Reducing the Service Function Chain Backup Cost over the Edge and Cloud by a Self-adapting Scheme

Xiaojun Shang, Yaodong Huang, Zhenhua Liu and Yuanyuan Yang
Stony Brook University, Stony Brook, NY 11794, USA

Abstract—Emerging virtual network functions (VNFs) bring new opportunities to network services on the edge within customers' premises. Network services are realized by chained up VNFs, which are called service function chains (SFCs). These services are deployed on commercial edge servers for higher flexibility and scalability. Despite such promises, it is still unclear how to provide highly available and cost-effective SFCs under edge resource limitations and time-varying VNF failures. In this paper, we propose a novel Reliability-aware Adaptive Deployment scheme named RAD to efficiently place and back up SFCs over both the edge and the cloud. Specifically, RAD first deploys SFCs to fully utilize edge resources. It then uses both static backups and dynamic ones created on the fly to guarantee the availability under the resource limitation of edge networks. RAD does not assume failure rates of VNFs but instead strives to find the sweet spot between the desired availability of SFCs and the backup cost. Theoretical performance bounds, extensive simulations, and small-scale experiments highlight that RAD provides significantly higher availability with lower backup costs compared with existing baselines.

Index Terms—Virtual network functions, Edge computing, Service function chain, Availability

1 Introduction

The development of virtual network functions (VNFs) transforms traditional middleboxes, e.g., firewalls, load balancers, proxies, on dedicated hardware to virtual functions on commercial servers and thus introducing more flexibility, scalability, and cost-efficiency. To promote such benefits, much research has been conducted, e.g., [1]–[4]. With the rapid development of edge computing and 5G networks, there is a growing motivation to deploy VNFs on the edge within customers' premises for lower latency and better performance. See [5]–[9] as examples.

Despite such benefits, detaching a network function from its specifically designed hardware may degrade its availability [10]–[12]. For instance, in many VNF systems, a VNF runs as an instance on a virtual machine (VM) with resources managed by an underlying hypervisor. Therefore, any failure of the hypervisor may cause the VNFs running over it unavailable [10]. To make matters worse, when multiple VNFs chain up to provide a network service as a whole, a failure of any VNF on this service function chain (SFC) makes the entire service unavailable. Therefore, the availability problem of an SFC is more severe than that of a single VNF [11].

Providing VNF backups is an effective way to improve the availability and has been widely studied for SFCs deployed in the cloud [11]–[15]. However, schemes designed for cloud environments face new challenges when SFCs are deployed on the edge. In particular, while a VNF may need multiple backups to guarantee its availability [14], resources on edge networks are often limited compared to those in the cloud. Simply duplicating each VNF by a prefixed number of copies may exceed the edge resource capacity. In addition, different VNFs may experience distinct failure rates that are dynamic over time. A thoughtful SFC backup scheme for edge networks needs to consider all the trade-offs to decide when to back up each VNF by how many copies and where to place the copies.

Clearly, cloud resources can be utilized when resources on the edge are insufficient [16], [17]. However, with the deployment of a larger number of (smaller) edge servers in 5G networks, the propagation delay involving multiple hops from the edge to the cloud is often much larger than that among edge servers within one edge network [16], [18]. Furthermore, forwarding service flows from the edge to the cloud when failures happen introduces extra traffic and may congest the network. Simply backing up SFCs in the cloud incurs costs, e.g., extra delay, congestion, cloud resource usage charge comparing to backing up over the edge. Therefore, we need a scheme to minimize the backup cost with limited edge resources while guaranteeing the SFC availability. One key challenge is that VNF failures are time-varying and hard to predict due to various failure types and causes [10].

In this paper, we propose a Reliability-aware Adaptive SFC Deployment scheme named RAD. An early version of this work was presented at [19], which focuses on backup strategies. We now extend RAD into an efficient SFC deployment scheme integrating SFC placement, adaptive backup deployment, and resource adjustment over the edge and the cloud. We also conduct practical experiments to show the feasibility and efficiency of RAD in real-world scenarios.

Since we focus on SFCs that bring more benefits if deployed on the edge, the RAD scheme first deploys SFCs aiming at the highest utilization of edge resources. Existing work maximizing resource usage of SFCs, e.g., [20], [21], has

theoretical bounds degrading with the size of the problem. Authors of [22] proposed a scheme with an approximation ratio of 6 to the optimal solution. The SFC deployment scheme in our RAD further pushes the constant bound to 2 or 4 in different edge resource situations. For those SFCs deployed in the cloud due to the limitation of edge resources, their reliability is taken care of by cloud service providers with Service Level Agreements [12], which is out of the scope of our paper. For SFCs deployed on the edge experiencing time-varying failures, RAD guarantees their reliability with low backup cost.

To achieve this goal, RAD deploys one static backup for each VNF and determines where to place the backups to minimize the backup cost under edge resource constraints. If the static backups of an SFC are deployed in the cloud, additional backup costs may be paid but its reliability is guaranteed by cloud service providers. For SFCs with static backups on the edge, one static backup may not be sufficient to meet the availability requirements [23]. Instead of deploying more static backups, RAD initializes a dynamic backup on the edge whenever a VNF or its static backup fails. The dynamic backup is released when the failure recovers. The fast creation of a dynamic backup has been verified by existing VNF platforms, e.g., [24], and can be accelerated by methods such as early VNF failure detection [10]. The existence of static backups further makes sure that a dynamic backup has enough time to set up unless both the VNF and the static backup fail successively within a very short time. As most VNFs recover quickly [14], the lifespan of the dynamic backups is short, making its resource footprint relatively small. Therefore, these dynamic backups can often be accommodated.

In the case of sudden failure rate spikes, RAD moves static backups of SFCs with lower backup cost from the edge into the cloud, thus freeing up more resources to accommodate additional dynamic backups on the edge. On the contrary, the scheme backs up more SFCs on the edge to reduce cloud backup cost when failure rates decrease. This adjustment thus optimizes backup placements adaptively to guarantee the desired availability while minimizing the backup cost.

While some research considers the availability problem of VNFs on edge networks [17], [25], [26], to the best of our knowledge, no existing method applies both static and dynamic backups over the edge and the cloud without knowing the failure rate of each VNF.

Our main contributions are summarized as follows.

- We design RAD for the sweet spot between high availability and low backup cost of SFCs over the edge and the cloud without assuming VNF failure patterns. RAD consists of SFC deployment, static backup deployment, dynamic backup deployment, and backup adjustment.
- As the SFC deployment problem is NP-hard, we solve it using a scheme that contains two sub-algorithms with approximation ratios of 2 and 4 under different network conditions.
- Similarly, we develop an approximation algorithm with a theoretical guarantee and good performance in practice for the static backup deployment problem.
- We further propose an online algorithm to solve the dynamic backup deployment problem and prove its competitive ratio compared to the offline optimal solution. RAD adjusts VNF backups between the edge and the cloud based on the feedback from the online algorithm for better performance.
- We conduct real-world trace-driven numerical simulations and small-scale experiments. Results highlight that our proposed RAD provides SFC deployment with high availability and low backup cost.

The remainder of this paper is organized as follows. Section 2 presents an overview of the RAD scheme. Section 3 formulates the SFC deployment problem and presents algorithms placing SFCs with constant bounds. Section 4 continues with the algorithm for the static backup deployment. Section 5 handles the dynamic backup deployment and the SFC backup adjustment. The performance evaluation of RAD is presented in Section 6. Section 7 briefly reviews the related work and Section 8 concludes this paper.

2 Overview of the RAD Scheme

Fig. 1 demonstrates the procedure of RAD with a brief example. In this section, we show the architecture of RAD and the general idea of designing each of its steps. Theoretical details will be presented in Section 3, 4, and 5.

In this paper, we consider types of SFCs that should be deployed at the edge for lower latency and better performance. Thus, the SFC deployment should utilize edge resources as much as possible to maximize such benefits. Here, we do not consider cases when some VNFs of one SFC are deployed on the edge while others are deployed in the cloud. This is because, in scenarios where RAD applies, e.g., [16], [18], dividing an SFC between the edge and the cloud will introduce large network latency and offset the benefits of placing SFCs on the edge. Therefore, an SFC will be deployed on the cloud as a whole if any of its VNFs cannot be deployed on the edge due to edge resource limitations. The SFC deployment will be described in detail in Section 3. After placing SFCs, the availability of SFCs on the edge needs to be further guaranteed. We achieve this goal by utilizing a novel mechanism with both static and dynamic backups. It operates with low backup costs and robust to failures changing over time.

