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

Huaqing Tu[1,2] Gongming Zhao[1,2] Hongli Xu[1,2] Yangming Zhao[1,2,3] Yutong Zhai[1,2]
[1]School of Computer Science and Technology, University of Science and Technology of China, China
[2]Suzhou Institute for Advanced Study, University of Science and Technology of China, China
[3]Department of Computer Science and Engineering, University at Buffalo, The State University of New York

***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 and real-time SFC routing update. Two algorithms with bounded approximation ratios are proposed for these two phases, respectively. The 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**

I. Introduction

Cloud computing has transformed a large part of the internet industry and attracted more and more attention from academic and industry communities [1]–[3]. In multi-tenant clouds, the cloud vendors (*e.g.*, Amazon Web Services and Google Cloud Platform) 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 [4]. 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) [5]. 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* [6].

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* [6], [7]. 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 [8], [9]. On the one hand, malicious tenants may use vicious programs like Bolt [8] 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 that the median time of two consecutive failures is 7.5 hours for firewalls while 5.2 hours for load balancers [9]. For intrusion detection and prevention systems, the median time between two consecutive failures is about 20 minutes [9]. 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 [10], it takes at least 0.5 milliseconds (ms) for the central controller to update a routing rule on a commodity

978-0-7381-3207-5/21/$31.00 © 2021 IEEE

switch. Considering that there are usually millions of requests/flows injected into the cloud every minute [11], 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. 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 [6], [7], [12], [13] 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.* [13] 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, 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 the 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 called NAUA 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, called RBRU, which can complete the update process under a given delay constraint.
4) We evaluate the proposed algorithms in 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.

II. Preliminaries

A. 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 $\gamma \in \Gamma$.

B. 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 NFs 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 [6], [7], [12]. 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, 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.

C. Algorithm Workflow

Inspired by [7], 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 [14]. 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 [15]. 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 [16], R^3-UA pursues robustness-aware real-time routing update through two phases: *robust NF instance assignment update* (RNAU) and *real-time SFC routing update* (RTU). 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 II-B). 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 II-B).

III. Robust NF Assignment Update

A. RNAU Formulation

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 constant value denoting whether the NF instance m^s_i is assigned to tenant t or not before the update, $b^s_{t,i}$ 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 $s \in S$. 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.

B. 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 [16]. 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. 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. 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 $t \in T$.

IV. Real-Time SFC Routing Update

A. 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 [6], [7], [17]. 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 [10]), 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:

$$\min \quad \psi$$
$$S.t. \begin{cases} \sum_{\gamma \in \Gamma} \sum_{p \in \mathcal{P}_\gamma} z^p_\gamma \cdot t(\gamma, p, p^*) \cdot \tau \le U \\ \sum_{p \in \mathcal{P}_\gamma} z^p_\gamma = 1, & \forall \gamma \in \Gamma \\ \sum_{\gamma \in \Gamma} \sum_{p \in \mathcal{P}_\gamma : m^s_i \in p} z^p_\gamma \cdot f(\gamma) \le C^s_i \cdot \psi, & \forall s \in S, m^s_i \in M_s \\ \sum_{\gamma \in \Gamma} \sum_{p \in \mathcal{P}_\gamma : e \in p} z^p_\gamma \cdot f(\gamma) \le C_e \cdot \psi, & \forall e \in E \\ z^p_\gamma \in \{0, 1\}, & \forall \gamma \in \Gamma, p \in \mathcal{P}_\gamma \end{cases} \tag{2}$$

where binary variable z^p_γ 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.

B. Algorithm to Solve RTU

Since the variable z^p_γ 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. We first relax $z^p_\gamma \in [0, 1]$ in (2) 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 $\{\hat{z}^p_\gamma\}$ based on a rounding method. Assume the optimal solutions of the relaxed linear programming problem are $\{\widetilde{z}^p_\gamma\}$, then for each request $\gamma \in \Gamma$ and feasible path $p \in \mathcal{P}_\gamma$, we set $\hat{z}^p_\gamma = 1$ with probability $\widetilde{z}^p_\gamma$. That is, the request γ will be routed through path p with probability $\widetilde{z}^p_\gamma$.

