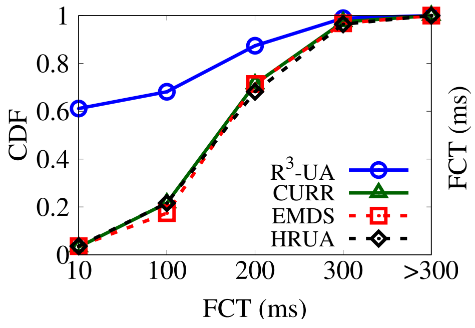

(c) Packet Loss Ratio of Instances

Fig. 17: The Packet Loss Ratio Performance with a Mailicous Tenant


(a) Frequency vs. Flow Completion Time


(b) CDF vs. Flow Completion Time

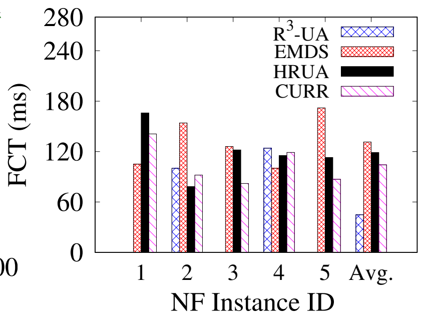

(c) Flow Completion Time of Instances

Fig. 18: The Flow Completion Time Performance with a Mailicous Tenant

while more than 46.2% of RTT results are greater than $100ms$ through the comparison algorithms. Taking the instances of NAT as an example, Fig. 16(c) shows the average RTT results of each NAT instance. We observe that the average RTT of R^3-UA is much lower than that of the comparison algorithms. The average RTTs of R^3-UA, EMDS, HRUA and CURR are $42.6ms$, $139.8ms$, $127ms$, and $108ms$, respectively. It means that R^3-UA can reduce the average RTT by 67.5%, 66.5% and 60.5% compared with EMDS, HRUA and CURR, respectively. Since the number of NF instances that the malicious tenant can access through R^3-UA is limited, R^3-UA can achieve lower RTT.

We investigate the packet loss ratio performance in Figs. 17(a)-17(c). Similarly, we also generate 1500 test cases to measure the packet loss ratio. From Figs. 17(a)-17(b), we find that R^3-UA can guarantee no packet loss in more than 60% of the tests, while no packet loss can be ensured in less than 20% of the tests by the comparison algorithms. Fig. 17(c) shows the average packet loss ratio results of each NAT instance. We observe that the average packet loss ratio of NAT instances through R^3-UA is much lower than that of the comparison algorithms. Specifically, the average packet loss ratios through R^3-UA, EMDS, HRUA and CURR are 8.05%, 21.6%, 17.07% and 18.48%, respectively.

We study the FCT performance given $1.5K$ requests in Figs. 18(a)-18(c). From Figs. 18(a)-18(b), we observe that most of the FCT results under R^3-UA are less than $10ms$, while around 70% of the FCT results through the comparison algorithms are greater than $100ms$. Fig. 18(c) illustrates the average FCT results of each NAT instance. It shows that the average FCT through R^3-UA is