In the static backup deployment, we deploy one static backup onto the edge or into the cloud for each edge VNF aiming at the minimization of the backup cost without violating the resource limitation of any edge server. For the same reason shown in the SFC deployment, if any SFC cannot be fully backed up on the edge, it is backed up completely in the cloud. The detailed deployment method is illustrated in Section 4. After the static backup deployment, the availability of SFCs with static backups in the cloud is guaranteed by well-established reliability mechanisms in the cloud [12]. However, for an SFC backed up on the edge, a single static backup for each VNF may not meet its availability requirement. There exist situations when neither the VNF nor its static backup is responding. To further guarantee the availability, we need to deploy more backups for each VNF backed up on the edge. As mentioned in Section 1, it is difficult to decide how many backups a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2020.3048885, IEEE Transactions on Mobile Computing

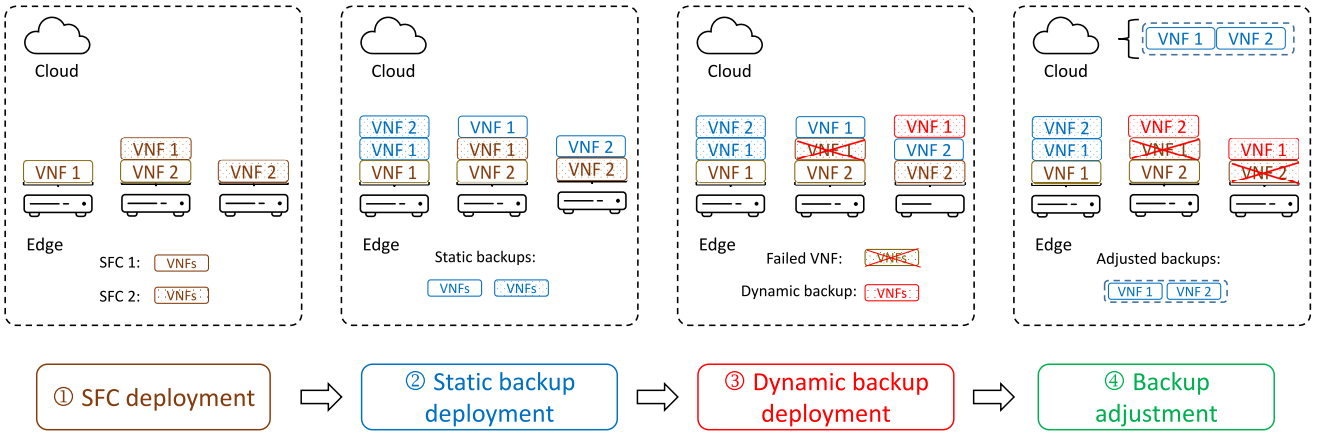


Fig. 1. The demonstration of RAD. In Step 1, RAD deploys SFCs in brown to edge servers for the maximal edge resource utilization. VNFs with different patterns belong to different SFCs. In Step 2, RAD deploys each VNF at the edge a static backup in blue for the lowest backup cost. Any SFC or static backup that cannot be deployed at the edge due to resource constraints will be placed in the cloud instead. In Step 3, whenever a VNF fails (marked by a red cross), RAD deploys a dynamic backup in red for the failed VNF and releases it if the VNF recovers. If edge resources are not sufficient for dynamic backups, RAD goes to Step 4 and adjusts static backups of some SFCs to the cloud, thus releasing more edge resources.

particular VNF needs and where to place these backups in advance, since failure rates of VNFs are hard to predict and change over time. Thus, we prefer not to decide which VNFs need more backups in advance and solve the problem in an online manner.

For such purpose, we use a dynamic backup deployment method which creates a dynamic backup whenever a VNF or its static backup just fails. With the existence of static backups, dynamic backups often have sufficient time to initialize when failures occur. Each dynamic backup is placed on an edge server using an online algorithm. The algorithm balances the load on each edge server in the long term to mitigate resource contention which is a main cause of VNF failures [27]. When both the VNF and the static backup resume, the dynamic backup is released. However, if the current availability (previous time operating normally/previous operating time) of an SFC is not satisfied, dynamic backups of this SFC will not be released, thus reinforcing the availability of this SFC. Since most VNFs recover after some short time and release corresponding dynamic backups, the resource utilization of dynamic backups at each moment is relatively small. Therefore, remaining edge resources are often sufficient for dynamic backups, and thus guaranteeing the required availability of SFCs.

When the current edge resources are not enough for upcoming dynamic backups, the dynamic backup deployment is paused. We then apply an SFC backup adjustment method to move backups of the SFC with the lowest cost from the edge to the cloud. The reliability of this SFC is then guaranteed by the cloud and its static and dynamic backups on the edge are released. The online algorithm then resumes. When failure rates of VNFs increase, the scheme adaptively adjusts more SFCs to the cloud to further guarantee the availability. When failure rates decrease, the scheme adaptively backs up more SFCs on the edge to reduce the cloud backup cost. The detailed dynamic backup deployment method and the SFC backup adjustment method are elaborated in Section 5.

3 SFC Deployment

We start with how to deploy SFCs onto the edge and the cloud. We aim to utilize as much edge resource as possible to maximize the benefits of SFCs on the edge. Important notations used in this paper are summarized in Table 1.

TABLE 1

Notation	Definition
V	Set of servers on the edge, $V = \{1, 2, ..., v, ..., \|V\|\}$
F	Set of SFCs, $F = \{1, 2, ..., f, ..., \|F\|\}$
I_f	Set of VNFs of SFC f, $I_f = \{1, 2, ..., i, ..., \|I_f\|\}$
$x_{f,i,v} \in \{0, 1\}$	Decision variable whether VNF i of SFC f is deployed on server v.
$y_{f,i,v} \in \{0, 1\}$	Decision variable whether static backup i of SFC f is deployed on server v.
$z_{k,v} \in \{0, 1\}$	Decision variable whether dynamic backup k is deployed on server v.
C	Total Resources on the edge
R_v	Resources on edge server v
a_v	Resource demand on v before deploying static backups
b_v	Resource demand on v before dynamic backups
β_f	Resource demand of SFC f
$\beta_{f,i}$	Resource demand of VNF i of SFC f
w_f	Backup cost of SFC f
$o_{f,i}$	Server holding the VNF i of SFC f
$o_{k,1}, o_{k,2}$	Servers holding VNF and static backup of k
K	Set of dynamic backups $K = \{1, 2, ..., k, ..., \|K\|\}$
γ_k	Resource demand of the dynamic backup k
W_v	Load on v after deploying $\|K\|$ dynamic backups

3.1 Model of the SFC Deployment

We suppose that the edge network consists of a set of servers denoted by $V = \{1, 2, ..., v, ..., |V|\}$. Denote by R_v the resources available for VNFs on server v. We define the set of SFCs to be deployed in the network as $F = \{1, 2, ..., f, ..., |F|\}$. All VNFs of SFC f form a set I_f with each VNF $i \in I_f$. We further define the resource demand of SFC f and any VNF i of this SFC as β_f and $\beta_{f,i}$,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2020.3048885, IEEE Transactions on Mobile Computing

respectively. We need to determine $x_{f,i,v}$, a binary variable denoting whether VNF i of SFC f is placed on server v.

$$\max_{x_{f,i,v}} \quad \sum_{f\in F} \beta_f \cdot \min_{i\in I_f} \left(\sum_{v\in V} x_{f,i,v} \right)$$

$$\text{s.t.} \quad \sum_{f\in F} \sum_{i\in I_f} \beta_{f,i} \cdot x_{f,i,v} \le R_v, \ \forall v \in V, \tag{1}$$

$$x_{f,i,v} \in \{0,1\}, \ \forall f \in F, \forall i \in I_f, \forall v \in V. \tag{2}$$

In the objective function, we aim to maximize the edge resources used by SFCs. If any VNF i of SFC f is not deployed on the edge, i.e., $\sum_v x_{f,i,v} = 0$, the resource utilization of that SFC on the edge, β_f, will not be counted in the objective function. The whole SFC f is thus deployed onto the cloud because it will reduce the left side of Constraint (1) without changing the objective function. Constraint (1) restricts that the total resource demand of VNFs on each server v should not exceed its capacity R_v. Constraint (2) is the integer constraint making sure that a VNF cannot be split over multiple machines.

When there are many SFCs or edge servers, the computational complexity of directly solving the problem modeled above may be unaffordable. The reason is that, when each SFC only has one VNF and there is only one edge server, a knapsack problem which is NP-hard can be reduced to our problem. In this way, our problem is also an NP-hard problem.

3.2 SFC Deployment Algorithm

To reduce the computational complexity of solving the problem above, we propose a novel SFC deployment algorithm. The algorithm consists of two sub-algorithms named Tight-SFC deployment and Loose-SFC deployment. Either of the sub-algorithms can solely solve the SFC deployment problem. Nevertheless, Tight-SFC achieves a tighter theoretical bound under more restrictive conditions, while the bound of Loose-SFC stands under most circumstances. For each specific case, we run both sub-algorithms and the output of the SFC deployment algorithm is determined by the winning one of the two sub-algorithms which utilizes more edge resources.

3.2.1 Tight-SFC Deployment

Tight-SFC has a tighter bound comparing with Loose-SFC but requires more resource capacity for each edge server. Denote the total resources for deploying SFCs on the edge by C, where $C = \sum_{v\in V} R_v$. Tight-SFC first sorts the set of edge servers V in a decreasing order of R_v to get a set V'. It also sorts the set of SFCs F in a decreasing order of β_f to get set F'. Tight-SFC further sorts VNFs of each SFC f in F' in a decreasing order of $\beta_{f,i}$ to get sets I'_f.

Starting from the last SFC in F' with the smallest β_f, the algorithm divides VNFs of each SFC into groups and makes sure that the aggregated demand of any group in SFC $f+1$ is smaller than that of any group in SFC f. VNF groups formulate a set G in a decreasing order of the aggregated resource demand. We simply denote by g the g^{th} largest VNF group in G. Starting from the first VNF group $\widetilde{g}$ that can fit the largest edge server, Tight-SFC deploys VNF groups successively into server v in V' which has the largest R_v and enough remaining resources for this VNF group. This process stops when G is traversed or a VNF group cannot fit any v. An SFC is deployed onto the edge if all its VNFs belong to those deployed successive VNF groups. Tight-SFC then uses a greedy method (*Greedy*) to determine the deployment of remaining SFCs.

Greedy: The method tries to deploy each VNF of remaining SFCs into an edge server with the largest remaining resources. If any VNF of SFC f cannot find an available server, the whole SFC f is deployed onto the cloud instead.