V. Performance Evaluation

A. Performance Metrics and Benchmarks

1) Performance Metrics: We adopt the following five performance metrics to evaluate the efficiency of the R^3-UA scheme. The first metric is the update delay, which is the time to complete the entire SFC routing update process. The second metric is 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. The third metric is the maximum NF instance utilization. Similar to link utilization, if the load on each NF instance $m^s_i \in M_s$ is $l(m^s_i)$, then the maximum NF instance utilization can be defined as $\max\{l(m^s_i)/C^s_i, m^s_i \in M_s, s \in S\}$. The fourth metric is the maximum number of NF instances assigned to a tenant, which reflects the maximum negative effect by a single malicious tenant. The fifth metric is the maximum number of tenants processed by an NF instance, which reflects the maximum negative effect brought by the failure of a single NF instance.

2) Benchmarks: We compare R^3-UA with the other three benchmarks. The first one is the current network configuration denoted as CURR, modified from the OSPF protocol. It only chooses the shortest path for each request to fulfill the SFC requirement, and does not apply a routing update. This benchmark is used to show that routing update can improve network performance (*i.e.*, reduce resource utilization). For the second benchmark, the central controller typically uses the multi-commodity flow (MCF) algorithm to determine the target configuration based on the current workload, and uses a scheduling algorithm (*e.g.*, Dionysus [12]) to update the routing scheme from the current configuration to the target one. Since this solution may update too many requests, it will cause a large update delay. For a fair comparison, we use an improved version that only updates the routes of elephant requests [18]. The combined method that updates elephant requests with MCF and Dionysus algorithms is denoted as EMDS. The third benchmark is a heuristic routing update

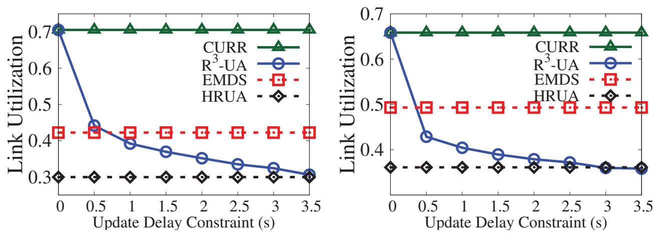


Fig. 1: Link Utilization vs. Update Delay Constraint *Left plot*: Topology (a); *right plot*: Topology (b).

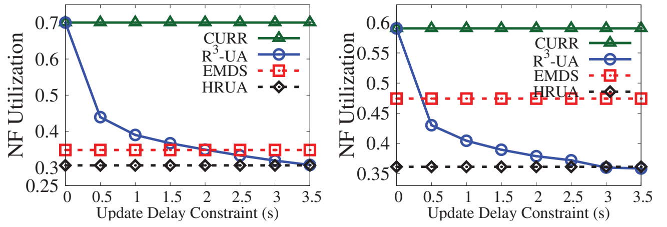


Fig. 2: NF Instance Utilization vs. Update Delay Constraint *Left plot*: Topology (a); *right plot*: Topology (b).

algorithm denoted as HRUA, adopted from [19]. HRUA first selects the NF instances whose load is heavier than the average load, called heavy-loaded instances. Then, HURA traverses each heavy-loaded instance in turn, and updates the routes of requests on the current heavy-loaded NF instance until the load of the current heavy-loaded instance is not greater than the average load.

B. Simulation Evaluation

1) Simulation Settings: We conduct simulations on two typical and practical topologies. The first topology is the Fat-Tree, denoted by (a), which contains 80 switches (including 16 core switches, 32 aggregation switches, and 32 edge switches) and 128 servers. The second topology is VL2, which contains 70 switches and 245 servers. The capacity of each link on both topologies is 1 Gbps. We use the data traces of Alibaba Cluster [20] to generate requests. The SFC requirement of each tenant is randomly generated from the NF set [7]. According to [10], 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 NF instances traversed by the requests from each tenant to be 5, *i.e.*, $k = 5$, and the maximum number of tenants served by each NF instance to be 50, *i.e.*, $q = 50$, by default.

2) Simulation Results: The first set of simulations mainly investigates how the update delay constraint affects link utilization and NF instance utilization. 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 3. 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

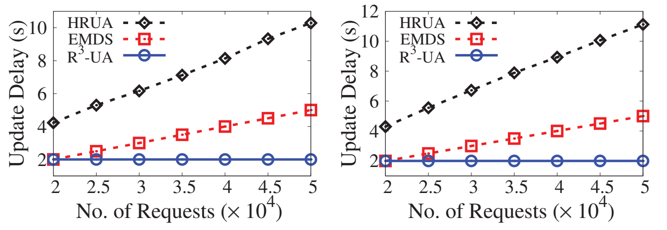


Fig. 3: Update Delay vs. No. of Requests *Left plot*: Topology (a); *right plot*: Topology (b).

