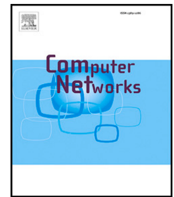
Computer Networks 215 (2022) 109212

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



RoNS: Robust network function services in clouds

Huaqing Tu, Gongming Zhao *, Hongli Xu *, Yangming Zhao, Yuhang Qiu, Liusheng Huang

School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, 230027, China

ARTICLE INFO

Keywords:
Network function
Failure
Fast recovery
Clouds

ABSTRACT

In multi-tenant clouds, the traffic of tenants (*e.g.*, enterprises) needs to be processed by network functions (NFs), for security and business logic issues. Due to potential hardware failures and software errors, NFs may break down. When encountering NF failures, we should consider two critical requirements for maintaining cloud robustness: *limited influence scope* and *fast failure recovery*. Without considering these two requirements, prior works based on deploying backup NF instances may result in large influence scope and long recovery time when a failure occurs. To bridge the gap, this paper investigates how to build robust network function services (RoNS) in multi-tenant clouds. Specifically, RoNS limits the number of tenants that each NF instance will serve so as to control the influence scope of an NF failure, and schedule requests with the help of agents designed in the data plane to achieve fast failure recovery. This is however a difficult undertaking. To solve this problem, RoNS takes a two-phase approach: *NF instance allocation* and *tenant request scheduling*. For NF instance allocation, we propose an efficient algorithm with bounded approximation factors based on the randomized rounding method. For tenant request scheduling, we present a primal–dual-based algorithm with a superior competitiveness ratio to solve it. We implement RoNS on a real testbed for experimental studies and use simulations for large-scale investigation. Both experiment results and simulation results show the superior performance of the proposed algorithms compared with other alternatives. For example, RoNS can cut down the number of affected tenants by 60%, and reduce recovery delay from 1170 ms to 316 ms on average, compared with existing failure recovery mechanisms based on deploying backup instances.

1. Introduction

With the development of virtualization technology [1], clouds can achieve elastic scaling of resources and reduce management complexity, thus more and more users migrate their businesses to clouds (*e.g.*, Amazon Web Services [2] and Google Cloud Platform [3]). As the scale of clouds grows, cloud vendors rely on a wide spectrum of network function (NF) services, such as intrusion detection systems (IDS), firewalls and load balancers, to ensure network security and enhance tenants' quality of service [4,5]. That is, cloud vendors provide various network services to tenants (*e.g.*, enterprises) with NF instances.

In practice, NF failures are common and usually incurred by some abnormal events such as connectivity errors, hardware faults, and overloads [6,7]. According to [7], the median time of two consecutive failures is 7.5 h for firewalls while 5.2 h for load balancers. In addition, the median time is only 20 min for intrusion detection and prevention systems [7]. With such common NF failures, the availability of NFs cannot always be guaranteed, which degrades the system robustness. When encountering NF failures, we should consider two critical requirements to maintain better cloud robustness. One is *limited influence scope* once a failure occurs. In multi-tenant clouds, multiple tenants may share the same NF instance to improve resource utilization. As an NF instance fails, it will stop providing services to all served requests, which will degrade the tenants' QoS. Thus, we expect that the failure of a single NF instance will not affect too many tenants' QoS. To this end, we should try to limit the number of tenants each NF instance will serve. The other requirement is *fast failure recovery*. After an NF instance fails, the traffic handled by the failed instance needs to be redirected to a working instance, which may cause a service outage and degrade the cloud robustness. Thus, when an NF failure occurs, we should restore service for the affected requests as soon as possible.

To deal with NF failures, several efficient solutions have been designed [8–11]. However, existing works often focus on rescheduling affected requests under resource constraints, but ignore the above robustness requirements in clouds. Specifically, these works often deploy backup instances first, and then reschedule the traffic from the failed NF instance to a backup instance. For example, the authors [10] deploy backup instances with the objective of minimizing the backup resource consumption while considering the heterogeneous resource demand of different NF instances. On the one hand, since they ignore the fact that one NF instance may serve lots of tenants, it may cause 25%–57%

* Corresponding authors.
E-mail addresses: gmzhao@ustc.edu.cn (G. Zhao), xuhongli@ustc.edu.cn (H. Xu).

https://doi.org/10.1016/j.comnet.2022.109212
Received 26 March 2022; Received in revised form 13 June 2022; Accepted 18 July 2022
Available online 22 July 2022
1389-1286/© 2022 Elsevier B.V. All rights reserved.

of tenants affected by the failure of an NF instance (see Section 5), resulting in *large influence scope once a failure occurs*. On the other hand, after an NF instance fails, the centralized control plane needs to select other backup NF instances for the traffic handled by the failed instance, then recalculates scheduling strategy, and finally deploys the corresponding rules in the data plane. That is, when a failure occurs, the above works need to be aware of failures, compute new rules and install them on switches [12], resulting in *long recovery time*. The experiment results in Section 5 show that the recovery time may be more than 1000 ms. Without considering these two robustness requirements, the above works may lead to poor tenants' QoS in facing NF failures. Thus, this paper focus on *building robust network function services in clouds* while satisfying these two requirements.

We should note that although there are some works trying to build robust services in the cloud, such as [13–16], they all have limitations. For example, the work [13] proposes the link failure recovery mechanism through the fast failover group table feature provided by Open Flow. Since the group table only supports monitoring the status of links and cannot detect whether there is an NF failure, their methods cannot handle NF failures. To deal with NF failures, the work [15] proposes to deploy backup NF instances using network function virtualization (NFV) technique. When an NF instance fails, the centralized control plane needs to detect the NF failure first, then reschedules the traffic from the failed instance to the backup one by deploying new forwarding rules in the data plane. However, this method requires the centralized control plane to participate in the recovery of NF failures, resulting in long recovery time. Moreover, they ignore the fact that an NF instance may serve a large number of tenants in their method, resulting in large influence scope once a failure occurs. Therefore, alternative solutions designed for limited influence scope and fast recovery from NF failures are in urgent need.

To overcome the shortcomings of the existing approaches, we propose RoNS to satisfy these two requirements simultaneously. The key ideas of RoNS are two-fold: (1) To control the influence scope of NF failures, RoNS limits the number of tenants served by each NF instance. In this way, we can control the number of affected tenants when encountering NF instance failures. (2) To achieve fast failure recovery, we design agents in the data plane. RoNS schedules requests and restores traffic processing when encountering NF failures. Specifically, we assign default and backup NF instances for each request when scheduling requests. Once the default NF instance fails, the traffic will be automatically and quickly forwarded to the backup NF instance pre-specified in the agents without the participation of the centralized control plane (see Section 4). Thus, we can achieve fast recovery when encountering NF failure events. The main contributions of this paper are as follows:

1. To deal with NF failures and enhance system robustness, this paper builds robust network function services (RoNS) in multi-tenant clouds. Due to traffic dynamics, RoNS takes a two-phase approach: NF instance allocation and tenant request scheduling.
2. For NF instance allocation, we formulate this problem as an integer linear programming, and propose a randomized-rounding-based algorithm called KNIT, which limits the number of NF instances that each tenant can access. We prove that the KNIT algorithm can achieve the approximation factor of $O(\log h)$, where h is the number of NF instances in a cloud, to limit the number of NF instances that each tenant can access.
3. For tenant request scheduling, we design a primal–dual-based request scheduling algorithm called PTRS. It first formulates the dual problem of tenant request scheduling, then assigns default and backup NF instances for each incoming request based on the dual problem. We prove that PTRS can achieve $[(1-\epsilon), O(\log h + \log(1/\epsilon))]$ competitiveness with $\epsilon \in (0, 1)$.
4. We implement RoNS on a real testbed for experimental studies and use simulations for large-scale investigation. The experiment results and simulation results show that RoNS can cut down the number of affected tenants by 60%, and reduce recovery delay from 1207 ms to 355 ms on average, compared with existing failure recovery mechanisms based on deploying backup instances.

The rest of this paper is organized as follows. Section 2 introduces the goals, intuition and workflow. In Section 3, we propose a rounding-based offline algorithm for the NF instance allocation sub-problem. Section 4 gives a primal–dual based online algorithm to solve the tenant request scheduling sub-problem. Section 6 presents the related works. The experiment and simulation results are given in Section 5. We conclude this paper in Section 7.

2. Goals and intuition

2.1. Design goals

This section introduces the goals of RoNS. Considering that tenants may generate traffic with various service requirements, these tenants' traffic should be scheduled to appropriate NF instances according to their needs. Since NF failures are common in clouds [6,7], in order to build robust network function services and ensure tenants' QoS, RoNS aims at achieving the following two design goals when scheduling tenants' traffic.

1. *Limited influence scope once a failure occurs.* In a multi-tenant cloud, an NF instance usually serves lots of tenants. When an NF instance fails, it stops processing traffic of all tenants it serves. Therefore, to improve cloud robustness, RoNS should limit the scope of influenced tenants once a failure occurs.
2. *Fast failure recovery after a failure occurs.* When an NF instance fails, we hope to quickly reschedule the traffic from the failed NF instance to other working ones. However, the previous failure recovery mechanisms require the participation of the centralized control plane, resulting in a long recovery delay. Thus, to reduce recovery delay, RoNS should achieve fast failure recovery after a failure occurs.

2.2. Our intuition

In Section 2.1, we summarize two critical goals of RoNS to deal with NF failures, *i.e.*, limited influence scope and fast failure recovery. Now, we introduce how RoNS achieves these two goals.

For the goal of limited influence scope, RoNS sets a constraint that the number of tenants served by an NF instance should not exceed a threshold p while scheduling tenants' traffic. p is a system-specific parameter and is determined according to the current system state by the administrator. This constraint limits the negative impact of a single NF instance failure. As a result, RoNS can control the influence scope of an NF failure event.

For the goal of fast failure recovery, RoNS designs agents (introduced in Section 4.1) in the data plane and assigns both the default and backup NF instances (in case the default one fails) to each request. The agents are responsible for the detection of the failed NF instances and the restoration of traffic processing. After the failure of a default NF instance is detected, the agents reschedule requests from the failed default NF instance to the backup ones with the help of agents without the participation of the centralized control plane, thereby achieving fast recovery. It should be noted that after failure recovery, each NF instance will still serve at most p tenants, and also meet the capacity constraint, even if some requests are rescheduled.

2.3. Problem statement

This section gives a more precise problem statement for building robust network function services in clouds. Specifically, considering that tenants may generate traffic with various service requirements on different compute nodes, we identify a request by three elements <tenant, compute node, service type>. We schedule each request to an appropriate NF instance according to its needs. The goal is to maximize the system throughput, which is defined as the total traffic amount of served requests. Since NF failures are common in clouds, we should consider the following two robustness requirements in request scheduling. (1) To limit the number of tenants affected by the failure of an NF instance, the number of tenants served by an NF instance should not exceed a threshold p. (2) To quickly reschedule the traffic processed by the failed NF instance to other working NF instances, both the default and backup NF instances should be assigned to each request. In this way, with the help of the agents designed in the data plane, we can achieve fast failure recovery without involving the centralized control plane.

2.4. System workflow

In Section 2.2, we present how to achieve the two goals of RoNS. On the one hand, to limit the influence scope once a failure occurs, RoNS considers which tenants each NF instance will serve (i.e., NF instance allocation). On the other hand, to achieve fast failure recovery after a failure occurs, RoNS designs agents in the data plane and assigns both the default and backup instances to each request (i.e., tenant request scheduling). Thus, RoNS needs to solve two sub-problems: NF instance allocation and tenant request scheduling.