Algorithm 1 Tight-SFC Algorithm

Input: Set F, V, $\{R_v\}$, $\{\beta_f\}$, $\{\beta_{f,i}\}$
Output: The deployment of SFCs in F

Calculate $C = \sum_{v\in V} R_v$.
Sort V in a decreasing order of R_v to get V'.
Sort F in a decreasing order of β_f to get F'.
Sort I_f in a decreasing order of $\beta_{f,i}$ to get I'_f.
$G \leftarrow \emptyset$, $g_{pr} \leftarrow \emptyset$
for $f = |F'|, ..., 1$ **do**
  for $i = |I'_f|, ..., 1$ **do**
    if the resource demand of ungrouped VNFs of f is smaller than g_{cu} **then**
      Group all unpacked VNFs of f to g_{cu}, $G \leftarrow G \cap g_{cu}$, $g_{pr} \leftarrow g_{cu}$, $g_{cu} \leftarrow \emptyset$, break.
    end if
    if $g_{cu} \le g_{pr}$ **then**
      $g_{cu} \leftarrow g_{cu} \cap VNF(f,i)$.
    else
      $G \leftarrow G \cap g_{cu}$, $g_{pr} \leftarrow g_{cu}$, $g_{cu} \leftarrow \emptyset$
    end if
  end for
end for
Sort G in a decreasing order.
for all $g \in G$ **do**
  for all $v \in V'$ **do**
    if v is capable for g **then**
      Group g is deployed on v and break.
    end if
  end for
  if No server v can hold g **then**
    Break.
  end if
end for
Apply *Greedy* to deploy SFCs in F_{remain}, which contains all SFCs with VNFs not deployed onto the edge.
for all $f \in F_{remain}$ **do**
  for all $i \in I'_f$ **do**
    for all $v \in V'$ **do**
      if v is capable for i **then**
        Group g is deployed on v.
      end if
    end for
  end for
  if All VNFs in f can be deployed **then**
    f is placed onto the edge.
  else
    f is placed into the cloud.
  end if
end for

Tight-SFC is shown in detail in Algorithm 1. The complexity of sorting all the sets in Algorithm 1 is $|F||I|\log(|I|)$, where $|I|$ is the size of the maximal I_f. The complexity of generating VNF groups is $|F||I|$. Deploying VNF groups and running *Greedy* both take the complexity of $|F||I||V|$.

Thus, the general computational complexity of Algorithm 1 is $|F||I||V|$. Finally, the theoretical guarantee of Algorithm 1 is stated in the following theorem.

Theorem 1. *Suppose the edge resources used by Algorithm 1 is $Z^\dagger$ and we have Z^* as the edge resources utilized by an optimal solution. Assuming $\max_{g\in G}\{\beta_g\} \leq \min_{v\in V}\{R_v\}$, we have $\frac{Z^\dagger}{Z^*} \geq \frac{1}{2}$. Here, β_g is the resource demand of VNF group g.*

The assumption in Theorem 1 means that the edge server with the fewest VNF resources can hold the largest VNF group. Since the largest VNF group utilizes no more resources than the largest SFC and an edge server often has sufficient resources for one network service, i.e., an SFC, this assumption is achieved in many cases. To prove Theorem 1, we need the following lemma.

Lemma 1. *Assume $\max_{g\in G}\{\beta_g\} \leq \min_{v\in V}\{R_v\}$. If Algorithm 1 cannot deploy all VNF groups on the edge, the resource demand of VNF groups deployed on the edge is at least $\frac{C}{2}$.*

Proof. Since $\max_{g\in G}\{\beta_g\} \leq \min_{v\in V}\{R_v\}$, if Algorithm 1 cannot deploy all VNF groups onto the edge, it starts from the first group and stops after deploying the group $\hat{g}$, i.e., $\hat{g}+1$ cannot fit any edge server. Also due to the assumption, all edge servers are deployed with VNF groups when the deployment stops. Assuming the total resource demand of all VNF groups deployed in server v is d_v, we have that $d_v + d_{v+1} \geq R_v$ for $v \in \{1, 2, ..., |V'|-1\}$. This is because if $d_v + d_{v+1} < R_v$, all VNF groups in $v+1$ will be deployed in v instead. Summing the inequality up from 1 to $|V'|-1$, we have $d_1 + 2\sum_{v=2}^{|V'|-1} d_v + d_{|V'|} \geq \sum_{v=1}^{|V'|-1} R_v$. In addition, we have $d_1 + d_{|V'|} \geq R_1 \geq R_{|V'|}$, otherwise all VNF groups in server $|V'|$ can be deployed in server 1. Therefore, summing up the two inequality, we have $\sum_{g=1}^{\hat{g}} \beta_g = \sum_{v=1}^{|V'|} d_v \geq \frac{1}{2}\cdot\sum_{v=1}^{|V'|} R_v = \frac{C}{2}$. □

With Lemma 1, we can now prove Theorem 1 as follows.

Proof. According to Lemma 1, Algorithm 1 either deploys all SFCs on the edge or utilizes at least half of the edge resources. Obviously, any optimal algorithm can at most utilize all edge resources. Thus, the approximation ratio of Algorithm 1 is at least $\frac{C/2}{C} = \frac{1}{2}$. □

3.2.2 Loose-SFC Deployment

Compared to Tight-SFC, Loose-SFC preserves a theoretical bound when capacities of edge servers are much smaller. Similar to Tight-SFC, Loose-SFC first gets V' and F' by sorting. It then successively picks SFC f in F' and formulates an empty set F_2. If the sum of current resource demand of set F_2 and β_f does not exceed $\frac{C}{2}$, f is added to F_2. This process stops when the sum becomes larger than $\frac{C}{2}$ for the first time or F' is traversed. In this way, we have $\sum_{f\in F_2} \beta_f \leq \frac{C}{2}$. Denote all VNFs of SFCs in set F_2 as I_2 and sort I_2 in a decreasing order of $\beta_{f,i}$ to get I_2'. Loose-SFC then successively deploys the largest VNF i in I_2' to the capable server v in V' with the largest R_v. If a VNF of any SFC cannot be deployed, the whole SFC will not be deployed. The process continues until I_2' is traversed. At this time, SFCs in F_2 with all VNFs deployed onto edge servers are deployed on the edge. Loose-SFC then applies *Greedy* similar to that in Algorithm 1 to determine the deployment of remaining SFCs. The detailed algorithm is illustrated in Algorithm 2.

Algorithm 2 Loose-SFC Algorithm

Input: Set F', V', $\{R_v\}$, $\{\beta_f\}$, $\{\beta_{f,i}\}$
Output: The deployment of SFCs in F
1: $F_2 = \emptyset$
2: **for all** $f \in F'$ **do**
3: **if** $\sum_{j\in F_2} \beta_j + \beta_f \leq \frac{C}{2}$ **then**
4: $F_2 \leftarrow F_2 \cup f$
5: **end if**
6: **end for**
7: Denote the set of all VNFs in F_2 by I_2. Sort I_2 in a decreasing order of $\beta_{f,i}$ and get I_2'.
8: **for all** $i \in I_2'$ **do**
9: **for all** $v \in V'$ **do**
10: **if** v is capable for VNF i **then**
11: VNF i is deployed at v
12: Break.
13: **end if**
14: **end for**
15: **if** VNF i is not deployed at any v **then**
16: The SFC with VNF f will not be deployed.
17: **end if**
18: **end for**
19: Apply *Greedy* to deploy remaining SFCs.

Similar to Algorithm 1, the computational complexity of Algorithm 2 is $|F||I||V|$. We now show the theoretical bound of Algorithm 2 by proving Theorem 2 as follows.

Theorem 2. *Suppose the result of Algorithm 2 is $Z^\dagger$ and we have Z^* as the result of the optimal solution. Assuming $\max_{f\in F, i\in I}\{\beta_{f,i}\} \leq \min_{v\in V}\{R_v\}$ and $\max_{f\in F}\{\beta_f\} \leq \frac{C}{2}$, we have $\frac{Z^\dagger}{Z^*} \geq \frac{1}{4}$.*

In Theorem 2, we assume that the edge server with the fewest VNF resources can hold the largest VNF on any SFC. This assumption is looser than that in Theorem 1 and can often be satisfied in edge networks. We also exclude extreme cases that an SFC demands more than half of the total edge resources, i.e., $\max_{f\in F}\{\beta_f\} \leq \frac{C}{2}$. To prove Theorem 2, we need to introduce the following Lemma 2.

Lemma 2. *Assuming $\max_{f\in F, i\in I}\{\beta_{f,i}\} \leq \min_{v\in V}\{R_v\}$, if $\sum_{f\in F_2} \beta_f \leq \frac{C}{2}$, then all SFCs in F_2 can be deployed on edge servers.*

Proof. We prove by contradiction and assume that the resource demand of SFCs is less than $\frac{C}{2}$ and there exist some SFCs cannot be deployed on the edge completely. According to the proof of Lemma 1, there must exist one VNF $\hat{i}$ which does not fit any server v when all servers are deployed with some VNFs. According to Lemma 1, the current resource demand of all deployed VNFs is thus larger than $\frac{C}{2}$, which is contradicting to the assumption. □

With Lemma 2, we then prove Theorem 2 as follows.

Proof. Lines 1-6 makes sure that the total resource demand of SFCs to be deployed on the edge is less than $\frac{C}{2}$. According

to Lemma 2 and the proof of Theorem 1, lines 7-14 can deploy all VNFs on these SFCs on edge servers. We then need to prove that the total resource demand of these SFCs are above $\frac{C}{4}$.