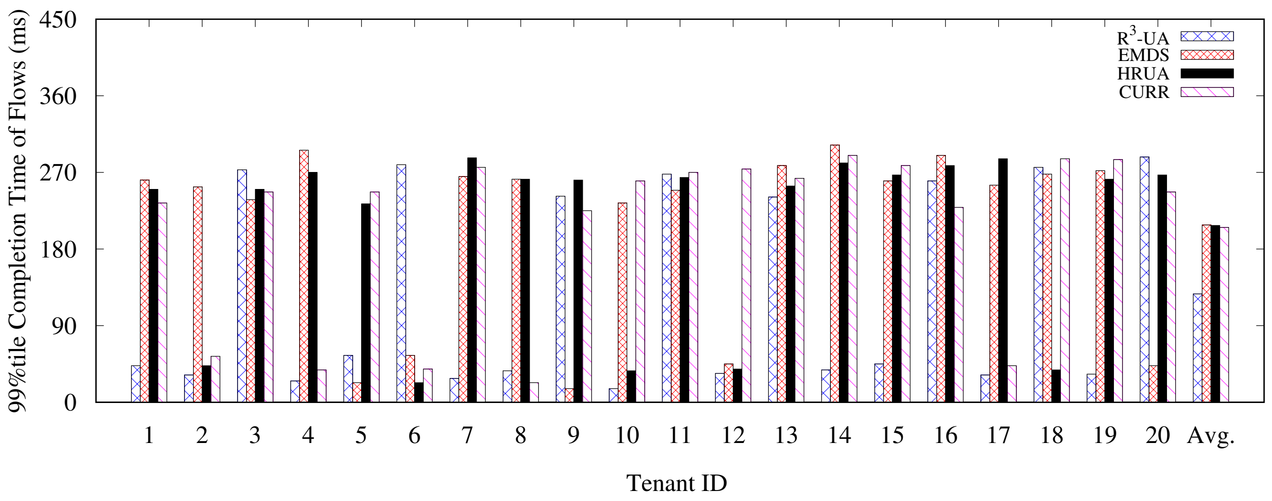


Fig. 19: 99%tile Completion Time of Flows of each Tenant with a Failed Instance

much lower than that of the comparison algorithms. R^3-UA can reduce the average flow completion time by 66.9%, 62.2% and 57% compared with EMDS, HRUA and CURR, respectively.

From the above experimental results, we see that the performances of RTT, packet loss ratio and FCT of R^3-UA are much better than those of the comparison algorithms. R^3-UA can reduce the average RTT, packet loss ratio and FCT by around 65%, 58%, 62%, respectively, compared with the comparison algorithms. These experimental results also show that limiting the number of NF instances traversed by the requests from each tenant can indeed enhance the system robustness when encountering malicious tenants.

6.3.4 Performance Comparison with a Failed Instance

In order to fully explain that limiting the number of tenants served by each NF instance can enhance the system robustness, this section focuses on the scenario with an NF instance failure. Since tenants expect their traffic to be handled as soon as possible, the 99%tile completion time of flows can reflect the QoS of tenants [52]. Thus, we mainly measure the 99%tile completion time of flows per tenant.

The experimental results are shown in Fig. 19. We randomly shut down an instance to simulate its failure. Since the flows processed by the failed instance need to be migrated to a working NF instance, the completion time of these flows will become much longer. Thus, the 99%tile flow completion time of tenants served by the failed instance is much longer than that of tenants served by other working instances. By Fig. 19, we know that the number of tenants affected by the failed NF instance are 8 and 15 corresponding to R^3-UA and the comparison algorithms. The average 99%tile completion times of flows through R^3-UA, EMDS, HRUA and CURR are $127.25ms$, $205.4ms$, $208.25ms$ and $207.6ms$, respectively. It means that R^3-UA can reduce the average 99%tile completion time of flows by 61.2% on average compared with the comparison algorithms.

From the experimental results in Fig. 19, we observe that the average 99%tile completion time of flows by R^3-UA is much shorter than that of the comparison algorithms. These experimental results mean that limiting the number of tenants served by each NF instance can indeed enhance the system robustness when encountering NF failures.

7 Conclusion

In this paper, we propose a robustness-aware real-time SFC routing update (R^3-UA) scheme, which takes into consideration the requirements of real-time routing update and system robustness in multi-tenant clouds. R^3-UA contains two phases: robust NF instance assignment update and real-time SFC routing update. Two algorithms with bounded approximation factors have been designed to solve the robust NF instance assignment update problem and the real-time SFC routing update problem, respectively. Both experimental results and simulation results show high efficiency of our proposed algorithms.

References

[1] H. Tu, G. Zhao, H. Xu, Y. Zhao, and Y. Zhai, "Robustness-aware real-time sfc routing update in multi-tenant clouds," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQoS)*. IEEE, 2021, pp. 1–6.

[2] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "Above the clouds: A berkeley view of cloud computing," *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, vol. 28, no. 13, p. 2009, 2009.

[3] D. Catteddu, "Cloud computing: benefits, risks and recommendations for information security," in *Iberic Web Application Security Conference*. Springer, 2009, pp. 17–17.

[4] A. Khajeh-Hosseini, I. Sommerville, and I. Sriram, "Research challenges for enterprise cloud computing," *arXiv preprint arXiv:1001.3257*, 2010.

[5] The amazon web service. [Online]. Available: http://www.aws.amazon.com/

[6] The google cloud platform. [Online]. Available: http://www.cloud.google.com/

[7] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker, "E2: a framework for nfv applications," in *Proceedings of the 25th Symposium on Operating Systems Principles*, 2015, pp. 121–136.