One may think that it is natural to design an algorithm to solve the above two sub-problems jointly. However, since these two sub-problems have some inherent differences, it may not be feasible. Specifically, if we update the NF instance set of each tenant, many requests will be affected and rescheduled due to the new allocation result. Since NF instances need to back up the state information (*e.g.*, packet counters) [17] of the newly allocated tenant's requests, it will significantly increase update delay and control overhead for consistency preserving [17]. Thus, we should *reallocate NF instances for requests in a long-term interval*. Meanwhile, in order to adapt to traffic dynamics in clouds, it is necessary to *schedule requests in an online manner*. In other words, these two problems should be performed at different frequencies.

To this end, RoNS takes a two-phase approach. In the first phase, we propose a rounding-based offline algorithm for NF instance allocation (Section 3) to achieve the first goal. Based on the results of the first phase, in the second phase, we present a primal–dual online algorithm for tenant request scheduling (Section 4) to determine the default and backup NF instances of each request. Then we restore this information in the agents designed in the data plane. Once the agents detect NF failures, they reschedule the traffic from the failed NF instances to other working ones without the help of the centralized control plane, thereby reducing recovery delay and achieving the second goal. The design details of the proposed fast recovery mechanism based on agents will be given in Section 4.1.

3. NF instance allocation

In this section, we first introduce the multi-tenant cloud model, then formulate the NF instance allocation (NIT) problem and present a rounding-based algorithm to solve it. At last, we analyze the approximation performance of our proposed algorithm.

3.1. Multi-tenant cloud model

A typical multi-tenant cloud consists of four components: a network function (NF) set, a compute node set, a centralized control plane and a tenant set. The network function set contains m types of network services, denoted as $S = \{s_1, s_2, \ldots, s_m\}$. For clear problem formulation, we use $N_s = \{N_1^s, N_2^s, \ldots, N_{h_s}^s\}$ to represent the set of NF instances with type $s \in S$, where $h_s = |N_s|$ is the number of NF instances with type s. We also use $N = N_1 \cup N_2 \cup \cdots \cup N_s$ to denote the set of all the NF instances. The total number of NF instances in the cloud is denoted as h, *i.e.*, $h = \sum_{s \in S} h_s$. Due to the processing capacity constraints, each instance n can only provide services for a limited amount of requests (or tenants), and such processing capacity is denoted as C_n. A set of compute nodes provide computing resources to tenants via creating VMs. The centralized control plane is responsible for managing the whole cloud system, *e.g.*, traffic scheduling.

In multi-tenant clouds, tenants rent VMs and buy services from cloud vendors according to their needs. We use $T = \{t_1, t_2, \ldots, t_{|T|}\}$ to denote a set of tenants. Different tenants may generate traffic with various service requirements on different compute nodes.

3.2. Problem definition for NIT

The NIT problem focuses on how to select at most p tenants for each NF instance, thereby limiting the scope of impact when a failure occurs. Let b_t be the total traffic demand of tenant $t \in T$. We use the binary variable x_t^n to represent whether tenant $t \in T$ can be served by NF instance $n \in N$ or not, and use y_t^n to represent the traffic proportion of tenant t processed by NF instance n. The NIT problem can be formulated as follows:

$$\max \sum_{t \in T} \sum_{n \in N} y_t^n \cdot b_t$$

$$S.t. \begin{cases} \sum_{n \in N} y_t^n \leq 1, & \forall t \in T \\ \sum_{t \in T} x_t^n \leq p, & \forall n \in N \\ y_t^n \leq x_t^n, & \forall t \in T, n \in N \\ \sum_{t \in T} y_t^n \cdot b_t \leq C_n, & \forall n \in N \\ x_t^n \in \{0,1\}, & \forall t \in T, n \in N \\ y_t^n \in [0,1], & \forall t \in T, n \in N \end{cases} \tag{1}$$

The first set of inequalities shows the inherent property of the variable y_t^n, *i.e.*, for each tenant t, NF instance n can handle part of its traffic. The second set of inequalities means that each NF instance will process traffic from at most p tenants (referred to *tenant constraint* for short). This constraint can limit influence scope once a failure occurs. The third set of inequalities represents that NF instance n is allocated to tenant t if and only if there exists some traffic of tenant t is scheduled to NF instance n. The fourth set of inequalities represents the processing capacity constraint of NF instances. Our objective is to maximize the tenant traffic that can be served by NF instances through proper NF instance allocation while satisfying the above constraints.

3.3. Algorithm design for NIT

By using the traditional randomized rounding algorithm to solve NIT, we can obtain the approximation factors of $(O(\log h), O(\log h))$ [18]. That is, the tenant constraint (*i.e.*, the second set of inequations in Eq. (1)) and the NF processing capacity constraint (*i.e.*, the fourth set of inequations in Eq. (1)) are violated at most by a multiplicative factor $O(\log h)$. This section proposes a novel knapsack-based rounding algorithm for NIT, called KNIT. It can achieve approximation factors of $(1, O(\log h))$, and with proper assumption, the bound can be tightened to

(1, 2). It means that the proposed KNIT algorithm can strictly satisfy the tenant constraint and violate the NF processing capacity constraint at most by a multiplicative factor 2 in most practical scenarios. Moreover, the total tenant traffic that can be served by NF instances derived by KNIT is close to the optimal value with a high probability as illustrated in Theorem 5.

The KNIT algorithm consists of two steps. The first step constructs a linear program as a relaxation of NIT. More specifically, the NIT problem assumes that each NF instance can process traffic from at most p tenants. By relaxing this assumption, each NF instance can handle the traffic of any number of tenants. The relaxed linear program LP_1 is formulated as follows:

$$\max \sum_{t\in T}\sum_{n\in N} y_t^n \cdot b_t$$

$$S.t.\begin{cases}\sum_{n\in N} y_t^n \le 1, & \forall t\in T\\ \sum_{t\in T} y_t^n\cdot b_t \le C_n, & \forall n\in N\\ y_t^n\in[0,1], & \forall t\in T, n\in N\end{cases} \tag{2}$$

Because Eq. (2) is a linear program, we can use a linear program solver to solve it in polynomial, then obtain the optimal solution $\{\tilde{y}_t^n\}$.

In the second step, for each NF instance $n\in N$, KNIT puts each tenant $t\in T$ with $\tilde{y}_t^n \ge 0$ into a set T_n', then it will choose at most p tenants from T_n', and allocate a feasible traffic $\hat{y}_t^n$ for each chosen tenant. If $|T_n'|\le p$, we set $\hat{y}_t^n=\tilde{y}_t^n$. Otherwise, we divide the set T_n' into two sets, T_n and T_n''. We first introduce how to construct the set T_n. Initially, we set $p'=p$, and $C_n'=\sum_{t\in T}\tilde{y}_t^n\cdot b_t$. In each iteration, we add tenant $t\in T_n'$ with $\tilde{y}_t^n\cdot b_t\ge\frac{C_n'}{p'}$ to set T_n. After each iteration terminates, we update $C_n'=\sum_{t\in T}\tilde{y}_t^n\cdot b_t-\sum_{t\in T_n}\tilde{y}_t^n\cdot b_t$, and $p'=p-|T_n|$. Then, we determine the set T_n'' as $T_n''=T_n'-T_n$. For each tenant $t\in T_n''$, we define another variable $w_t=\frac{p'\cdot\tilde{y}_t^n\cdot b_t}{C_n'}=\frac{p'\cdot\tilde{y}_t^n\cdot b_t}{\sum_{t\in T_n''}\tilde{y}_t^n\cdot b_t}$. Obviously, $\sum_{t\in T_n''}w_t=p'$. We place each tenant with weight w_t into p' knapsacks so as to minimize the total weight of all tenants in each knapsack. Then, for each knapsack g, assume that it contains a set of tenants, denoted by T_n^g, and $e_g=\sum_{t\in T_n^g}w_t$. One tenant $t\in T_n^g$ will be chosen with probability $\frac{w_t}{e_g}$, and the traffic proportion of tenant t processed by NF instance n is set to $\hat{y}_t^n=\frac{e_g\cdot C_n'}{p'\cdot b_t}$. Otherwise, we set $\hat{y}_t^n=0$. The detailed description of KNIT is given in Alg. 1.

3.4. Performance analysis

We first give the following lemma to analyze the approximation performance.

Lemma 1. *The KNIT algorithm can guarantee that each NF instance will process traffic from no more than p tenants.*

Proof. If $|T_n'|\le p$, KNIT will assign these tenants to NF instance $n\in N$. In this case, the lemma holds. Otherwise, KNIT divides the set T_n' into two sets, T_n and T_n''. After degerming the set T_n, we update $C_n'=\sum_{t\in T}\tilde{y}_t^n\cdot b_t-\sum_{t\in T_n}\tilde{y}_t^n\cdot b_t$, and $p'=p-|T_n|$. Then, the algorithm will choose p' tenants from T_n'' for NF instance $n\in N$. After this, the number of tenants assigned to instance n is $|T_n|+p'=p$. Thus, the lemma still holds.

We introduce the following two famous lemmas of probability analysis before analyzing the approximation performance of the KNIT algorithm.

Theorem 2 (*Chernoff Bound [19]*). *Given n independent variables: $y_1, y_2, \dots, y_n$, where $\forall y_i\in[0,1]$. Let $\mu=\mathbb{E}[\sum_{i=1}^n y_i]$. Then, $\mathbf{Pr}[\sum_{i=1}^n y_i\ge(1+\epsilon)\mu]\le e^{\frac{-\epsilon^2\mu}{2+\epsilon}}$ and $\mathbf{Pr}[\sum_{i=1}^n y_i\le(1-\varsigma)\mu]\le e^{-\varsigma\mu/2}$, where ϵ and ς are arbitrary positive values.*

Algorithm 1 KNIT: Knapsack-Based Rounding for NIT

1: **Step 1: Solving the Relaxed Problem**
2: Construct a linear program LP_1 in Eq. (2)
3: Obtain the optimal solutions $\{\tilde{y}_t^n\}$
4: **Step 2: Allocating Tenants to Each NF Instance**
5: **for** each NF instance $n\in N$ **do**
6: Set $p'=p$, $C_n'=\sum_{t\in T}\tilde{y}_t^n\cdot b_t$, $T_n'=\{t|\tilde{y}_t^n\ge 0, t\in T\}$, and $T_n=\emptyset$
7: **if** $|T_n'|\le p'$ **then**
8: Put all tenants in T_n' to set T_n, and set $\hat{y}_t^n=\tilde{y}_t^n$
9: continue
10: **while** $\max\{\tilde{y}_t^n\cdot b_t\}\ge\frac{C_n'}{p'}$ **do**
11: Add tenant $t\in T_n'$ with $\tilde{y}_t^n\cdot b_t\ge\frac{C_n'}{p'}$ to set T_n, and set $\hat{y}_t^n=\tilde{y}_t^n$
12: $C_n'=\sum_{t\in T}\tilde{y}_t^n\cdot b_t-\sum_{t\in T_n}\tilde{y}_t^n\cdot b_t$, and $p'=p-|T_n|$
13: $T_n''=T_n'-T_n$
14: **for** each tenant $t\in T_n''$ **do**
15: $w_t=\frac{p'\cdot\tilde{y}_t^n\cdot b_t}{C_n'}=\frac{p'\cdot\tilde{y}_t^n\cdot b_t}{\sum_{t\in T_n''}\tilde{y}_t^n\cdot b_t}$
16: Put tenants in T_n'' into p' knapsacks with min-max weight
17: The tenant set in knapsack g is denoted by T_n^g
18: **for** each knapsack g **do**
19: $e_g=\sum_{t\in T_n^g}w_t$
20: Randomly choose a tenant $t\in T_n^g$ with probability $\frac{w_t}{e_g}$, and set $\hat{y}_t^n=\frac{e_g\cdot C_n'}{p'\cdot b_t}$