We assume $\max_{f\in F}\{\beta_f\} \leq \frac{C}{2}$ and focus on the case that $\sum_{f=1}^{|F'|}\beta_f \geq \frac{C}{4}$. This is because, if $\sum_{f=1}^{|F'|}\beta_f < \frac{C}{4}$, all SFCs can be deployed on the edge according to Lemma 2. Thus, the performance of Algorithm 2 is as good as that of the optimal solution according to Lemma 2.

With $\sum_{f=1}^{|F'|}\beta_f \geq \frac{C}{4}$, if all SFCs starting from 1 in F' are in F_2 and deployed on the edge, the resource demand of deployed SFCs is of course larger than $\frac{C}{4}$. If there exists one $\hat{f}$ in F_2 that SFC $\hat{f}+1$ is not in F_2 for the first time, we have $2\sum_{f=1}^{\hat{f}}\beta_f \geq \sum_{f=1}^{|\hat{f}|}\beta_f + \beta_{\hat{f}+1} \geq \frac{C}{2}$. Since SFCs from 1 to $\hat{f}$ belong to F_2, the resource demand of SFCs in F_2 is larger or equal to $\frac{C}{4}$. Therefore, Algorithm 2 can at least utilize $\frac{C}{4}$ edge resources which proves Theorem 2. □

4 Static Backup Deployment

In this section, we present the static backup deployment which deploys each VNF on the edge a static backup while minimizing the backup cost.

4.1 Model of the Static Backup Deployment

For the deployment of static backups, we need to determine $y_{f,i,v}$, a binary variable denoting whether the backup of VNF i of SFC f is placed on server v. Denote by F_{edge} the set of SFCs deployed on the edge by the SFC deployment. We use $U(Y)$ to denote the total backup cost, where $Y = \{y_{f,i,v} | \forall f \in F_{edge}, \forall i \in I_f, \forall v \in V\}$. We minimize $U(Y)$ subject to constraints of resources, reliability, and indivisibility of VNFs as follows.

$$\begin{aligned}
\min \quad & U(Y) \\
\text{s.t.} \quad & \sum_{f\in F_{edge}}\sum_{i\in I_f}\beta_{f,i}\cdot y_{f,i,v} \leq R_v - a_v,\ \forall v\in V, && (3)\\
& y_{f,i,v} = 0, \quad \forall f\in F_{edge}, \forall i\in I_f, v = o_{f,i}, && (4)\\
& y_{f,i,v}\in\{0,1\},\ \forall f\in F_{edge}, \forall i\in I_f, \forall v\in V. && (5)
\end{aligned}$$

In this paper, we restrict our attention to a particular objective function. The method can be applied to more general scenarios. In our application scenario mentioned in Section 1, the backup cost of SFC f is the extra cost incurred by backing up f in the cloud instead of on the edge. Denote the backup cost of SFC f by w_f, which depends on factors such as types and quantity of VNFs, the resource demand and the delay requirement of SFC f. We thus formulate

$$U(Y) = \sum_{f\in F_{edge}} w_f \cdot \max_{i\in I_f}\left(1 - \sum_{v\in V} y_{f,i,v}\right)^+.$$

If any VNF i of SFC f is not backed up on the edge (i.e., $\sum_{v\in V} y_{f,i,v} = 0$), the whole SFC f is backed up into the cloud, and the backup cost w_f is thus counted.

For resource constraint (3), the overall resource demand on v cannot exceed the total resources on this server, R_v. Since SFCs are deployed on edge servers, we denote a_v as the resource demand on v before deploying static backups. The resource demand of static backups cannot exceed $R_v - a_v$. For Constraint (4), denote by $o_{f,i}$ the node holding VNF i of SFC f. According to the reliability requirement, the backup VNF cannot be deployed on the same server with the original one in case of hardware failures. Therefore, for each f and i, we have $y_{f,i,v} = 0$ if $v = o_{f,i}$. Constraint (5) is the integer constraint restricting that a VNF cannot be split.

Similar to the proof in 3.1, we can transform the static backup deployment problem and reduce a knapsack problem to it. Thus, the static backup deployment problem is also NP-hard. We omit the details due to space limitations.

4.2 Static Backup Deployment Algorithm

In this section, we propose a static backup deployment algorithm to solve the problem with much lower complexity while guaranteeing a theoretical bound to the optimal solution. In the algorithm, we define a new binary variable y_f and let $y_f = \max_{i\in I_f}(1 - \sum_{v\in V} y_{f,i,v})^+$. The variable y_f represents whether an SFC on the edge is backed up in the cloud, i.e., $y_f = 1$ represents that SFC f is backed up in the cloud. We substitute y_f for $\max_{i\in I_f}(1 - \sum_{v\in V} y_{f,i,v})^+$ in the objective function and formulate an equivalent ILP problem with both $y_{f,i,v}$ and y_f and an extra constraint, which is

$$1 - \sum_{v\in V} y_{f,i,v} \leq y_f, \quad \forall f\in F_{edge}, \forall i\in I_f. \qquad (6)$$

By relaxing $y_{f,i,v}\in\{0,1\}$ and $y_f\in\{0,1\}$ to $\tilde{y}_{f,i,v}\in[0,1]$ and $\tilde{y}_f\in[0,1]$, we then get a linear programming (LP) problem. We solve the LP problem with an LP solver [28] and get relaxed solutions $\{\tilde{y}_{f,i,v}\}$ and $\{\tilde{y}_f\}$.

We further need to determine whether each $y_{f,i,v}$ and y_f should be 0 or 1 based on the relaxed solutions. In the static backup deployment algorithm, we first determine y_f and let the value of y_f determines the corresponding set of $y_{f,i,v}$ for the same SFC. If $y_{f'} = 1$ for a particular f', there must be at least one $i\in I_{f'}$ with $(1 - \sum_{v\in V} y_{f',i,v})^+ = 1$. Then we can make $(1 - \sum_{v\in V} y_{f',i,v})^+ = 1$ for all $i\in I_{f'}$. Since this will not change the objective but reduce the left side of Constraint (3) as all $y_{f',i,v} = 0$. If $y_{f'} = 0$, it is clear that $(1 - \sum_{v\in V} y_{f',i,v})^+ = 0$ for all $i\in I_{f'}$.

For a particular $i'\in I_{f'}$, since Constraint (3) is linear, we only choose one $v\in V$ and make $y_{f',i',v} = 1$ while satisfying Constraint (4). This is because multiple $y_{f',i',v} = 1$ will not reduce the objective function but only increase the left side of Constraint (3). Based on the fractional result $\{\tilde{y}_{f,i,v}\}$, when $\sum_v y_{f',i',v} = 1$, we look for the $y_{f',i',v}$ with the largest $\tilde{y}_{f',i',v}$ among all v and make it 1. By following steps above, whenever the rounding of y_f is given, all corresponding $y_{f,i,v}$ are determined.

To determine $\{y_f\}$, we first sort all $\tilde{y}_f$ in a decreasing order. Denote by θ the threshold for rounding and always equal to the largest $\tilde{y}_f$ in each iteration. In the first iteration, all y_f are set to 0. For each f and i, $y_{f,i,v}$ with the largest $\tilde{y}_{f,i,v}$ among all v are set to 1. In each iteration, any y_f with

$\tilde{y}_f = \theta$ is set to 1 and corresponding $y_{f,i,v}$ are set to 0. This means that the static backup deployment algorithm determines these SFCs to be backed up in the cloud. Then, the value of $\tilde{y}_f$ is set to 0. After all y_f with $\tilde{y}_f = \theta$ are determined and there still exist violated constraints, a new iteration begins until all constraints are satisfied for the first time. The total number of iterations is limited by $|F_{edge}|$. When this process finishes, all y_f with $\tilde{y_f} \geq \theta$ are rounded to 1 and others rounded to 0. We further add a withdraw procedure for better performance. For each $f \in F_{edge}$ and $y_f = 1$, we set y_f back to 0 and corresponding $y_{f,i,v}$ back to 1, if no constraint is violated by doing so. The detailed static backup deployment algorithm is presented in Algorithm 3.

Algorithm 3 Static Backup Deployment Algorithm

Input: $\{\tilde{y}_{f,i,v}\}$ and $\{\tilde{y}_f\}$ from the LP solver.
Output: Binary output $\{y_{f,i,v}\}$, $\{y_f\}$ and threshold θ
for all $f \in F_{edge}, i \in I_f$ **do**
  if $\tilde{y}_{f,i,v} = \max_{v \in V \setminus o_{f,i}} \{\tilde{y}_{f,i,v}\}$ **then**
    $y_{f,i,v} \leftarrow 1$
  else
    $y_{f,i,v} \leftarrow 0$
  end if
end for
Denote the set $\{y_{f,i,v} | y_{f,i,v} = 1\}$ for each y_f as Y_f.
while Constraints are not satisfied **do**
  $\theta \leftarrow \max_{f \in F_{edge}} \{\tilde{y}_f\}$
  for all $\tilde{y}_f = \theta$ **do**
    $\tilde{y}_f \leftarrow 0$
    $y_f \leftarrow 1$ and corresponding $y_{f,i,v} \leftarrow 0$
  end for
end while
for all $f \in F_{edge}$ **do**
  if $y_f = 1$ and setting $y_{f,i,v} \in Y_f$ back to 1 does not violate any constraint **then**
    $y_f \leftarrow 0$ and set all $y_{f,i,v} \in Y_f$ back to 1.
  end if
end for

In Algorithm 3, the complexity of getting corresponding $y_{f,i,v} = 1$ for every y_f is $|F_{edge}||I||V|\log(|V|)$. The process of determining y_f with the withdraw procedure totally takes $|F_{edge}|^2|V|$. Since the number of SFCs is often much larger than the maximal length of SFCs ($|F_{edge}| \gg |I|$), the computational complexity of Algorithm 3 except solving an LP is $|F_{edge}|^2|V|$. We then prove that $\frac{1}{\theta}$ is a bound between the result of Algorithm 3 and that of the optimal solution.