[8] B. Li, K. Tan, L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, P. Cheng, and E. Chen, "Clicknp: Highly flexible and high performance network processing with reconfigurable hardware," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 1–14.

[9] V. Sekar, S. Ratnasamy, M. K. Reiter, N. Egi, and G. Shi, "The middlebox manifesto: enabling innovation in middlebox deployment," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, 2011, pp. 1–6.

[10] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using sdn," in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, 2013, pp. 27–38.

[11] G. Zhao, H. Xu, J. Liu, C. Qian, J. Ge, and L. Huang, "Safe-me: Scalable and flexible middlebox policy enforcement with software defined networking," in *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. IEEE, 2019, pp. 1–11.

[12] C. Delimitrou and C. Kozyrakis, "Bolt: I know what you did last summer... in the cloud," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 599–613, 2017.

[13] R. Potharaju and N. Jain, "Demystifying the dark side of the middle: a field study of middlebox failures in datacenters," in *Proceedings of the 2013 conference on Internet measurement conference*, 2013, pp. 9–22.

[14] G. Zhao, L. Huang, Z. Yu, H. Xu, and P. Wang, "On the effect of flow table size and controller capacity on sdn network throughput," in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 2017.

[15] K. Kannan and S. Banerjee, "Compact tcam: Flow entry compaction in tcam for power aware sdn," in *International conference on distributed computing and networking*. Springer, 2013, pp. 439–444.

[16] B. Zhang, P. Zhang, Y. Zhao, Y. Wang, X. Luo, and Y. Jin, "Co-scaler: Cooperative scaling of software-defined nfv service function chain," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2016, pp. 33–38.

[17] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 539–550, 2014.

[18] K. Qu, W. Zhuang, Q. Ye, X. Shen, X. Li, and J. Rao, "Dynamic flow migration for embedded services in sdn/nfv-enabled 5g core networks," *IEEE Transactions on Communications*, vol. 68, no. 4, pp. 2394–2408, 2020.

[19] H. Xu, Z. Yu, X.-Y. Li, L. Huang, C. Qian, and T. Jung, "Joint route selection and update scheduling for low-latency update in sdns," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 3073–3087, 2017.

[20] J. Zheng, B. Li, C. Tian, K.-T. Foerster, S. Schmid, G. Chen, J. Wu, and R. Li, "Congestion-free rerouting of multiple flows in timed sdns," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 968–981, 2019.

[21] S. Achleitner, N. Bartolini, T. He, T. La Porta, and D. Z. Tootaghaj, "Fast network configuration in software defined networking," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1249–1263, 2018.

[22] V. Narayanan, Y. Huang, G. Tan, T. Jaeger, and A. Burtsev, "Lightweight kernel isolation with virtualization and vm functions," in *Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2020, pp. 157–171.

[23] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: elastic resource scaling for multi-tenant cloud systems," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, 2011, pp. 1–14.

[24] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, "Split/merge: System support for elastic execution in virtual middleboxes," in *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, 2013, pp. 227–240.

[25] H. Xu, X.-Y. Li, L. Huang, H. Deng, H. Huang, and H. Wang, "Incremental deployment and throughput maximization routing for a hybrid sdn," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, 2017.

[26] Y. Zhou, Y. Zhang, C. Ma, S. Chen, and O. O. Odegbile, "Generalized sketch families for network traffic measurement," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 3, pp. 1–34, 2019.

[27] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide measurements," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 561–575.

[28] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "On the effect of forwarding table size on sdn network utilization," in *IEEE INFOCOM 2014-IEEE conference on computer communications*. IEEE, 2014, pp. 1734–1742.

[29] A. Lebre, J. Pastor, A. Simonet, and M. Südholt, "Putting the next 500 vm placement algorithms to the acid test: The infrastructure provider viewpoint," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 1, pp. 204–217, 2018.

[30] H. Wang, H. Xu, H. Huang, M. Chen, and S. Chen, "Robust task offloading in dynamic edge computing," *IEEE Transactions on Mobile Computing*, 2021.

[31] J. Liu, H. Xu, G. Zhao, C. Qian, X. Fan, and L. Huang, "Incremental server deployment for scalable nfv-enabled networks," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2361–2370.