Theorem 3 (*Union Bound [20]*). *Given a countable set of n events: $A_1, A_2, \dots, A_n$, each event A_i happens with possibility $\mathbf{Pr}(A_i)$. Then, $\mathbf{Pr}(A_1\cup A_2\cup\cdots\cup A_n)\le\sum_{i=1}^n\mathbf{Pr}(A_i)$.*

Assume that the minimum capacity of all NF instances is denoted as C_{min}. We define a variable ϖ as follows:

$$\varpi=\min\left\{\frac{C_{min}}{b_t}, t\in T\right\} \tag{3}$$

Theorem 4. *The proposed KNIT algorithm guarantees that the total traffic on any NF instance $n\in N$ will not exceed its capacity C_n by a factor of $\frac{\log h}{\varpi}+3$. Moreover, in practice, the bound can be tighten to 2.*

Proof. The traffic load on NF instance n from tenant $t\in T$ is denoted by a random variable $\delta_{t,n}$. For each tenant, if $t\in T_n$, $\delta_{t,n}=\tilde{y}_n^t\cdot b_t$. If $t\in T_n''$, it follows that $\mathbb{E}[\delta_{t,n}]=\frac{w_t\cdot C_n'}{p'}=\tilde{y}_n^t\cdot b_t$. Thus, $\mathbb{E}[\delta_{t,n}]=\tilde{y}_n^t\cdot b_t, \forall t\in T$. By this definition, random variable δ are mutually independent. We can get the expected traffic load on each NF instance $n\in N$:

$$\mathbb{E}\left[\sum_{t\in T}\delta_{t,n}\right]=\sum_{t\in T}\mathbb{E}\left[\delta_{t,n}\right]=\sum_{t\in T}\tilde{y}_n^t\cdot b_t\le C_n \tag{4}$$

Combining Eq. (4) and the definition of ϖ in Eq. (3), we have:

$$\begin{cases}\frac{\delta_{t,n}\cdot\varpi}{C_n}\in[0,1]\\ \mathbb{E}\left[\sum_{t\in T}\frac{\delta_{t,n}\cdot\varpi}{C_n}\right]\le\varpi\end{cases} \tag{5}$$

By applying Theorem 2, assume that ρ is an arbitrary positive value, it follows:

$$\mathbf{Pr}\left[\sum_{t\in T}\frac{\delta_{t,n}\cdot\varpi}{C_n}\ge(1+\rho)\varpi\right]\le e^{\frac{-\rho^2\varpi}{2+\rho}} \tag{6}$$

Now, we assume that:

$$\mathbf{Pr}\left[\sum_{t\in T}\frac{\delta_{t,n}}{C_n}\ge(1+\rho)\right]\le e^{\frac{-\rho^2\varpi}{2+\rho}}\le\frac{1}{h} \tag{7}$$

where h is the number of NF instances in clouds. We know that $\frac{1}{h} \to 0$ when the network grows. By solving Eq. (7), we have the following results:

$$\rho \geq \frac{\log h + \sqrt{\log^2 h + 8\varpi \log h}}{2\varpi} \Rightarrow \rho \geq \frac{\log h}{\varpi} + 2 \tag{8}$$

In most practical scenarios, according to the definition of ϖ, we assume that $\varpi \geq 3\log h$. Thus, we have:

$$\rho \geq \frac{\log h + \sqrt{(\log h - 2\varpi)^2 - 4\varpi^2 + 12\varpi \log h}}{2\varpi} \Rightarrow \rho \geq 1 \tag{9}$$

Thus, the approximation factor for NF instance capacity constraints is $\rho+1 = \frac{\log h}{\varpi}+3$. Under proper assumption (*i.e.*, $\varpi \geq 3\log h$), the bound can be tighten to $\rho + 1 = 2$.

Theorem 5. *Suppose O_{LP} is the optimal objective value to the relaxed version of Eq. (2), while O_R is the objective value of Eq. (2) associated with the KNIT algorithm. We have $\mathbf{Pr}[O_R \leq (1-\varsigma)O_{LP}] \leq e^{-\varsigma O_{LP}/2}$, which means the objective value O_R derived by KNIT is close to the optimal value O_{LP} with a high probability.*

Proof. From the analysis in Theorem 4, we have $\mathbb{E}[\hat{y}_t^n \cdot b_t] = \tilde{y}_t^n \cdot b_t$. Accordingly,

$$\mathbb{E}[O_R] = \sum_{t\in T}\sum_{n\in N} \hat{y}_n^t \cdot b_t = \sum_{t\in T}\sum_{n\in N} \tilde{y}_n^t \cdot b_t = O_{LP} \tag{10}$$

Then, we have $e^{-\varsigma O_R/2} = e^{-\varsigma O_{LP}/2}$. Based on Theorem 2, we have:

$$\mathbf{Pr}[O_R \leq (1-\varsigma)\mathbb{E}[O_R]] \leq e^{-\varsigma \mathbb{E}[O_R]/2} \tag{11}$$

Combining above discussions, we conclude that:

$$\mathbf{Pr}[O_R \leq (1-\varsigma)O_{LP}] \leq e^{-\varsigma O_{LP}/2} \tag{12}$$

Thus, the objective value O_R derived by KNIT is close to the optimal value O_{LP} with a high probability.

Approximation Factors: According to the above theorems, the tenant constraint can be strictly satisfied, and the processing capacity of NF instances will hardly be violated by a factor of $\frac{\log h}{\varpi}+3$. It means that KNIT can achieve the optimal solution, violating the NF instance capacity by at most a factor $\frac{\log h}{\varpi}+3$, while strictly satisfying the tenant constraint. Thus, we can conclude that KNIT can achieve the bi-criteria approximation of $(1, \frac{\log h}{\varpi}+3)$. Under some proper assumptions (*e.g.*, $\varpi \geq 3\log h$), the bound can be tightened to (1, 2). It means that KNIT can achieve the optimal solution and strictly satisfy the tenant constraint, violating the NF instance capacity constraints by at most a factor 2. It should be noted that the traffic size of each request will be scaled to a specific value by using the traffic controlling method. Under this case, we can avoid the network congestion.

4. Tenant request scheduling

To achieve the second goal (introduced in Section 2.1), we need to design a fast recovery mechanism in the data plane without the participation of the centralized control plane, and assign both the default and backup NF instances to each request (tenant request scheduling) in case the default one fails. In the following, we first introduce the design of fast recovery, then define the problem of tenant request scheduling (TRS) and present an online algorithm called PTRS to solve it. Finally, we analyze its approximation performance and introduce how to deal with the failure of multiple NF instances.

4.1. Design of fast recovery

In the previous works, their failure recovery methods are tightly coupled with the central control plane. Specifically, the centralized control plane needs to continuously monitor the status of NF instances to determine whether there is an NF failure. Then the centralized control plane selects another working NF instance and installs new forwarding rules on Open vSwitches (OVSes) [21] to reroute traffic to the selected NF instance when an NF failure is detected. However, the failure recovery process in this method requires the participation of the centralized control plane, resulting in a long recovery time. In the OpenFlow protocol [22], the group table with "fast failover" type supports recovery from link failures without notifying the centralized control plane by pre-designating the backup output ports on switches [23,24]. But in fact, since the fast failover group table can only monitor the status of adjacent ports, it cannot be aware of NF failures. Therefore, it does not support the recovery of NF failures.

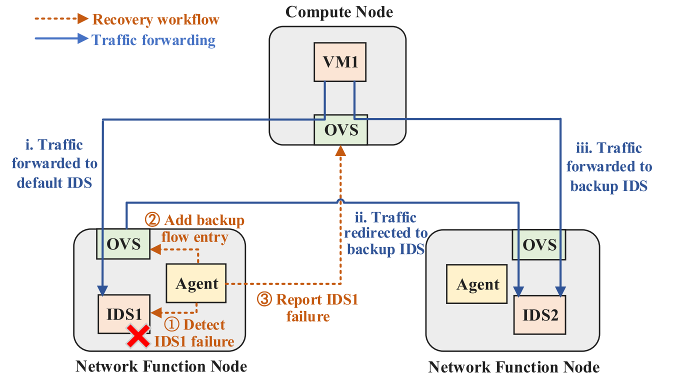


Fig. 1. Illustration of the fast recovery design. There is one compute node and two NF nodes. A virtual machine (VM1) is launched on the compute node. Two IDS instances (IDS1, IDS2) are running on two NF nodes, respectively. IDS1 and IDS2 are designated by the centralized control plane as the default NF instance and backup NF instance for VM1's traffic, respectively. The dotted line with arrow indicates the recovery workflow. The solid line with arrow indicates the traffic forwarding.

Motivated by these observations and fast failover group table in OpenFlow, we design agents on servers (referred to NF nodes in the following) where NF instances are running. The agent on an NF node is mainly responsible for: (1) detecting whether the NF instances started in the NF node are failed or not; (2) adding a higher priority flow entry (backup flow entry for short) on OVS for redirecting traffic on the failed NF instance to the backup NF instance; (3) reporting the NF failure event and backup instance information (*e.g.*, IP, MAC, *etc.*) to compute nodes which have traffic processed by the failed NF instance. When a compute node receives the NF failure message and backup instance information from the agents on the NF nodes, it updates the flow entry on OVS, so that the subsequent traffic will be sent to the backup NF instance directly. It should be noted that when scheduling requests, each request's backup NF instance information has been stored on the agent in advance. Since NF failure detection and forwarding rule update are both implemented by the agents on NF nodes without the participation of the centralized control plane, the failure recovery time can be greatly reduced, thereby achieving fast recovery. After the agent finishes the restoration of traffic processing in the data plane, it reports the information of the failed NF instance to the control plane, so that the control plane can obtain the global state of the data plane.

Now we give an example in Fig. 1 to further illustrate the proposed fast recovery design in detail. For simplicity, we assume that a tenant's virtual machine (VM1) is launched on a compute node, and VM1's traffic needs to be processed by intrusion detection system (IDS) for security. Two IDS instances (IDS1, IDS2) are running on two NF nodes, respectively. The centralized control plane assigns IDS1 and IDS2 as the default NF instance and backup NF instance of VM1's traffic, respectively. The centralized control plane schedules VM1's traffic to the default instance IDS1 by configuring the rules on the OVS of the compute node, stores IDS2's information in the agent of the NF node where IDS1 is located. Once the agent detects that IDS1 is failed,

it immediately adds a backup flow entry based on the stored IDS2's information to redirect the traffic to IDS2. Then the agent reports the IDS1 failure event and the backup instance IDS2's information to the compute node. When the compute node receives these messages, it updates the flow entry, so that the subsequent traffic will be forwarded to the backup instance IDS2 directly. From the proposed fast recovery procedure, we can see that neither the failure detection of IDS1 nor the recovery of traffic processing involves the centralized control plane.