Theorem 3. *Suppose the result of Algorithm 3 is $Z^\dagger$ and we have Z^* as the result of the optimal solution. Then, we have $Z^\dagger \leq \frac{1}{\theta} \cdot Z^*$.*

Proof. Since $y_f = \max_{i \in I_f}(1 - \sum_{v \in V} y_{f,i,v})^+$, the goal is to minimize $\sum_{f \in F_{edge}} w_f \cdot y_f$. First, since the optimal solution of a relaxed problem is at least as good as the original ILP problem, we have

$$Z^* \geq \sum_{f \in F_{edge}} w_f \cdot \tilde{y}_f.$$

From Algorithm 3, we get the threshold θ which sets all y_f with $\tilde{y}_f$ above it as 1 and below it as 0. It is clear that

$$\sum_{f \in F_{edge}} w_f \cdot \tilde{y}_f \geq \sum_{\{\tilde{y}_f | \tilde{y}_f \geq \theta\}} w_f \cdot \tilde{y}_f \geq \theta \cdot \sum_{\{y_f | y_f = 1\}} w_f \cdot y_f.$$

Suppose the set of y_f that are set back to 0 through the withdraw process is Y_f. It is clear that

$$\sum_{\{y_f | y_f = 1\}} w_f \cdot y_f \geq \sum_{\{y_f | y_f = 1\}/Y_f} w_f \cdot y_f = Z^\dagger.$$

In this way, we have $Z^* \geq \theta \cdot Z^\dagger$. □

Whenever Algorithm 3 is finished, the bound $\frac{1}{\theta}$ is known. In addition, the performance is often much better than the bound with the withdraw procedure. The efficiency of Algorithm 3 will be evaluated in Section 6.

5 Dynamic Backup Deployment

When Algorithm 3 finishes, each VNF backed up on the edge has one static backup. Since the availability of SFCs may not be guaranteed yet and future failures of VNFs are hard to predict, we decide which SFC needs more backups in an online manner. RAD deploys dynamic backups for SFCs with failed VNFs or static backups.

When deploying dynamic backups, we need to balance the load of each server to avoid resource contention as much as possible. Deploying dynamic backups without considering load balancing may cause some servers with heavy load. Heavy-loaded servers suffer from resource contention which is a main reason for unavailable VNFs. In addition, VNF failures occur successively over time. To guarantee the availability, we cannot wait for later failures before deploying previous dynamic backups. Therefore, we need an algorithm to deploy dynamic backups in an online manner while balancing load on servers. In this section, we formulate the dynamic backup deployment problem and design an online algorithm with a proven competitive ratio to the offline optimum. We also discuss conditions to release dynamic backups and the SFC backup adjustment method which deals with insufficient or excessive edge resources during the dynamic backup deployment.

5.1 Model of Dynamic Backup Deployment

For the deployment of dynamic backups, denote by K the set of dynamic backups arriving over time, where $K = \{1, 2, ..., k, ..., |K|\}$. For simplicity, we assume no VNF recovers during this time which leads to the heaviest overall load and the fiercest resource contention. Dynamic backup k has a resource demand γ_k. We define a decision variable $z_{k,v}$ which denotes whether the dynamic backup k is deployed on node v. We also define a variable W_v representing the load (resource demand/resource capacity) on node v after deploying $|K|$ dynamic backups. We aim to minimize the maximal W_v. This goal should be achieved under constraints of resource and reliability requirements. We formulate the dynamic backup deployment problem as follows.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2020.3048885, IEEE Transactions on Mobile Computing

$$\min \quad \max_{v \in V} \quad W_v$$

$$\text{s.t.} \quad \frac{b_v}{R_v} + \sum_{k \in K} \frac{\gamma_k}{R_v} \cdot z_{k,v} = W_v, \quad \forall v \in V, \tag{7}$$

$$W_v \leq 1, \quad \forall v \in V, \tag{8}$$

$$\sum_{v \in V} z_{k,v} = 1, \quad \forall k \in K, \tag{9}$$

$$z_{k,v} = 0, \quad \forall k \in K, v = \{o_{k,1}, o_{k,2}\}, \tag{10}$$

$$z_{k,v} \in \{0,1\}, W_v \geq 0, \ \forall k \in K, \forall v \in V. \tag{11}$$

With the deployment of static backups from Algorithm 3, we denote b_v as the current resource demand on v. Therefore, Constraint (7) makes sure that W_v is the final load on node v after deploying all dynamic backups, which is the sum of VNFs, static backups, and dynamic backups over the resource. Constraint (8) serves as the resource constraint and ensures that W_v, the load on server v, is restricted by 1. Constraint (9) restricts that there must be one edge server chosen to hold each dynamic backup. Constraint (10) is the reliability constraint, where $o_{k,1}$ and $o_{k,2}$ are servers holding the VNF and the static backup of k. We cannot deploy dynamic backup k on $o_{k,1}$ or $o_{k,2}$.

The formulated problem is non-trivial since failures of VNFs happen in an online manner over time. Dynamic backup k should be allocated to an edge server as soon as its VNF or static backup fails without knowing any future information such as $\gamma_{k'}$, where $k' > k$. The formulated problem is thus an online integer linear programming problem. To solve this problem, we propose an online algorithm with a competitive ratio of $\log(|V|)$ to the offline optimum.

5.2 Online Algorithm with a Competitive Ratio

With the sequence of total K dynamic backups, the online algorithm operates in total N iterations (N is bounded). In iteration n, the algorithm maintains a load parameter M_n. For each server v, it also maintains an iteration parameter $\eta_{n,v}$. Before deploying the first dynamic backup, we set $\eta_{1,v}$ to 0 for all v. M_1 is the largest load on any v with the current deployment of VNFs and static backups, i.e., $M_1 = \max_{v \in V}\{\frac{b_v}{R_v}\}$. At the beginning of placing each dynamic backup k, we exclude servers that cannot hold k from the set V. If V becomes empty, the algorithm is terminated and the backup adjustment mechanism introduced in Section 5.4 is triggered. We define an increment $\delta^k_{n,v} = \frac{\gamma_k}{R_v \cdot M_n}$ for each remaining server v. We sort remaining servers in an increasing order of $\epsilon^{\eta_{n,v}+\delta^k_{n,v}} - \epsilon^{\eta_{n,v}}$ to get V', where $\epsilon \in (1, \frac{\rho+1}{\rho}]$, $\rho > 1$. We find the smallest server v' in V' which does not belong to the set $\{o_{k,1}, o_{k,2}\}$. If $\eta_{n,v'} + \delta^k_{n,v'} \leq log_\epsilon(\frac{\rho}{\rho-1}|V|)$, the dynamic backup k is deployed on this server and $\eta_{n,v'} = \eta_{n,v'} + \delta^k_{n,v'}$. If not, a new iteration starts with all $\eta_{n+1,v} = 0$ and $M_{n+1} = 2 \cdot M_n$. The detailed process is presented in Algorithm 4.

Denote the result of Algorithm 4 (the offline optimal solution) by $M^\dagger$ (M^*, respectively). We now prove that the result of Algorithm 4 is bounded by a competitive ratio to the offline optimum in Theorem 4.

Theorem 4. *Assume that Algorithm 4 has not excluded the optimal server for dynamic backup k when deploying it. We have*

Algorithm 4 Dynamic Backup Deployment Algorithm

Input: Set of edge servers V, deployment of current static backups, dynamic backup set K.
Output: Placement of each dynamic backup $\{x_{k,v} | \forall k \in K, \forall v \in V\}$.

1: $n \leftarrow 1; \eta_{1,v} \leftarrow 0, \forall v \in V; M_1 \leftarrow \max_{v \in V}\{\frac{b_v}{R_v}\}$
2: Whenever a dynamic backup comes, $overflow \leftarrow True$ and do lines 3-23.
3: **for all** $v \in V$ **do**
4: **if** $\sum_{k'=1}^{k-1} \frac{\gamma_{k'}}{R_v} \cdot x_{k',v} + \frac{\gamma_k}{R_v} > 1$ **then**
5: $V \leftarrow V \setminus v$
6: **end if**
7: **end for**
8: **if** $V = \emptyset$ **then**
9: Terminate the algorithm and execute backup adjustment.
10: **end if**
11: **while** $overflow = True$ **do**
12: $overflow = False$
13: **for all** $v \in V$ **do**
14: $\delta^k_{n,v} \leftarrow \frac{\gamma_k}{R_v \cdot M_n}$
15: **end for**
16: Sort servers in V in an increasing order of $\epsilon^{\eta_{n,v}+\delta^k_{n,v}} - \epsilon^{\eta_{n,v}}$ to get V', where $\epsilon \in (1, \frac{\rho+1}{\rho}]$, $\rho > 1$.
17: Find the smallest v' in V' satisfying $v \notin \{o_{k,1}, o_{k,2}\}$.
18: **if** $\eta_{n,v'} + \delta^k_{n,v'} > log_\epsilon(\frac{\rho}{\rho-1}|V|)$ **then**
19: $n \leftarrow n+1; M_n \leftarrow 2 \cdot M_{n-1}; \eta_{n,v} \leftarrow 0, \forall v \in V; overflow \leftarrow True$
20: **else**
21: $x_{k,v'} = 1; \eta_{n,v'} \leftarrow \eta_{n,v'} + \delta^k_{n,v'}$
22: **end if**
23: **end while**

$M^\dagger \leq (1 + 4log_\epsilon(\frac{\rho}{\rho-1}|V|)) \cdot M^*$, *where* $\epsilon \in (1, \frac{\rho+1}{\rho}]$ *and* $\rho > 1$.