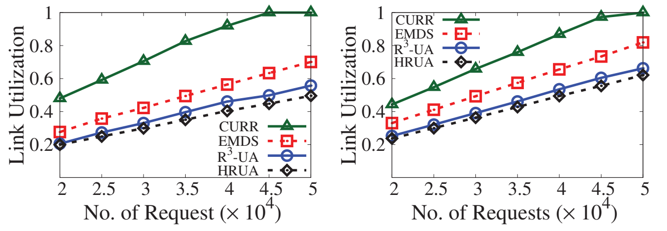


Fig. 4: Link Utilization vs. No. of Requests When $U = 2s$ *Left plot*: Topology (a); *right plot*: Topology (b).

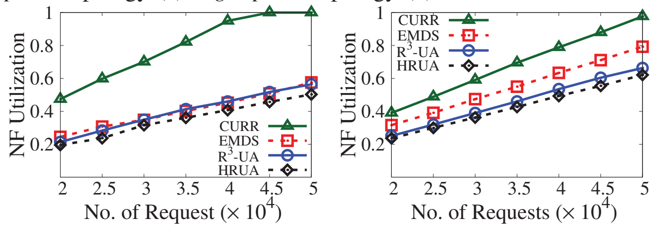
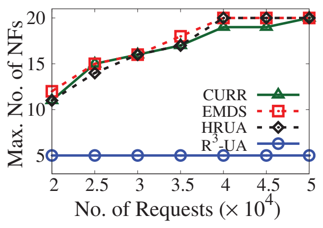


Fig. 5: NF Instance Utilization vs. No. of Requests When $U = 2s$ *Left plot*: Topology (a); *right plot*: Topology (b).

lower. For instance, when there are $30K$ requests on topology (b), 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 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 CUR-R 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 topology (a), 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.

The third set of simulations shows how the number of requests affects the maximum number of NF instances allo-



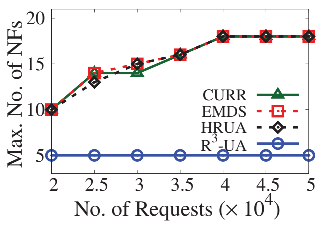
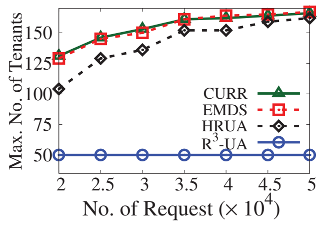


Fig. 6: Max. No. of NFs vs. No. of Requests *Left plot*: Topology (a); *right plot*: Topology (b).



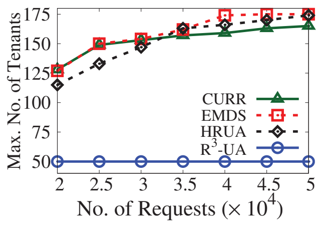


Fig. 7: Max. No. of Tenants vs. No. of Requests *Left plot*: Topology (a); *right plot*: Topology (b).

cated 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. Thus, R^3-UA performs better on these two metrics and has better system robustness.

From the above simulation results, we can make the following conclusions. First, our algorithm can update more requests, and thus achieves better resource utilization. Second, 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, our proposed algorithm can obtain much better system robustness, and its impact on resource utilization performance can be ignored.

VI. Conclusion

In this paper, we propose the 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.

Acknowledgement

The corresponding authors of this paper are Gongming Zhao and Hongli Xu. This article was supported in part by the National Science Foundation of China (NSFC) under Grants 61822210, 61936015, and U1709217; and in part by Anhui Initiative in Quantum Information Technologies under Grant AHY150300.

References

[1] 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.

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

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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, pp. 1–6.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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, pp. 1861–1875, 2017.

[17] 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.

[18] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat *et al.*, “Hedera: dynamic flow scheduling for data center networks.” in *Nsdi*, vol. 10, no. 8, 2010, pp. 89–92.

[19] B. Zhang, P. Zhang, Y. Zhao, Y. Wang, X. Luo, and Y. Jin, “Coscaler: 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.

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