4.2. Problem definition for TRS

In practice, a cloud usually consists of multiple regions/subnets [25]. Based on efficiency considerations, service requests are usually scheduled within a region [25]. Assume that the number of NF instances of the same type in a region is 100 and the failure probability of an NF instance is 0.1% [8]. Under this case, the failure probability of single instance and multiple instances is 9.06% and 0.47%, respectively. It means that single instance failure is more common than multiple instance failures in a region. Thus, this section only selects one default instance and one backup instance for each request to cope with the single instance failure event. The scenario of multi-instance failure will be discussed in Section 4.5.

We can obtain the set of tenants allocated to each NF instance using the KNIT algorithm. Based on these results, we first derive a candidate scheduling scheme set for each tenant $t \in T$ (denoted as Ψ_t). Each scheduling scheme includes a default NF instant and a backup one. Considering that the traffic generated by different tenants may have various service requirements on different compute nodes, we distinguish a request by three elements <tenant, compute node, service type>. Let Γ_t be the request set of tenant t. For each request $\gamma \in \Gamma_t$ of tenant t, we select a scheme from Ψ_t for request γ, such that both the capacity constraints and the limited influence scope requirement are still satisfied on all NF instances, even after failure recovery. We define a binary variable $z^{\psi}_{t,\gamma}$ to represent whether request $\gamma \in \Gamma_t$ of tenant $t \in T$ will be scheduled with scheme $\psi \in \Psi_t$. Let the binary constant $I_1(\psi, n)$ represent whether the default instance in ψ is the same as $n \in N$ ($I_1(\psi, n) = 1$) or not ($I_1(\psi, n) = 0$), and $I_2(\psi, n)$ represent whether the backup instance in ψ is the same as n ($I_2(\psi, n) = 1$) or not ($I_2(\psi, n) = 0$). Now, we introduce how the overall load on NF instance n_j can be obtained once NF instance n_i fails. The traffic size of request γ is recorded as b_γ. Without failure, the load on NF instance n_j is:

$$\sum_{t\in T}\sum_{\gamma\in\Gamma_t}\sum_{\psi\in\Psi_t} z^{\psi}_{t,\gamma}\cdot I_1(\psi, n_j)\cdot b_\gamma \tag{13}$$

When NF instance n_i fails, the incremental load on NF instance n_j is:

$$\sum_{t\in T}\sum_{\gamma\in\Gamma_t}\sum_{\psi\in\Psi_t} z^{\psi}_{t,\gamma}\cdot I_1(\psi, n_i)\cdot I_2(\psi, n_j)\cdot b_\gamma \tag{14}$$

Thus, when NF instance n_i fails, the overall load on NF instance n_j is:

$$\sum_{t\in T}\sum_{\gamma\in\Gamma_t}\sum_{\psi\in\Psi_t} z^{\psi}_{t,\gamma}\cdot \theta^{\psi}_{i,j}\cdot b_\gamma \tag{15}$$

where $\theta^{\psi}_{i,j} = I_1(\psi, n_j) + I_1(\psi, n_i)\cdot I_2(\psi, n_j)$. Accordingly, the TRS problem can be formulated as follows:

$$\max \sum_{t\in T}\sum_{\gamma\in\Gamma_t}\sum_{\psi\in\Psi_t} z^{\psi}_{t,\gamma}\cdot b_\gamma$$

$$S.t.\begin{cases}\sum_{\psi\in\Psi_t} z^{\psi}_{t,\gamma}\le 1, & \forall t\in T, \gamma\in\Gamma_t\\ \sum_{t\in T}\sum_{\gamma\in\Gamma_t}\sum_{\psi\in\Psi_t} z^{\psi}_{t,\gamma}\cdot\theta^{\psi}_{i,j}\cdot b_\gamma\le C^j_n, & \forall n_i, n_j, n_i\ne n_j\\ z^{\psi}_{t,\gamma}\in\{0,1\}, & \forall t,\gamma,\psi\end{cases} \tag{16}$$

The first set of inequalities means at most one scheme is selected for request γ. The second set of inequalities denotes that the load on instance n_j will not exceed its capacity, even if some requests served by failed instance n_i are migrated to n_j, with $\forall n_i, n_j \in N$. The objective goal of TRS is to maximize throughput, *i.e.*, total traffic amount of requests successfully served by NF instances.

One may think that the traffic proportion of tenant t handled by NF instance n needs to be equal to the solution of variable y^n_t in the first phase. However, this is not the case. When solving the NF instance allocation problem, the traffic demand b_t of tenant t is obtained through long-term observation and statistical collection. Then we use the KNIT algorithm to determine the tenant set served by each NF instance through the solution of variable x^n_t and obtain the traffic proportion of tenant t handled by NF instance n through the solution of variable y^n_t. Based on the result of x^n_t, the second phase determines the scheduling scheme for each incoming request in an online manner. Thus, the traffic information input in the first phase is based on long-term offline statistics, and that of the second phase is based on online incoming requests. That is, the traffic input in the first phase is totally different from that in the second phase. Therefore, the traffic proportion of tenant t handled by NF instance n in the second phase does not need to be equal to the solution of variable y^n_t in the first phase.

Lemma 6. *The TRS problem is NP-hard.*

Proof. Consider a special case of TRS, in which there are only two NF instances with equal processing capacity. Each request will be assigned with a default NF instance and a backup NF instance in case the default NF instance fails and there is no difference to distinguish the default NF instance and the backup NF instance. Therefore, the TRS problem with the objective to maximize the overall traffic amount of requests successfully served by NF instances becomes the knapsack problem with the objective to maximize the total value of all accepted items. Specifically, the NF instance and requests can be regarded as the knapsack and items, respectively. The processing capacity constraint on the NF instance is like the capacity constraint of the knapsack. The throughput of TRS, i.e., the total traffic amount of all accepted requests, can be regarded as the total value of all accepted items in the knapsack problem. Since the special case of the TRS problem is substantially the knapsack problem, which is a well-know NP-hard problem, our TRS problem is NP-hard as well.

4.3. Algorithm design for TRS

To solve the problem in Eq. (16), we design an online algorithm called PTRS based on a primal–dual method. We first construct the dual problem for the linear relaxation of Eq. (16). Let $\alpha_{t,\gamma}$ and β_{n_i,n_j} be the dual variables of the first and second sets of inequalities, respectively, and all dual variables are non-negative. The dual problem can be formulated as:

$$\min \sum_{t\in T}\sum_{\gamma\in\Gamma_t}\alpha_{t,\gamma} + \sum_{n_i,n_j,n_i\ne n_j}\beta_{n_i,n_j}\cdot C^j_n$$

$$S.t.\begin{cases}\alpha_{t,\gamma} + \sum_{n_i,n_j,n_i\ne n_j}\theta^{\psi}_{i,j}\cdot b_\gamma\cdot\beta_{n_i,n_j}\ge b_\gamma, & \forall t,\gamma,\psi\\ \alpha_{t,\gamma}\ge 0, \beta_{n_i,n_j}\ge 0, & \forall t,\gamma,n_i,n_j,n_i\ne n_j\end{cases} \tag{17}$$

Then, the first constraint of Eq. (17) can be rewritten as:

$$\alpha_{t,\gamma}\ge b_\gamma\left(1-\sum_{n_i,n_j,n_i\ne n_j}\theta^{\psi}_{i,j}\cdot\beta_{n_i,n_j}\right), \forall t,\gamma,\psi \tag{18}$$

The first step of the PTRS is to initialize corresponding constants and all the dual variables. Based on the first inequality set in Eq. (17), we define constant B^* as the maximum usage of resource over all scheduling schemes for each request.

$$B^* = \max_{t,\psi,n_i,n_j,n_i\ne n_j}\{\theta^{\psi}_{i,j}\} \tag{19}$$

The second step of PTRS is to select a scheduling scheme for each incoming request. Once a new request comes, PTRS computes the price of each candidate scheme. Suppose that this new request belongs to tenant t, the price of scheme $\psi \in \Psi_t$ can be defined as:

$$K_\psi = \sum_{n_i,n_j,n_i \neq n_j} \theta^{\psi}_{i,j} \cdot \beta_{n_i,n_j} \tag{20}$$

The PTRS algorithm figures out the scheme with lowest price, denoted as K_{ψ^*}. If $K_{\psi^*} > 1$, it will violate the dual program's constraints (be proved in Lemma 7). Then, PTRS will reject this request and set the corresponding dual variable $\alpha_{t,\gamma}$ to 0. Otherwise, PTRS will accept the request, and the dual variables will be updated as follows:

$$\begin{cases} \alpha_{t,\gamma} \leftarrow b_\gamma (1 - K_{\psi^*}) \\ \beta_{n_i,n_j} \leftarrow \beta_{n_i,n_j}(1 + \frac{\theta^{\psi}_{i,j} \cdot b_\gamma}{C^j_n}) + \frac{\theta^{\psi}_{i,j} \cdot b_\gamma}{h^2 \cdot \varphi \cdot C^j_n}, \quad \forall n_i, n_j \end{cases} \tag{21}$$

Now, we give the formal description of PTRS in Alg. 2.

Algorithm 2 PTRS: Primal–Dual Algorithm for TRS

1: **Step 1: Algorithm Initialization**
2: $\varphi = B^*/\epsilon$, where $\epsilon \in (0, 1)$
3: Initialization all the dual variables:
4: $\alpha(t, \gamma) \leftarrow 0, \forall t \in T, \gamma \in \Gamma_t$
5: $\beta_{n_i,n_j} \leftarrow 0, \forall n_i, n_j \in N, n_i \neq n_j$
6: **Step 2: On arriving of a request**
7: **for** each arrival request γ **do**
8: Denote the tenant that request γ belongs to as t
9: Calculate the price K_ψ of candidate scheme $\psi \in \Psi_t$ according to Eq. (20)
10: $\psi^* \leftarrow \arg\min_\psi K_\psi$
11: **if** $K_{\psi^*} < 1$ **then**
12: Assign default and backup instances for request γ according to scheme ψ^*
13: $\alpha_{t,\gamma} \leftarrow b_\gamma(1 - K_{\psi^*})$
14: $\forall n_i, n_j : \beta_{n_i,n_j} \leftarrow \beta_{n_i,n_j}(1 + \frac{\theta^{\psi}_{i,j} \cdot b_\gamma}{C^j_n}) + \frac{\theta^{\psi}_{i,j} \cdot b_\gamma}{h^2 \cdot \varphi \cdot C^j_n}$
15: **else**
16: Reject request γ and set $\alpha(t, \gamma) \leftarrow 0$

4.4. Performance analysis

The feasibility of the proposed PTRS algorithm is proved before the performance analysis.

Lemma 7. *When PTRS completes, its solution will not violate the constraints in the dual program Eq. (17).*

Proof. Eq. (17) presents two constraints. We first consider the second set of constraints in Eq. (17). At the beginning, PTRS sets all the dual variables to 0, so the positivity of the second constraint is satisfied. Given $n_i, n_j \in N, n_i \neq n_j$, when dual variables β_{n_i,n_j} are updated, according to the update rules in Eq. (21), they will never be decreased. Furthermore, the update rule, *i.e.*, $\alpha_{t,\gamma} \leftarrow b_\gamma(1-K_{\psi^*})$, can guarantee that $\alpha_{t,\gamma}$ is positive because only the request with $K_{\psi^*} < 1$ will be accepted. Therefore, the second set of inequalities in Eq. (17) is satisfied after PTRS ends.