When $|V| > 1$, the bound in Theorem 4 is minimized if ρ satisfies $\frac{\rho-1}{\rho} \cdot e^{\frac{\rho+1}{\rho-1}\log(\frac{\rho+1}{\rho})} = |V|$. This conclusion and the detailed value of ρ can be obtained by well-established numerical methods. To prove Theorem 4, we first need to prove the following lemma.

Lemma 3. *If $M^* \leq M_n$, Algorithm 4 can deploy all dynamic backups within the n^{th} iteration.*

Proof. We assume $\eta_{n,v}(k)$ as the parameter $\eta_{n,v}$ of server v in iteration n after deploying dynamic backup k. We also represent $\delta^k_{n,v}$ by $\delta_{n,v}(k)$. Suppose $\Phi_n(k)$ is the set of backups deployed by Algorithm 4 in iteration n after deploying backup k, $\Phi^*_v(k)$ is the set of backups deployed by the offline optimum on v after deploying backup k. Thus, we have $\eta^*_{n,v}(k) = \frac{\sum_{k' \in \Phi_n(k) \cap \Phi^*_v(k)} \gamma_{k'}}{R_v \cdot M_n}$. We define a function $F_n(k) = \sum_{v \in V} \epsilon^{\eta_{n,v}(k)} \cdot (\rho - \eta^*_{n,v}(k))$, where $\rho > 1$. Here, we first prove that $F_n(k)$ is a non-increasing function. Knowing V may be shrinking due to lines 3-7 in Algorithm 4, we denote by $V(k)$ the set V when deploying dynamic backup k and have $V(k+1) \subseteq V(k)$. We first have

$$F_n(k+1) - F_n(k) = \sum_{v \in V(k+1)} \epsilon^{\eta_{n,v}(k+1)} \cdot (\rho - \eta^*_{n,v}(k+1)) - \sum_{v \in V(k+1)} \epsilon^{\eta_{n,v}(k)} \cdot (\rho - \eta^*_{n,v}(k))$$

$$- \sum_{v \in V(k) \setminus V(k+1)} \epsilon^{\eta_{n,v}(k)} \cdot (\rho - \eta^*_{n,v}(k)).$$

Since $\eta^*_{n,v}(k) \leq \frac{M^*}{M_n} \leq 1 < \rho$, we further have

$$F_n(k+1) - F_n(k) \leq \sum_{v \in V(k+1)} \epsilon^{\eta_{n,v}(k+1)} \cdot (\rho - \eta^*_{n,v}(k+1))$$

$$- \sum_{v \in V(k+1)} \epsilon^{\eta_{n,v}(k)} \cdot (\rho - \eta^*_{n,v}(k)) = H_1. \quad (12)$$

We define $v^\dagger$ and v^* as corresponding nodes chosen by Algorithm 4 and offline algorithm to place dynamic backup $k+1$. According to the assumption in Theorem 4, we have $v^* \in V(k+1)$. When $v^\dagger$ and v^* are different nodes, we simplify the right hand side of (12) and get

$$H_1 = (\epsilon^{\eta_{n,v^\dagger}(k+1)} - \epsilon^{\eta_{n,v^\dagger}(k)}) \cdot (\rho - \eta^*_{n,v^\dagger}(k))$$
$$-\epsilon^{\eta_{n,v^*}(k)} \cdot \delta_{n,v^*}(k+1).$$

When $v^\dagger$ and v^* are the same node, we have

$$H_1 = (\epsilon^{\eta_{n,v}(k+1)} - \epsilon^{\eta_{n,v}(k)}) \cdot (\rho - \eta^*_{n,v}(k))$$
$$-\epsilon^{\eta_{n,v}(k+1)} \cdot \delta_{n,v}(k+1).$$

Here, v can be either $v^\dagger$ or v^*. Since $-\epsilon^{\eta_{n,v^*}(k+1)} \leq -\epsilon^{\eta_{n,v^*}(k)}$, considering both cases, we have

$$H_1 \leq (\epsilon^{\eta_{n,v^\dagger}(k+1)} - \epsilon^{\eta_{n,v^\dagger}(k)}) \cdot (\rho - \eta^*_{n,v^\dagger}(k))$$

$$-\epsilon^{\eta_{n,v^*}(k)} \cdot \delta_{n,v^*}(k+1) = H_2. \quad (13)$$

We further have

$$H_2 \leq \rho \cdot (\epsilon^{\eta_{n,v^\dagger}(k+1)} - \epsilon^{\eta_{n,v^\dagger}(k)}) - \epsilon^{\eta_{n,v^*}(k)} \cdot \delta_{n,v^*}(k+1).$$

Due to lines 16 and 17, we have $v^\ddagger, v^* \notin \{o_{k,1}, o_{k,2}\}$ and $\epsilon^{\eta_{n,v^\dagger}(k+1)} - \epsilon^{\eta_{n,v^\dagger}(k)} \leq \epsilon^{\eta_{n,v^*}(k+1)} - \epsilon^{\eta_{n,v^*}(k)}$. Therefore, the inequality goes

$$H_2 \leq \rho \cdot (\epsilon^{\eta_{n,v^*}(k+1)} - \epsilon^{\eta_{n,v^*}(k)}) - \epsilon^{\eta_{n,v^*}(k)} \cdot \delta_{n,v^*}(k+1)$$
$$= \epsilon^{\eta_{n,v^*}(k)} \cdot [\rho \cdot (\epsilon^{\delta_{n,v^*}(k+1)} - 1) - \delta_{n,v^*}(k+1)].$$

Since v^* is the choice of the offline optimum, we have $\frac{\gamma_{k+1}}{R_{v^*}} \leq M^*$. Then we have $\delta_{n,v^*}(k+1) = \frac{\gamma_{k+1}}{R_{v^*} \cdot M_n} \leq \frac{M^*}{M_n} \leq 1$. Then $\rho \cdot (\epsilon^{\delta_{n,v^*}(k+1)} - 1) - \delta_{n,v^*}(k+1) \leq 0$, for $\epsilon \in [1, \frac{\rho+1}{\rho}]$. In this way, $F_n(k)$ is a non-increasing function. With this conclusion, we further prove Lemma 3. We know $\eta^*_{n,v}(k) \leq 1$ and thus $F_n(k) \geq (\rho - 1) \cdot \sum_{v \in V} \epsilon^{\eta_{n,v}(k)}$. Combining the non-increasing of $F_n(k)$, we have $\eta_{n,v}(k) = \log_\epsilon(\epsilon^{\eta_{n,v}(k)}) \leq \log_\epsilon(\frac{1}{\rho-1} F_n(k)) \leq \log_\epsilon(\frac{1}{\rho-1} F(0)) \leq \log_\epsilon(\frac{\rho}{\rho-1} \cdot |V(0)|) = \log_\epsilon(\frac{\rho}{\rho-1} \cdot |V|)$. In this way, line 18 of Algorithm 4 will not be violated and thus Lemma 3 is proved. □

With Lemma 3, we now prove Theorem 4.

Proof. According to lines 3-7, load on any excluded server is smaller than the maximal load among the remaining servers after Algorithm 4. Therefore, $M^\dagger$ is the maximal load among remaining servers. Suppose the maximal load increment in iteration n is Δ_n. When $N = 1$, since $\Delta_1 \leq \log_\epsilon(\frac{\rho}{\rho-1} \cdot |V|) \cdot M_1$ and $M^* \geq M_1$, we have $\frac{M^\dagger}{M^*} \leq 1 + \log_\epsilon(\frac{\rho}{\rho-1} \cdot |V|)$. When $N > 1$, we have $\Delta_n \leq \log_\epsilon(\frac{\rho}{\rho-1} \cdot |V|) \cdot 2^{n-1} \cdot M_1$. Then, $M^\dagger \leq M_1 + \sum_{n=1}^{N} \Delta_n \leq M_1 + \sum_{n=1}^{N} \log_\epsilon(\frac{\rho}{\rho-1} \cdot |V|) \cdot 2^{n-1} \cdot M_1$. We know $M^* > 2^{N-2} \cdot M_1$, since Algorithm 4 will stop at $N-1$ otherwise. We then have $\frac{M^\dagger}{M^*} \leq \frac{M_1 + \sum_{n=1}^{N} \log_\epsilon(\frac{\rho}{\rho-1} \cdot |V|)) \cdot 2^{n-1} \cdot M_1}{2^{N-2} \cdot M_1} \leq 1 + 4 \log_\epsilon(\frac{\rho}{\rho-1} \cdot |V|))$. □

The proof of Theorem 4 indicates that the number of iterations N is bounded by $\mathcal{O}(\log(M^*))$. Suppose γ_{max} is the largest resource demand of dynamic backups and r_{min} is the the minimal server resources, we have $M^* \leq M_1 + \frac{|K| \cdot \gamma_{max}}{r_{min}}$. Thus, N is bounded by $\mathcal{O}(\log(M_1 + \frac{|K| \cdot \gamma_{max}}{r_{min}}))$. In addition, the complexity of each iteration is $\mathcal{O}(|K||V| \log(|V|))$, In this way, the computational complexity of Algorithm 4 is $\mathcal{O}(|K||V| \log(|V|) \cdot \log(M_1 + \frac{|K| \cdot \gamma_{max}}{r_{min}}))$.