[32] X. Yang, H. Xu, L. Huang, G. Zhao, P. Xi, and C. Qiao, "Joint virtual switch deployment and routing for load balancing in sdns," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 397–410, 2018.

[33] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "Timely: Rtt-based congestion control for the datacenter," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 537–550, 2015.

[34] Z. Sun, R. Tao, N. Xiong, and X. Pan, "Cs-plm: Compressive sensing data gathering algorithm based on packet loss matching in sensor networks," *Wireless Communications and Mobile Computing*, vol. 2018, 2018.

[35] D. Z. Tootaghaj, F. Ahmed, P. Sharma, and M. Yannakakis, "Homa: An efficient topology and route management approach in sd-wan overlays," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2351–2360.

[36] G. Zhao, H. Xu, S. Chen, L. Huang, and P. Wang, "Joint optimization of flow table and group table for default paths in sdns," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1837–1850, 2018.

[37] S. Cai, F. Zhou, Z. Zhang, and A. Meddahi, "Disaster-resilient service function chain embedding based on multi-path routing," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2021, pp. 1–7.

[38] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM computer communication review*, vol. 38, no. 4, pp. 63–74, 2008.

[39] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, 2009, pp. 51–62.

[40] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, 2014, pp. 459–473.

[41] T. Lukovszki, M. Rost, and S. Schmid, "It's a match! near-optimal and incremental middlebox deployment," *ACM SIGCOMM Computer Communication Review*, vol. 46, no. 1, pp. 30–36, 2016.

[42] Alibaba cluster data trace. [Online]. Available: https://github.com/alibaba/clusterdata

[43] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, "Measuring isp topologies with rocketfuel," *IEEE/ACM Transactions on Networking*, vol. 12, 2016.

[44] J. Y. Yen, "Finding the k shortest loopless paths in a network," *management Science*, vol. 17, no. 11, pp. 712–716, 1971.

[45] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 133–145, 2002.

[46] Open vswitch. [Online]. Available: https://www.openvswitch.org/

[47] Ryu. [Online]. Available: https://ryu-sdn.org/

[48] The alibaba cloud. [Online]. Available: http://www.alibabacloud.com/

[49] iperf3. [Online]. Available: https://iperf.fr/

[50] vnstat. [Online]. Available: https://humdi.net/vnstat/

[51] hping. [Online]. Available: http://hping.org/

[52] P. Wang, G. Trimponias, H. Xu, and Y. Geng, "Luopan: Sampling-based load balancing in data center networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 1, pp. 133–145, 2018.

Huaqing Tu (Student Member, IEEE) is currently pursuing the Ph.D. degree in computer science at the University of Science and Technology of China. Her main research interests are software-defined networks and cloud computing.

Gongming Zhao (Member, IEEE) received the Ph.D. degree in computer software and theory from the University of Science and Technology of China in 2020. He is currently an Associate Professor with the University of Science and Technology of China. His current research interests include software-defined networks and cloud computing.

Hongli Xu (Member, IEEE) received the B.S. degree in computer science and the Ph.D. degree in computer software and theory from the University of Science and Technology of China, China, in 2002 and 2007, respectively. He is currently a Professor with the School of Computer Science and Technology, University of Science and Technology of China (USTC). He has published more than 100 articles in famous journals and conferences, including IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, International Conference on Computer Communications (INFOCOM), and International Conference on Network Protocols (ICNP). He has held more than 30 patents. His research interests include software defined networks, edge computing, and the Internet of Thing. He was awarded the Outstanding Youth Science Foundation of NSFC in 2018. He has won the best paper award or the best paper candidate in several famous conferences.

Yangming Zhao is a research professor at school of computer science and technology, University of Science and Technology of China. Before that, he was a research scientist with University at Buffalo. He received the B.S. degree in communication engineering and the Ph.D. degree in communication and information system from University of Electronic Science and Technology of China in 2008 and 2015, respectively. His research interests include network optimization, quantum networks, edge computing and machine learning.

Yutong Zhai received the B.S. degree in computer science from University of Science and Technology of China in 2017. He is currently a Ph.D. student in the School of Computer Science and Technology at the University of Science and Technology of China. His research interests are in the areas of software defined networking, Internet traffic measurement and edge computing.