Now, we consider the constraints from the first set of inequalities in Eq. (17). For each request, we will determine a scheduling scheme ψ, which has the lowest price K_{ψ^*}. According to Line 10 of Alg. 2, if this request is rejected, we have $K_{\psi^*} \geq 1$ and the right side of Eq. (18) will be negative. Because the dual variable $\alpha_{t,\gamma}$ is nonnegative, it will satisfy the first constraint set in Eq. (17). If the request of tenant t is accepted, combining with the definition of K_ψ and the update rule of Alg. 2, it follows:

$$\alpha_{t,\gamma} = b_\gamma(1 - K_{\psi^*}) \geq b_\gamma \left(1 - \sum_{n_i,n_j,n_i \neq n_j} \theta^{\psi}_{i,j} \cdot \beta_{n_i,n_j}\right) \tag{22}$$

The above inequality is the same as the first set of constraints in Eq. (17). Thus, the update of dual variables $\alpha_{t,\gamma}$ meets the first set of constraints in Eq. (17). In addition, since the update rule of the dual variables β_{n_i,n_j} will not make the right side greater, it means that the constraints will not be violated. Thus, we can conclude that the first set of constraints in Eq. (17) is also always satisfied.

We define the competitive ratio according to [26] for the performance analysis of PTRS.

Definition 1. An online algorithm is $[\zeta, \eta]$ competitive if it achieves at least $\zeta \cdot OPT$, where OPT is the result of the optimal solution, and the constraints are violated by a factor η at most.

It is apparent that we hope PTRS can achieve performance close to that of the optimal solution, *i.e.*, $\zeta \to 1$, and $\eta \to 1$. But in fact, it is difficult for any online algorithm to obtain the above performance since the TRS problem is NP-hard. We will prove that the competitive ratio of our PTRS algorithm is $[1 - \epsilon, O(\log h + \log \frac{1}{\epsilon})]$ in the following.

Lemma 8. *The system throughput of PTRS is at least* $(1-\epsilon) \cdot OPT$*, where* OPT *is the solution of the optimal method.*

Proof. When request γ is accepted, the PTRS algorithm will increase the value of the optimization objective by b_γ, but the value of the optimization objective in the dual program is only increased by Δ_i according to the update rules of dual variables in Alg. 2:

$$\begin{aligned} \Delta_i &= b_\gamma(1 - K_{\psi^*}) \\ &+ \sum_{n_i,n_j,n_i \neq n_j} \left(\beta_{n_i,n_j} \cdot \frac{\theta^{\psi}_{i,j} \cdot b_\gamma}{C^j_n} + \frac{\theta^{\psi}_{i,j} \cdot b_\gamma}{h^2 \cdot \varphi \cdot C^j_n} \right) \cdot C^j_n \\ &= b_\gamma - \sum_{n_i,n_j,n_i \neq n_j} \theta^{\psi}_{i,j} \cdot \beta_{n_i,n_j} \cdot b_\gamma \\ &+ \sum_{n_i,n_j,n_i \neq n_j} \left(\beta_{n_i,n_j} \cdot \frac{\theta^{\psi}_{i,j} \cdot b_\gamma}{C^j_n} + \frac{\theta^{\psi}_{i,j} \cdot b_\gamma}{h^2 \cdot \varphi \cdot C^j_n} \right) \cdot C^j_n \\ &\leq b_\gamma + \frac{b_\gamma}{h^2 \cdot \varphi} \cdot h^2 \cdot B^* = (1 + \epsilon) \cdot b_\gamma \end{aligned} \tag{23}$$

It means that the objective of the dual program is increased by at most $(1 + \epsilon)b_\gamma$ through the PTRS algorithm. Thus, the overall value of the optimization objective in the dual program is at least $1/(1+\epsilon) \geq 1-\epsilon$ times as that of the optimal solution. As a result, the system throughput achieved by the PTRS algorithm is at least $(1 - \epsilon) \cdot OPT$.

Now, we analyze the violation extent of the processing capacity constraints on NF instances. For each request γ and scheduling scheme ψ, let $L(n_j, \gamma)$ be the load on n_j after request γ has been processed. It should be noted that $L(n_j, \gamma)$ includes the incremental load rescheduled from other NF instances.

Lemma 9. *For each request* γ_k*, and its scheduling scheme* ψ*, we have:*

$$\beta_{n_i,n_j,k} \geq \frac{exp[L(n_j, k)/C^j_n] - 1}{h^2 \cdot \varphi} \tag{24}$$

Proof. By the induction of request $\gamma_k, k = 1, 2, \dots, |\Gamma - 1|$, we prove the lemma as follows. At the beginning, for all n_i and n_j, $\beta_{n_i,n_j,k} = L(n_j, 0) = 0$. Thus, the inequality holds. During the running of Alg. 2, $\beta_{n_i,n_j,k}$ will be updated. For clarity purpose, we use $\beta_{n_i,n_j,k}$ to denote the value of β_{n_i,n_j} after request γ_k is processed. For each arrival request γ_k, if γ_k

is rejected by the algorithm, the values of β_{n_i,n_j} and $L(n_j,k)$ will not change, *i.e.*, $\beta_{n_i,n_j,k} = \beta_{n_i,n_j,k-1}$ and $L(n_j,k) = L(n_j,k-1)$. Therefore, the inequality still holds. If request γ_k is not rejected by the algorithm, we have: $L(n_j,k) = L(n_j,k-1) + \theta^{\psi}_{i,j} \cdot b_\gamma$. Combining with the update rule of $\beta_{n_i,n_j,k}$ in Eq. (21), we also have:

$$\beta_{n_i,n_j,k} = \beta_{n_i,n_j,k-1} \cdot \left(1 + \frac{\theta^{\psi}_{i,j} \cdot b_\gamma}{C^j_n}\right) + \frac{\theta^{\psi}_{i,j} \cdot b_\gamma}{h^2 \cdot \varphi \cdot C^j_n} \tag{25}$$

By induction hypothesis, we apply inequality $\beta_{n_i,n_j,k-1} \geq \frac{\exp[L(n_j,k-1)/C^j_n]-1}{h^2\cdot\varphi}$ to Eq. (25) and obtain:

$$\begin{aligned}\beta_{n_i,n_j,k} &\geq \frac{\exp[L(n_j,k-1)/C^j_n]-1}{h^2\cdot\varphi}\left(1+\frac{\theta^{\psi}_{i,j}\cdot b_\gamma}{C^j_n}\right)+\frac{\theta^{\psi}_{i,j}\cdot b_\gamma}{h^2\varphi C^j_n}\\ &= \frac{1}{h^2\cdot\varphi}\left[\exp\left[\frac{L(n_j,k-1)}{C^j_n}\right]\left(1+\frac{\theta^{\psi}_{i,j}\cdot b_\gamma}{C^j_n}\right)-1\right]\\ &\approx \frac{1}{h^2\cdot\varphi}\left[\exp\left[\frac{L(n_j,k-1)}{C^j_n}\right]\exp\left[\frac{\theta^{\psi}_{i,j}\cdot b_\gamma}{C^j_n}\right]-1\right]\\ &= \frac{\exp[L(n_j,k)/C^j_n]-1}{h^2\cdot\varphi}\end{aligned} \tag{26}$$

In the above derivation, for a small positive value x, the first order approximation $\exp(x) \approx 1 + x$ is applied. We can establish the strict inequality by a more complicated update rule, but it will incur unnecessary complexity and difficulty. Thus, Lemma 9 holds.

Now, we give the following lemma, which presents the violation extent of processing capacity constraint on an NF instance.

Lemma 10. *The processing constraint on an NF instance will not be violated by a factor of* $O(\log h + \log(1/\epsilon))$ *through the proposed PTRS algorithm.*

Proof. Without loss of generality, if an arbitrary NF instance n_i fails, we consider the violation of the processing capacity constraint on NF instance n_j. Alg. 2 will update the value of β_{n_i,n_j} only if $K_{\psi^*} < 1$. Under this case, PTRS will accept this request, and we have $\theta^{\psi}_{i,j} \neq 0$. If $K_{\psi^*} \geq 1$, PTRS will not change the value of β_{n_i,n_j} according to Line 14 of Alg. 2. Combining that $K_{\psi^*} = \sum_{n_i,n_j,n_i\neq n_j} \theta^{\psi}_{i,j} \cdot \beta_{n_i,n_j}$, we have $\beta_{n_i,n_j} \leq 1$ before the last update of β_{n_i,n_j}. Next, we consider the last update of β_{n_i,n_j}. According to the definition of B^* given in Eq. (19) and the update rule of β_{n_i,n_j} in Line 13 of Alg. 2, we have:

$$\beta_{n_i,n_j} \leq 1 + \frac{\theta^{\psi}_{i,j}\cdot b_\gamma}{C^j_n} + \frac{\theta^{\psi}_{i,j}\cdot b_\gamma}{h^2\cdot\varphi\cdot C^j_n} \leq 1 + 2B^* \tag{27}$$

Combining with Eq. (24), it follows:

$$\frac{L(n_j,k)}{C^j_n} \leq \log((2B^*+1)\cdot h^2\cdot\varphi+1) = O(\log h + \log\frac{1}{\epsilon}) \tag{28}$$

It means that PTRS will not violate the load on NF instance n_j by a factor of $O(\log h + \log\frac{1}{\epsilon})$ at most when the failure of NF instance n_i occurs. Since two instances n_i and n_j are chosen arbitrarily, we can apply the above performance analysis for all schemes.

Then, combining Lemmas 8 and 10, the competitive ratio of Alg. 2 can be induced in the follow theorem.

Theorem 11. *The proposed PTRS algorithm can achieve the competitive ratio of* $[1-\epsilon, O(\log h + \log(1/\epsilon))]$*, where* $\epsilon \in (0,1)$*, and* h *represents the number of NF instances in clouds.*

4.5. Dealing with failure of multiple NF instances

Since the failure probability of multiple instances is very low (*e.g.*, 0.47%), we believe that choosing one backup instance for each request has met most needs. Thus, this paper mainly focuses on the scenario of single instance failure and discusses how to deal with multiple instance failures only in this section. Specifically, when multiple instances fail simultaneously, only minor modifications to PTRS can make it still applicable. Take the simultaneous failure of two instances as an example. Similar to ψ defined in Section 4.2, let ϕ be the scheduling scheme with one default instance and two backup instances. We can use $\kappa^{\phi}_{i,j,l}$ to denote that whether the request with scheme ϕ will be scheduled to instance n_l when n_i and n_j fail. Once $\kappa^{\phi}_{i,j,l}$ is determined, the problem formulation with multiple instance failures can be formulated as a linear program like Eq. (16), which can be solved with a similar primal–dual algorithm.

5. Performance evaluation

We first introduce the performance metrics and benchmarks in Section 5.1, then give the experimental results on a real testbed in Section 5.3. Last, the large-scale simulation results are presented in Section 5.2.

5.1. Performance metrics and benchmarks

5.1.1. Performance metrics

For performance comparison, we adopt the following six performance metrics: (1) the maximum number of tenants that an NF instance provides services to at any time when the system is running; (2) the system throughput; (3) the traffic load of NF instances; (4) flow completion time (FCT) [27]; (5) recovery time after a failure occurs and (6) control traffic overhead [12].