Related work proposes algorithms for online load balancing problems, e.g., [29], [30]. Different from their work, Algorithm 4 balances the load under reliability constraints of dynamic backups, i.e., Constraint (10). In addition, the proof of Theorem 4 considers a shrinking server set and applies more generalized parameters compared with previous work. In this way, our algorithm covers a wider range of situations while preserving a theoretical performance guarantee.

5.3 Availability Reinforcement and Dynamic Backup Release

If two of the three copies (i.e., the VNF, the static backup, and the dynamic backup) fail, the dynamic backup deployment method will deploy the second dynamic backup. By doing so, the dynamic backup deployment always makes sure there are at least two functioning instances at a time for each VNF. If the availability (previous time operating normally/previous operating time) of a particular SFC does not reach its availability requirement, all dynamic backups of this SFC will not be released at recovery and work like static backups to reinforce the availability. These semi-static backups will be released when the availability is satisfied. In this way, we dynamically guarantee the availability of SFCs backed up on the edge.

5.4 SFC Backup Adjustment

During the deployment of dynamic backups, there exist situations that there are not enough resources to deploy the next dynamic backup. This means more VNFs are currently unavailable (e.g., a burst of failures) on the edge, and we need more resources to deploy dynamic backups. There are also opposite cases that edge resources are excessive and more SFCs can be backed up on the edge to save backup cost. To deal with these situations, RAD further moves backups of SFCs between the edge and the cloud to balance the availability and backup cost.

When edge resources are not sufficient, Algorithm 4 will be terminated. We sort SFCs backed up on the edge and pick the SFC f with the smallest w_f. We then back up SFC f in the cloud instead. After the backup of SFC f is established in the cloud, all static and dynamic backups of VNFs of f are released from the edge and Algorithm 4 starts again. Since SFC f is now backed up in the cloud, its availability is taken care of by the cloud. Meanwhile, more resources are available for dynamic backups of the SFCs still backed up

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2020.3048885, IEEE Transactions on Mobile Computing

on the edge. Due to these reasons, the availability of all SFCs is increased. Above steps will be repeated until reaching the point that Algorithm 4 can be processed continuously.

The adjustment method also maintains t, the running time of Algorithm 4 without termination. When t grows larger than a predefined threshold τ_1, the method deploys static *backups* for each VNF of SFC f which has the largest w_f and has been moved to the cloud previously. If t continuously increases and becomes larger than another threshold τ_2, all copies of SFC f in the cloud are released and f is placed on the edge again. In this way, the backup cost is reduced. Smaller τ_1 and τ_2 will make the backup adjustment more sensitive and reduce the backup cost whenever failure rates decrease. However, the backup adjustment also introduces extra cost (e.g., cost of VNF migration between edge and cloud). Therefore, we choose relatively large τ_1 and τ_2 in Section 6 so that the extra cost can be omitted compared with the backup cost.

5.5 Overhead Analysis of the RAD Scheme

As mentioned in previous sections, the overheads of SFC deployment, static backup deployment and dynamic backup deployment are $|F||I||V|$, $|F_{edge}|^2|V|$ plus solving an LP problem, and $\mathcal{O}(|K||V|\log(|V|)\cdot\log(M_1+\frac{|K|\cdot\gamma_{max}}{r_{min}}))$, respectively. The SFC deployment algorithm and the static deployment algorithm only need to be executed once throughout the RAD scheme. Although the total overhead of the dynamic backup deployment algorithm is determined by $|K|$, the number of failures, the executing time of deploying each dynamic backup is often covered by the time waiting for the next failure to happen. Since failure rates are low in most cases, time intervals between failures are often long enough to deploy a dynamic backup before the next failure happens. Similarly, the overhead of SFC backup adjustment which is determined by the speed of migrating VNFs is also covered by the time of waiting VNF failures in general. In this way, the general overhead of the RAD scheme is mild and affordable under most scenarios.

6 Performance Evaluation

In this section, we first evaluate the performance of RAD with extensive simulations. We then conduct small-scale experiments to demonstrate its feasibility and effectiveness in real-world cases.

6.1 Simulation Setup

For the simulation setup, we refer to basic settings of the VNF platform on commercial servers in [24]. We consider each edge server as a commercial server with a single CPU of 6 cores. As edge servers may have tasks other than VNFs, we assume each server randomly has 1 to 6 cores available (uniformly distributed). In the network, each SFC is randomly chained up by 1 to 5 VNFs. We focus on CPU utilization to represent the VNF resource demand (e.g., a VM uses 0.6 CPU core). In our simulations, we use the real-world trace of VM CPU utilization in MS Azure [31]. The numbers of SFCs and edge servers vary in different simulation sets to show the advantages of RAD from different aspects. Therefore, the distributions of normal SFCs and backups also change from case to case due to the deployment algorithms in RAD. For instance, when the number of SFCs ranges from 30 to 50 and there are 20 edge servers available, the average ratios of normal SFCs and static backups to the total edge resources are 69.6% and 25.3% over 100 repeated simulations. When edge servers increase to 60, the ratios change to 26.4% and 24.2%. More details will be illustrated along with simulation figures. The distributions of dynamic backups change over time and adapt to real-time failures, which can be reflected by the following Fig 5(b). The backup cost of an SFC depends on its delay requirement and resource utilization. We assign each SFC a random number uniformly distributed in the range $[1, 5]$ to represent the relative tightness in delay requirements. We then multiply this number with the total resource demand of all VNFs in this SFC to obtain its backup cost.

6.2 Performance of SFC Deployment Algorithms

We first show the performance of our SFC deployment algorithms, i.e., Tight-SFC and Loose-SFC, under corresponding resource assumptions in Theorem 1 and 2. Fig. 2(a) shows simulation results when the edge server with the minimal resources can hold all VNFs of the largest SFC. Lines in red and blue show resource utilization rates of Tight-SFC and Loose-SFC respectively when the number of edge servers increases. Each node represents the average result of 100 simulations with 80-120 SFCs. The black dashed line represents the total resource ratio which is the average ratio between the resources needed by all SFCs and the total edge resources. We observe that all three lines converge with the increase of edge resources and start to unify when the total resource ratio is below 79.8%. This means that when the assumption in Theorem 1 is satisfied, both algorithms can deploy all SFCs on the edge if the total resource demand of SFCs is less than 79.8% of edge resources.

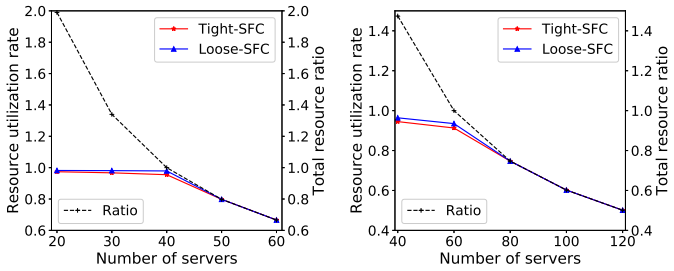


Fig. 2. The edge resource utilization rates of Tight-SFC and Loose-SFC when assumptions in Theorem 1 and Theorem 2 stand. (a) shows the resource utilization rates of both algorithms, when the edge resources increase and always satisfy the assumption in Theorem 1. (b) shows the resource utilization rates of both algorithms, when the edge resources increase and always satisfy the assumption in Theorem 2.

Fig. 2(b) shows the edge resource utilization of both algorithms but under a looser condition that the edge server with the minimal resources can hold the largest VNF on any SFC. It is clear that three lines also converge and start to unify at the resource ratio of 74.7%. This indicates that when the assumption in Theorem 2 is satisfied, both algorithms can deploy all SFCs on the edge with the total resource

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2020.3048885, IEEE Transactions on Mobile Computing

demand of SFCs less than 74.7% of edge resources. Fig. 2 shows that both algorithms work much better than their theoretical bounds, i.e., 50% for Tight-SFC and 25% for Loose-SFC.

We further simulate Tight-SFC and Loose-SFC with real-world settings of SFCs and servers in Section 6.1 and compare their performance with that of a greedy baseline algorithm. As existing algorithms for SFC deployment either vary in the objective functions optimized [1]–[4] or do not focus on deploying SFC on the edge [20]–[22], it is hard to compare our comprehensive scheme directly to them. Instead, we use a greedy algorithm as the baseline. In this greedy algorithm, whenever an SFC arrives, the server with the most available resources is picked for the first VNF and this procedure iterates until all VNFs of this SFC are placed. If there are not enough resources in the edge networks for any of these VNFs, the SFC will be placed in the cloud.

Fig. 3(a) presents the average edge resource utilization rates of the greedy algorithm, Tight-SFC, and Loose-SFC with 80-120 SFCs and an increasing number of edge servers. We observe that both Tight-SFC and Loose-SFC outperform the greedy algorithm significantly. Fig. 3(b) presents the CDF of Tight-SFC, Loose-SFC and the greedy baseline over 1,000 simulations with 40 edge servers. We can observe from the figure that more than 95% simulations of Loose-SFC utilizes more than 90% total edge resources. Tight-SFC performs even better since it has more simulations with higher edge resource utilization rates. Both algorithms outperform the greedy algorithm.