During the execution of the system, we measure how many tenants an NF instance serves and use their largest values as the first metric during the simulation. This metric shows the degree of negative effect caused by the failure of an NF instance, *i.e.*, the cloud robustness. As the number of requests is continuously increased, the maximum throughput that the system can support is measured for the second metric, which is an important indicator of system performance [28]. The last four metrics are measured through a small-scale testbed, since the they cannot be easily obtained in the simulation, we measure these metrics through a small-scale testbed. Specifically, Packet Generator (PktGen) [29] is adopted in testbed to generate network traffic. PktGen is a powerful tool also used by [30,31]. With the help of it, we can generate requests with different traffic sizes/patterns, and collect the information of FCT/load by PktGen APIs. When encountering an NF instance failure during the experiment, we record the time consumption of migrating all requests from the failed instance to corresponding backup instance(s) as the recovery time. Furthermore, we measure the total traffic amount between the data plane and the centralized plane during failure recovery procedure, and record these traffic consumption as control traffic overhead.

5.1.2. Benchmarks

We compare the proposed RoNS scheme with other three benchmarks. The first benchmark is the request-based backup (RQB) scheme [32–34], in which each request has a default instance and a backup instance. Note that, the RQB method refers to a class of failure recovery methods, and the optimization goals of these works [32–34] are different. For fair comparison, we modify the methods in [32–34] so that the optimization goals are the same to that of our proposed algorithm, *i.e.*, maximizing the system throughput. Specifically, RQB adopts a greedy-based online bin packing algorithm to assign default and backup NF instances. Two NF instances with the largest residual processing capacity will be chosen as the default and backup NF instances, respectively, for each incoming request. The second benchmark is the NF-based backup (NFB) scheme [8–11,35], in which each NF instance is assigned a backup one. That is, when an NF instance fails, all requests on this NF instance will be migrated to the backup one. According to [8–11,35],

the second benchmark first selects an NF instance in the network as the backup one, then schedules traffic to the remaining instances. When an NF instance fails, the control plane reschedules the traffic from the failed NF instance to the backup one. It should be noted that, both RQB and NFB do not limit the number of tenants served by each NF instance and the failure recovery process involves the centralized control plane, resulting in large influence scope and long recovery time. The third benchmark is OPT-LP. We first replace $z^{\psi}_{t,\gamma} \in \{0,1\}$ with $z^{\psi}_{t,\gamma} \in [0,1]$ in Eq. (16) and then derive the results of OPT-LP by optimally solving the relaxed tenant request scheduling problem. It should be noted that since OPT-LP adopts KNIT to solve the NF instance allocation problem, OPT-LP can also meet the requirement of limited influence scope. The results of OPT-LP are the upper bound of our algorithm, and are used to verify the gap between our method and the optimal solution.

5.2. Simulation evaluation

5.2.1. Simulation settings

Two practical topologies with different network sizes are chosen to evaluate the performance of the proposed methods. The first one is a small-scale NSF network topology [36], denoted by (a), which contains 16 NF instances. Since this topology does not contain the information of compute nodes and tenants, we set the number of compute nodes as 480 according to the ratio of 1:30. This ratio has been verified in the Google cluster-data [37]. In addition, the number of tenants on this topology is equal to that of the compute nodes. The second topology, denoted by (b), comes from Google cluster-data [37]. Topology (b) contains 324 NF instances and 10 047 compute nodes. The number of tenants is set to 1000. The traffic size of each request is generated according to the traffic trace in Google cluster-data [37]. Besides, p is set to 60/100 for topologies (a)/(b) by default.

5.2.2. Simulation scenarios

To thoroughly verify the performance of RoNS, we conduct the simulation under two scenarios. The first scenario, denoted as (i), does not consider the requirement of limited influence scope once a failure occurs (Section 2.1) for RQB and NFB. Under this scenario, we mainly focus on the metrics of the maximum number of tenants served by an NF instance and the system throughput. The purpose of this scenario is to verify that RoNS can limit the influence scope once an NF failure occurs, while achieving throughput performance close to that of other benchmarks. The second scenario, denoted as (ii), considers the requirement of limited influence scope for RQB and NFB. That is, same as our algorithm, each NF instance serves at most p tenants through RQB and NFB. Under this scenario, we mainly focus on the metric of system throughput. The purpose of this scenario is to verify that although RQB and NFB can be modified to meet the requirement of limited influence scope, their throughput is much lower than that of our algorithm.

5.2.3. Performance comparison in Scenario (i)

Our first set of simulations is conducted in the scenario (i), where RQB and NFB do not consider the limited influence scope requirement. The results are shown in Figs. 2–4. Fig. 2 shows how the number of requests affects the maximum number of tenants that an NF instance serves. From the simulation results, it is observed that RoNS can always perform better than both NFB and RQB, since these two benchmarks do not consider the requirement of limited influence scope. For example, in the right plot of Fig. 2, when there are $200K$ requests, the maximum number of tenants served by an NF instance are 420, 251, 100 and 100 corresponding to NFB, RQB, RoNS and OPT-LP, respectively. That is, RoNS can reduce the maximum number of affected tenants by 76.2% and 60.2% compared with NFB and RQB, respectively. Since the NF instance allocation algorithm of OPT-LP is the same as that of RoNS, OPT-LP has the same performance as RoNS on this metric.

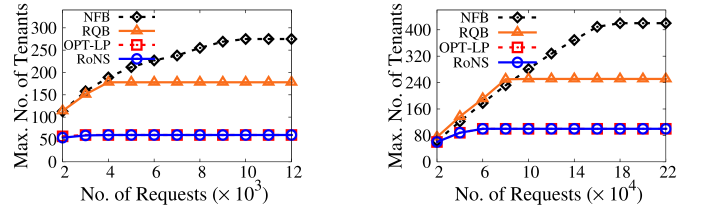


Fig. 2. Max. No. of tenants vs. No. of requests in Scenario (i) *Left plot*: Topology (a); *right plot*: Topology (b).

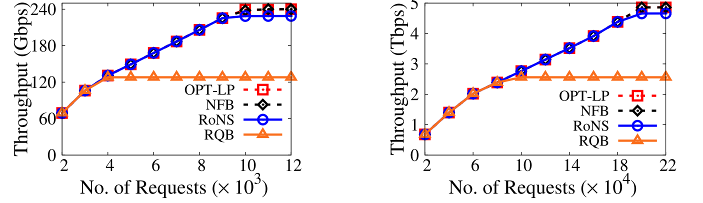


Fig. 3. Throughput vs. No. of requests in Scenario (i) *Left plot*: Topology (a); *right plot*: Topology (b).

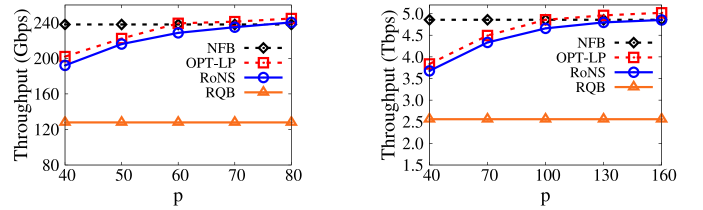


Fig. 4. Throughput vs. p in Scenario (i) *Left plot*: Topology (a); *right plot*: Topology (b).

Fig. 3 depicts how the number of request affects the system throughput. As the number of requests increases, the system throughput of all algorithms increases on both topologies. RoNS achieves similar performance compared with OPT-LP and NFB, and better performance compared with RQB. Specifically, the system throughput gap between our algorithm and OPT-LP, NFB is less than 5%. Since NFB does not consider the limited influence scope requirement, the throughput achieved by NFB is slightly higher than that of RoNS. In addition, given $10K$ requests, RoNS can increase the system throughput by about 78.7% compared with RQB in the left plot of Fig. 3. From these simulation results, we conclude that RoNS can improve the system robustness at the expense of less than 5% throughput.

Fig. 4 investigates system throughput by changing the value of p when there are $10K$ requests. The results show that the throughput of RoNS is very close to that of NFB and OPT-LP, and greater than that of RQB, with the increasing of p. For example, given $p = 60$ in the left plot of Fig. 4, the system throughputs achieved by OPT-LP, NFB, RoNS and RQB are 239.3 Gbps, 238.1 Gbps, 228.6 Gbps and 127.9 Gbps, respectively. Moreover, when p is greater than 60, the system throughputs of RoNS and OPT-LP remain and are very close to that of NFB. Since NFB processes requests as much as possible, regardless of the requirement of limited influence scope, the performance of NFB is close to that of RoNS.

From Figs. 2–4, we can conclude that without considering the limited influence scope requirement, both NFB and RQB will affect a lot of tenants when a failure occurs, resulting in low cloud robustness. Meanwhile, RoNS can limit the influence scope once a failure occurs while achieving almost the same throughput performance as OPT-LP and NFB, and much better throughput performance compared with RQB.

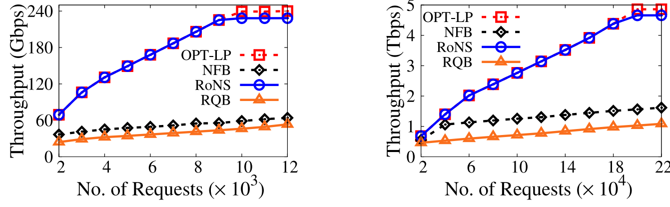


Fig. 5. Throughput vs. No. of requests in Scenario (ii) *Left plot*: Topology (a); *right plot*: Topology (b).

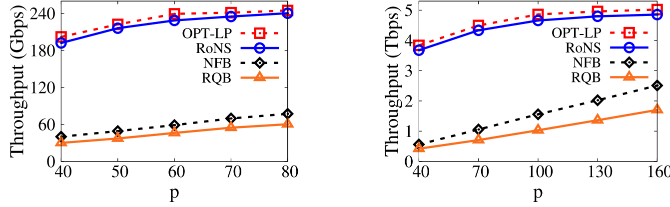


Fig. 6. Throughput vs. p in Scenario (ii) *Left plot*: Topology (a); *right plot*: Topology (b).

5.2.4. Performance comparison in Scenario (ii)

The second set of simulations is conducted in the scenario where RQB and NFB take the limited influence scope requirement into consideration. That is, same as our algorithm, each NF instance serves at most p tenants. The results are shown in Figs. 5–6. As shown in Fig. 5, the throughput achieved by all algorithms increases as the number of requests increases. The performance of RoNS is close to that of OPT-LP and far better than that of NFB and RQB. For example, when there are $220K$ requests in the right plot in Fig. 5, the gap between OPT-LP and RoNS is less than 5%. Besides, RoNS can increase throughput by 198% and 352.6% compared with NFB and RQB, respectively. Fig. 6 shows that, when the value of p increases given $200K$ requests, the throughput achieved by RoNS is better than that of NFB and RQB. For example, when $p = 60$, RoNS can improve the throughput by 288.5% and 395.2% compared with NFB and RQB, respectively, by the left plot of Fig. 6. The reason is that the limited influence scope requirement restricts the system throughputs of NFB and RQB.

From these simulation results, we can draw some conclusions. First, by Fig. 2, RoNS can significantly reduce the maximum number of affected tenants, thereby improving the system robustness. For example, when there $10K$ request in topology (a), RoNS can reduce the number of affected tenants by 76.2% and 60.2% compared with NFB and RQB, respectively. Second, from Figs. 3–4, the system throughput loss caused by limited influence scope requirement in RoNS does not exceed 5% compared with other benchmarks that ignore this requirement. Third, from Figs. 5–6, RoNS can achieve better throughput performance when NFB and RQB consider the limited influence scope requirement. For example, RoNS improves the system throughput by about 288.5% and 395.2% compared with NFB and RQB, respectively, in topology (a).

5.3. Testbed evaluation

5.3.1. Testbed settings

We implement RoNS and other benchmarks on a real testbed to evaluate their performance. Specifically, we use OpenStack [38] to construct the topology with six compute nodes and six NF instances, all with Ubuntu 18.04 OS, 4-Core vCPU and 8G of RAM. Moreover, all NF instances are running the static network address translation (SNAT), which is a typical service in cloud computing environments. During the system running, we randomly turn off an instance to simulate an instance failure. The instance with ID 1 is used as a standby instance for the NFB algorithm by default. That is, for the NFB algorithm, the standby instance will start processing requests when an instance fails. Besides, we use a server with a core i7-8700k and 32 GB of RAM as the controller. The controller is mainly responsible for traffic scheduling, as well as monitoring the status of NF instances and restoring traffic processing by installing new flow entries for comparison algorithms. The testbed contains 20 tenants, and p is set to 10 by default. We implement our experiments with traffic trace in Google cluster-data [37]. The number of requests ranges from 200 to 800. We adopt Packet Generator (PktGen) [29] to generate requests and load information of each instance.

5.3.2. NF instance performance

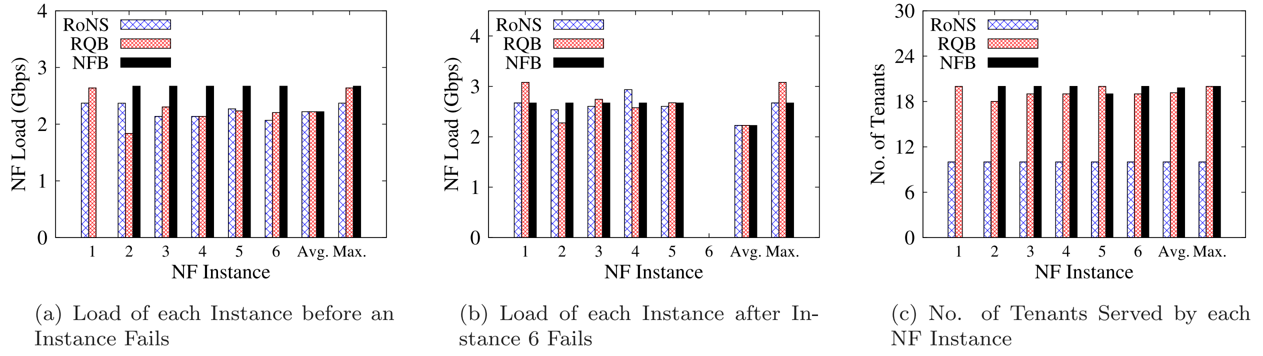
Fig. 7 shows the performance of each NF instance. We generate 400 requests in total and schedule these requests to NF instances based on the results of algorithms. Figs. 7(a) and 7(b) show the load information of each NF instance before and after an NF instance failure, respectively. The results show that the total throughput of RoNS is close to that of NFB and RQB. Moreover, no matter before or after failure, the maximum load among NF instances through RoNS is lower than that of RQB and NFB. Fig. 7(c) depicts the number of tenants served by each NF instance. We observe that the number of tenants served by any instance through RoNS is much smaller than that of RQB and NFB. Specifically, RoNS can reduce the average number of tenants served by an instance by about 50% compared with other benchmarks. It indicates better system robustness of RoNS.

5.3.3. Failure recovery performance

In Fig. 8, we investigate the failure recovery performance by changing the number of requests. During the system running, we randomly turn off an instance to simulate an instance failure, and then measure the delay and control traffic overhead required to recovery the processing of requests served by the failed instance. When an NF failure occurs, the comparison algorithms need to be aware of failures, compute new routing paths and install the corresponding forwarding rules to reroute requests. With the help of agents designed in the data plane, the failure recovery of RoNS is decoupled from the centralized control plane, thereby reducing recovery time significantly. Fig. 8(a) shows that the recovery time of RoNS is much lower than that of the comparison algorithms as the number of requests grows. For example, when there are 400 requests, the recovery time of RoNS, RQB and Standy are 315.94 ms, 1165 ms and 1200 ms, respectively. Fig. 8(b) shows that the control traffic overhead grows when the number of requests increases for NFB and RQB. In contrast, the control traffic overhead of RoNS during the failure recovery procedure is always zero, since our algorithm does not require the participation of the centralized controller for failure recovery. When there are more than 600 requests, the requests served by RQB is close to saturation, so the growth trend of its recovery time and control traffic overhead slows down. Fig. 8(c) shows the FCT performance of 400 requests. We observe that there are 66 and 80 requests with FCT over 1000 ms through RQB and NFB, respectively. Since RoNS can achieve a much lower recovery time, the FCT performance of RoNS is better than that of benchmarks.

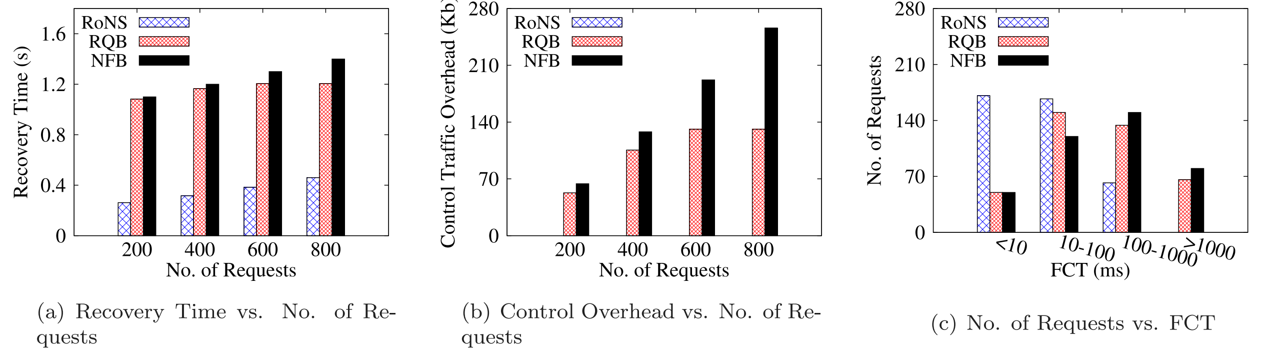
5.3.4. Limited influence scope performance

Since every tenant hopes their traffic to be processed as quickly as possible, the nearly last FCT can be a meaningful metric to measure the QoS of tenants [39]. Thus, we study how the failure of an instance affects the 99%tile FCT of each tenant in Fig. 9. From the testbed results, we see that the average 99%tile FCT among tenants of RoNS is much lower than that of other benchmarks. For example, the average 99%tile FCTs of RoNS, RQB and NFB are 188.1 ms, 1163.3 ms and 1199.3 ms, respectively. There are two reasons for the significant performance improvement of RoNS. First, since RoNS limits the number of tenants served by an instance, the scope of the affected tenants is limited once an NF instance fails. Second, with the help of agents designed in the data plane, the failure recovery time is much lower



(a) Load of each Instance before an Instance Fails (b) Load of each Instance after Instance 6 Fails (c) No. of Tenants Served by each NF Instance

Fig. 7. NF instance performance.



(a) Recovery Time vs. No. of Requests (b) Control Overhead vs. No. of Requests (c) No. of Requests vs. FCT

Fig. 8. Failure recovery performance.

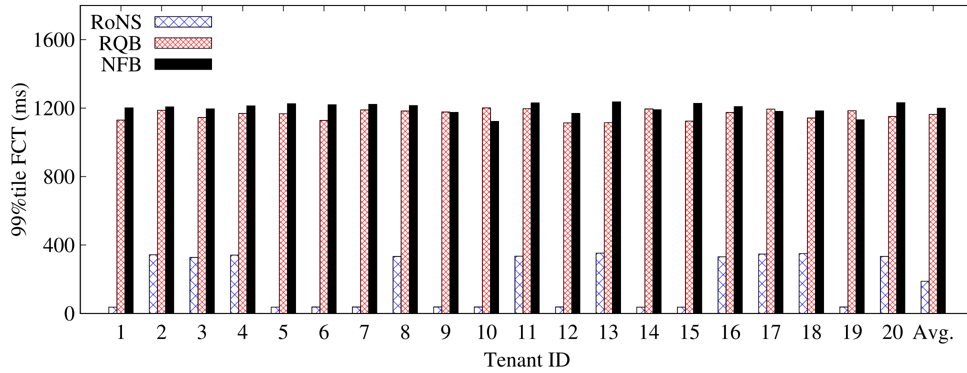


Fig. 9. 99%tile FCT vs. Tenant ID for limited influence scope performance comparison.

than that of benchmarks. Thus, RoNS performs better than benchmarks in terms of 99%tile FCT.

According to the above experiment results, we can draw the following conclusions. First, Figs. 7–8 show that RoNS can achieve much better failure recovery performance (recovery time, control traffic overhead and FCT) when encountering an instance failure with similar throughput performance to benchmarks. Second, according to Fig. 9, RoNS can achieve much lower 99%tile FCT of each tenant through limiting the influence scope and fast failure recovery with the help of agents.

6. Related works

The problem of request scheduling on NFs has been widely studied in recent years for different targets, such as utility maximization, throughput optimization or latency minimization [40–42]. Gu et al. [40] investigate how to manage the resource of NF instances (*i.e.*, deactivation and activation of NF instances). The authors also study request scheduling algorithm for the purpose of maximizing the overall network utility, while taking various costs and end-to-end delay into consideration. Since this problem is formulated as a non-convex program, it is difficult to design a heuristic algorithm to obtain the optimal solution. Thus, they resort to the deep reinforcement learning technique, which has low computation complexity. The authors in [41] study the runtime request scheduling problem to satisfy service function chaining (SFC) requirement. They design an integer allocation maximum pressure policy, which can achieve nearly optimal throughput. To apply their method in the large-scale distributed system, they also present a multi-site cooperative algorithm called STEAM, which can achieve performance close to that of the optimal solution and also fulfill the runtime requirements. Liu et al. [42] study the request scheduling problem by jointly considering multiple resource constraints and dynamic NF instance placement. They devise an auxiliary weighted graph for this problem, then propose two algorithms to solve it. Their optimization goal is to minimize the cost of resource consumption while satisfying multiple QoS constraints.

Although these works have conducted in-depth research on request scheduling for NFs with different optimization targets, they ignore NF failures and their impact on system robustness. To deal with NF failure, the currently popular approach is to deploy backup NF instances using network function virtualization (NFV) technique [8–11,32–35]. After an NF instance fails, the centralized control plane reschedules the requests from the failed instance to the backup one. Backup-based failure recovery can be divided into two categories. The first one is called *request-based backup* scheme for simplicity, in which each request has a default and backup instance. The request-based backup scheme is adopted by [32–34]. For example, the authors in [32] proposes a control plane called OpenNF, which provides a replicated state for each request on the assigned backup instance. OpenNF's southbound API defines a standard NF interface for the control plane to reschedule requests to the assigned backup instance. Tomassilli et al. [33] consider building a failure recovery scheme with the global rerouting strategy in which a new routing to the backup instance of each request is computed for each failure situation. Their optimization object is to minimize the amount of consumed bandwidth.

The second one is called *NF-based backup* scheme for simplicity, in which each NF instance is assigned a backup one. Moreover, a backup instance will only process requests if there are NF failures, and it can be backup for multiple instances. The NF-based backup scheme is adopted by [8–11,35]. For example, Li et al. [8] focus on provisioning reliable network service under the scenario of mobile edge-cloud (MEC), by placing primary and backup NF instances to cloudlets, so that the service reliability requirements of users can be guaranteed. They formulate it as an NF instance reliability problem, and its optimization goal is to maximize the total revenue by processing as many as users' requests. The authors in [9] study the problem of how to backup NF instances to minimize the cost in an online manner, and propose a self-adaptive scheme named SAB to efficiently backup NF instances over both the edge and the cloud. Zhang et al. [10] propose to minimize the backup resource consumption while considering the heterogeneous resource demands of different NFs. They formalize it as the resource-aware backup allocation problem, and prove the NP-hardness of this problem. Then they propose an approximate algorithm based on differential evolution to solve it.

However, these works only focus on the deployment of backup NF instances, and ignore the requirements of limited influence scope and fast recovery time. Therefore, by their method, NF failures affect a large number of tenants, and the failure recovery time will be too long to be tolerated. Though there are works like [13,14] which propose fast recovery mechanism for link failure, their method can only handle link failures, not NF failures. For example, Petale et al. [13] propose the link failure recovery mechanism through the fast failover group table feature provided by OpenFlow. They store the forwarding rules of the backup paths in the fast failover group table. When the OpenFlow switch detects a link failure, it will forward packets through the backup paths.

Different from all the discussed works above, to deal with NF failures, this paper proposes to control the influence scope of NF failures by limiting the number of served tenants on each NF instance while scheduling requests. Moreover, this paper presents the design of agents in the data plane which can achieve fast failure recovery without the participation of the centralized control plane.

7. Conclusion

This paper investigates building robust network function services (RoNS) in clouds. To limit the influence scope once a failure occurs, RoNS limits the number of tenants served by an NF instance. To achieve fast failure recovery, RoNS design agents in the data plane. Due to traffic dynamics, RoNS takes a two-phase approach: NF instance allocation and tenant request scheduling. Two algorithms with bounded approximation factors have been designed to for these two phases. Both experiment and simulation results show the high performance of our proposed algorithms compared with state-of-the-art solutions.

CRediT authorship contribution statement

Huaqing Tu: Conceptualization, Methodology, Writing – original draft. **Gongming Zhao:** Formal analysis, Writing – review & editing. **Hongli Xu:** Methodology, Writing – review & editing. **Yangming Zhao:** Writing – review & editing. **Yuhang Qiu:** Software, Validation. **Liusheng Huang:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors are unable or have chosen not to specify which data has been used.

Acknowledgment

The corresponding authors of this paper are Gongming Zhao and Hongli Xu. This work was supported by the National Science Foundation of China under Grants No. 62132019, 62102392; and in part by Open Research of Projects of Zhejiang Lab (No. 2022QA0AB04); and in part by the Natural Science Foundation of Jiangsu Province under Grant BK20210121.

References

[1] R. Kumar, S. Charu, An importance of using virtualization technology in cloud computing, Glob. J. Comput. Technol. 1 (2) (2015).
[2] The amazon web service, URL http://www.aws.amazon.com/.
[3] The google cloud platform, URL http://www.cloud.google.com/.
[4] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, A. Ghalsasi, Cloud computing the business perspective, Decis. Support Syst. 51 (1) (2011) 176–189.
[5] K. Gai, J. Guo, L. Zhu, S. Yu, Blockchain meets cloud computing: a survey, IEEE Commun. Surv. Tutor. 22 (3) (2020) 2009–2030.
[6] A. Engelmann, A. Jukan, A reliability study of parallelized VNF chaining, in: 2018 IEEE International Conference on Communications (ICC), IEEE, 2018, pp. 1–6.
[7] R. Potharaju, 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.
[8] J. Li, W. Liang, M. Huang, X. Jia, Reliability-aware network service provisioning in mobile edge-cloud networks, IEEE Trans. Parallel Distrib. Syst. 31 (7) (2020) 1545–1558.
[9] X. Shang, Y. Huang, Z. Liu, Y. Yang, Reducing the service function chain backup cost over the edge and cloud by a self-adapting scheme, IEEE Trans. Mob. Comput. (2021).
[10] J. Zhang, Z. Wang, C. Peng, L. Zhang, T. Huang, Y. Liu, Raba: Resource-aware backup allocation for a chain of virtual network functions, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, 2019, pp. 1918–1926.
[11] X. Shang, Y. Liu, Y. Mao, Z. Liu, Y. Yang, Greening reliability of virtual network functions via online optimization, in: 2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS), IEEE, 2020, pp. 1–10.
[12] G. Zhao, H. Xu, J. Liu, C. Qian, J. Ge, 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.
[13] S. Petale, J. Thangaraj, Link failure recovery mechanism in software defined networks, IEEE J. Sel. Areas Commun. 38 (7) (2020) 1285–1292.
[14] A. Tomassilli, G. Di Lena, F. Giroire, I. Tahiri, D. Saucez, S. Perennes, T. Turletti, R. Sadykov, F. Vanderbeck, C. Lac, Poster: design of survivable SDN/NFV-enabled networks with bandwidth-optimal failure recovery, in: 2019 IFIP Networking Conference (IFIP Networking), IEEE, 2019, pp. 1–2.
[15] S. Yang, F. Li, R. Yahyapour, X. Fu, Delay-sensitive and availability-aware virtual network function scheduling for NFV, IEEE Trans. Serv. Comput. (2019).
[16] L. Mai, Y. Ding, X. Zhang, L. Fan, S. Yu, Z. Xu, Energy efficiency with service availability guarantee for network function virtualization, Future Gener. Comput. Syst. 119 (2021) 140–153.
[17] X. Fan, H. Xu, H. Huang, X. Yang, Real-time update of joint SFC and routing in software defined networks, IEEE/ACM Trans. Netw. (2021).

[18] A. Srinivasan, Approximation algorithms via randomized rounding: A survey, 2001, Series in Advanced Topics in Mathematics Polish Scientific Publishers Pwn.
[19] M. Hellman, J. Raviv, Probability of error, equivocation, and the Chernoff bound, IEEE Trans. Inform. Theory 16 (4) (1970) 368–372.
[20] C. Tellambura, Evaluation of the exact union bound for trellis-coded modulations over fading channels, IEEE Trans. Commun. 44 (12) (1996) 1693–1699.
[21] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, et al., The design and implementation of open vswitch, in: 12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15), 2015, pp. 117–130.
[22] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, ACM SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74.
[23] Y.-D. Lin, H.-Y. Teng, C.-R. Hsu, C.-C. Liao, Y.-C. Lai, Fast failover and switchover for link failures and congestion in software defined networks, in: 2016 IEEE International Conference on Communications (ICC), IEEE, 2016, pp. 1–6.
[24] M. Borokhovich, L. Schiff, S. Schmid, Provable data plane connectivity with local fast failover: Introducing openflow graph algorithms, in: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, 2014, pp. 121–126.
[25] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, J. Wilkes, Large-scale cluster management at Google with Borg, in: Proceedings of the Tenth European Conference on Computer Systems, 2015, pp. 1–17.
[26] L. Guo, J. Pang, A. Walid, Joint placement and routing of network function chains in data centers, in: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, IEEE, 2018, pp. 612–620.
[27] N. Dukkipati, N. McKeown, Why flow-completion time is the right metric for congestion control, ACM SIGCOMM Comput. Commun. Rev. 36 (1) (2006) 59–62.
[28] V.A. Rusakov, Using metrics in the throughput analysis and synthesis of undirected graphs, in: International Conference on Integrated Science, Springer, 2020, pp. 277–287.
[29] R. Olsson, Pktgen the linux packet generator, in: Proceedings of the Linux Symposium, Vol. 2, Ottawa, Canada, 2005, pp. 11–24.
[30] G. Chen, Y. Lu, Y. Meng, B. Li, K. Tan, D. Pei, P. Cheng, L.L. Luo, Y. Xiong, X. Wang, et al., Fast and cautious: Leveraging multi-path diversity for transport loss recovery in data centers, in: 2016 {USENIX} Annual Technical Conference ({USENIX}{ATC} 16), 2016, pp. 29–42.
[31] W.-E. Chen, Packet forwarding enhancement for virtualized next-generation core networks, in: 2018 27th Wireless and Optical Communication Conference (WOCC), IEEE, 2018, pp. 1–2.
[32] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, A. Akella, Opennf: Enabling innovation in network function control, ACM SIGCOMM Comput. Commun. Rev. 44 (4) (2014) 163–174.
[33] A. Tomassilli, G. Di Lena, F. Giroire, I. Tahiri, D. Saucez, S. Perennes, T. Turletti, R. Sadykov, F. Vanderbeck, C. Lac, Design of robust programmable networks with bandwidth-optimal failure recovery scheme, Comput. Netw. 192 (2021) 108043.
[34] H. Wang, H. Xu, H. Huang, M. Chen, S. Chen, Robust task offloading in dynamic edge computing, IEEE Trans. Mob. Comput. (2021).
[35] J. Fan, M. Jiang, O. Rottenstreich, Y. Zhao, T. Guan, R. Ramesh, S. Das, C. Qiao, A framework for provisioning availability of NFV in data center networks, IEEE J. Sel. Areas Commun. 36 (10) (2018) 2246–2259.
[36] G. Sun, Z. Chen, H. Yu, X. Du, M. Guizani, Online parallelized service function chain orchestration in data center networks, IEEE Access 7 (2019) 100147–100161.
[37] Google cluster-data, URL http://github.com/google/cluster-data/.
[38] O. Sefraoui, M. Aissaoui, M. Eleuldj, Openstack: toward an open-source solution for cloud computing, Int. J. Comput. Appl. 55 (3) (2012) 38–42.
[39] N.K. Sharma, C. Zhao, M. Liu, P.G. Kannan, C. Kim, A. Krishnamurthy, A. Sivaraman, Programmable calendar queues for high-speed packet scheduling, in: 17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20), 2020, pp. 685–699.
[40] L. Gu, D. Zeng, W. Li, S. Guo, A.Y. Zomaya, H. Jin, Intelligent VNF orchestration and flow scheduling via model-assisted deep reinforcement learning, IEEE J. Sel. Areas Commun. 38 (2) (2019) 279–291.
[41] M. Blöcher, R. Khalili, L. Wang, P. Eugster, Letting off STEAM: Distributed runtime traffic scheduling for service function chaining, in: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, IEEE, 2020, pp. 824–833.
[42] L. Liu, S. Guo, G. Liu, Y. Yang, Joint dynamical VNF placement and SFC routing in NFV-enabled SDNs, IEEE Trans. Netw. Serv. Manag. (2021).

Huaqing Tu 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 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 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 in 2002 and 2007, respectively. He is currently an Associate Professor with the School of Computer Science and Technology, University of Science and Technology of China. He has authored or coauthored over 70 papers, and held about 30 patents. His main research interest is software-defined networks, cooperative communication, and vehicular ad hoc network.

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.

Yuhang Qiu received B.S. degree in 2021 from the University of Science and Technology of China. He is currently a MA.Sc. candidate in the School of Computer Science and Technology, University of Science and Technology of China. His main research interests are software defined networks and cloud computing.

Liusheng Huang received the M.S. degree in computer science from the University of Science and Technology of China in 1988. He is currently a Senior Professor and the Ph.D. Supervisor with the School of Computer Science and Technology, University of Science and Technology of China. He has authored six books and over 300 journal/conference papers. His research interests are in the areas of Internet of Things, vehicular ad hoc network, information security, and distributed computing.