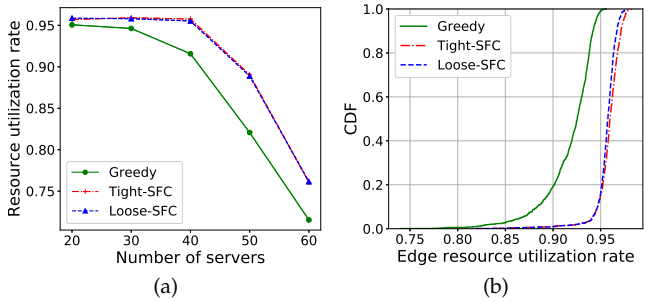


Fig. 3. The performance of Tight-SFC, Loose-SFC, and the greedy algorithm with real-world traces. (a) shows the average resource utilization rate of three algorithms with an increasing number of servers. (b) presents the CDF of three algorithms with 40 edge servers.

6.3 Performance of Static and Dynamic Backup Placement Algorithms

In this section, we present the evaluation of the backup placement algorithms. We conduct the following simulations with 20 edge servers in the network. For the static backup deployment, we again use a greedy algorithm as the baseline. With the VNF placed, the greedy algorithm deploys static backups starting from the SFC with the largest backup cost and follows a decreasing order in the backup cost. For each VNF backup of the SFC currently considered, the greedy algorithm places it on the server with the most available resources. If any VNF of an SFC cannot be backed up on the edge, the whole SFC is backed up in the cloud. Besides the greedy algorithm, we also use the solution of a relaxed LP problem (LP) [28] as another baseline. The relaxed LP solution serves as a lower bound of the optimal integral solution. We are doing this because it is computationally inefficient to solve the original integer problem directly. This provides a conservative comparison for our algorithms.

Fig. 4(a) shows the average backup cost of different algorithms with an increasing number of SFCs. The cost is the average value of 100 simulations. It is evident that our algorithm significantly reduces the backup cost compared to the greedy baseline, and its cost is not far from the relaxed optimal cost. In particular, when there are abundant resources for backups (with 10-30 SFCs and an average of 20.0%-53.9% load of SFCs), the static backup deployment algorithm reduces cost by 61.1%-50.9%. Even with limited edge resources for backups (e.g., with 50 SFCs and an average of 80.2% load on the edge before deploying backups), our algorithm can still save 15.9%.

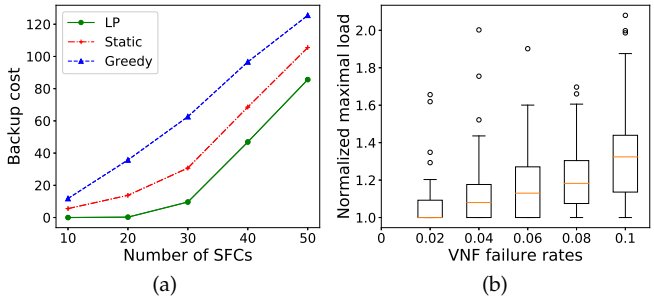


Fig. 4. Effectiveness of the proposed backup deployment algorithms. (a) illustrates the average backup cost of the LP, the static backup deployment algorithm, and the greedy baseline with an increasing number of SFCs. (b) shows the maximal load on edge servers from the dynamic backup deployment algorithm normalized by the offline optimum when failure rates increase.

We continue to evaluate the performance of the dynamic backup deployment algorithm with 30-50 SFCs deployed in the network. Each box in Fig. 4(b) concludes the maximal load on edge servers from the dynamic backup deployment algorithm in 100 simulations under a particular failure rate of VNFs. The results are normalized by offline optimal solutions solved by an ILP solver with all upcoming dynamic backups known in advance. Fig. 4(b) highlights that, in the majority of simulations, our algorithm achieves near-optimal performance. Even with a relatively large failure rate (10%), more than 75% of simulations are below 1.4 times of the offline optimal solution. This means the dynamic backup deployment algorithm is effective in balancing the load on edge servers, and thus reducing resource contention in an online manner.

6.4 Benefits of the RAD Scheme

In this section, we evaluate the performance of the entire reliability-aware adaptive deployment scheme with numerous simulations. 30-50 SFCs are deployed in the network of 20 edge servers. We choose rather limited edge resources since backup schemes are more challenged with limited backup resources at the edge. In each time slot, we assume each VNF (original, static backup, or dynamic backup) has a failure rate, and the VNF is randomly working or failed according to the rate [11]–[15]. By default, we assume a VNF recovers after 10 time slots (the creation of a VNF takes 1

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2020.3048885, IEEE Transactions on Mobile Computing

time slot) and we vary the recover time and create time in Fig. 7 to evaluate its impacts. Note the create time of a dynamic backup is the time between its deployment and the moment it starts to work.

We compare our RAD scheme with two representative baselines, i.e., static backup scheme with h backups for each VNF (STA-h) and predictive backup scheme with prediction accuracy l (PRE-l). STA-h is similar to that of the greedy algorithm in Section 6.3 but deploys h static backups instead of one. The PRE-l scheme can predict an upcoming VNF failure utilizing methods such as early failure detection [10] or machine learning [32]. The detailed prediction method is out of the scope of our paper. However, the performance of predictive methods can be measured by the prediction accuracy l and higher l leads to better results. PRE-l in our simulations can correctly predict a VNF failure at the rate of l and get a backup copy ready before the failure happens. It can also predict the failure of the backup at the same rate and create a second backup.

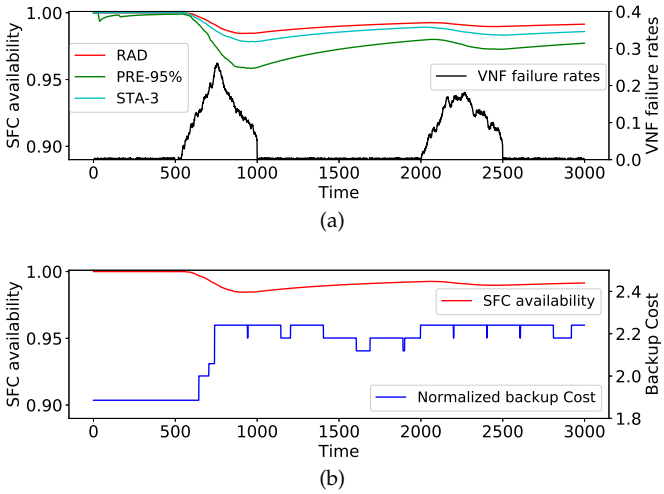


Fig. 5. Adaptive features of RAD. (a) The availability of SFCs with different schemes when failure rate spikes occur. (b) The change of backup cost with RAD when failure rate spikes happen.

We first demonstrate how our scheme adaptively balances the trade-off between the backup cost and the availability without knowing failure rates. Fig. 5(a) shows the availability of SFCs (averaged from the beginning to the current time slot) with RAD, PRE-95% and STA-3 during 3,000 time slots. There are two failure rate spikes shown by the black line. Clearly, the availability curve of RAD outperforms the baselines, meaning our scheme is more robust against bursts of VNF failures comparing with STA-3 using 3 static backups for each edge VNF or PRE-95% with high prediction accuracy of 95%.

Fig. 5(b) illustrates how RAD reacts to sudden failure rate spikes. In particular, when a spike happens, RAD creates more backups to handle the failures. When the spike ends, RAD waits until the time-averaged availability meets requirements before releasing the backups. Specifically, the backup cost increases when the failure rates start to burst. This is because RAD keeps deploying new dynamic backups to increase the availability against the failure rate spikes. More SFCs are thus backed up onto the cloud to release more edge resources for these dynamic backups. When failure rates fall back, the scheme does the opposite which releases unnecessary backups and reduces the backup cost. Considering the total running time as 3,000 time slots, we make the backup adjustment parameters $\tau_1 = 100$ and $\tau_2 = 50$ in our simulations.

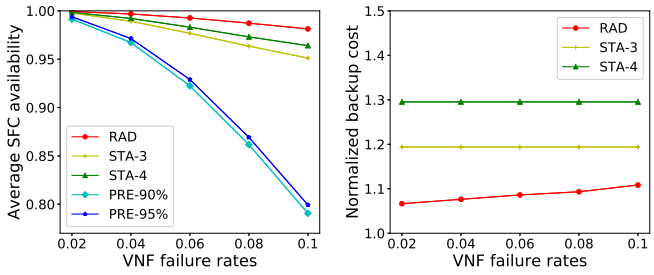


Fig. 6. Performance of RAD and baselines under different failure rates. (a) Average availability of SFCs with RAD, STA-h, and PRE-l when VNF failure rates increase. (b) Average backup costs applying RAD and STA-l. All backup costs are normalized by the static backup cost from the static backup deployment algorithm.

We then compare the performance of RAD with the baselines in extensive simulations. Each value in the following figures is the average of 100 independent simulations with 1000 time slots. Fig. 6(a) shows availability of SFCs applying different schemes when VNF failure rates increase. Fig. 6(b) presents corresponding backup costs of these schemes. We first find in Fig. 6(a) that RAD always preserves the highest SFC availability compared to baseline schemes and the superiority is more obvious with higher failure rates. We also observe that predictive backup schemes (PRE-l) degrade sharply with increasing failure rates even when the prediction accuracy is high, i.e., PRE-95%. Therefore, although they have little backup cost due to no deployment of static backups (omitted in Fig. 6(b)), predictive backup schemes cannot guarantee sufficient SFC availability as RAD does when VNF failures are more frequent or failure spikes happen. For the static backup scheme (STA-h), we find in Fig. 6(a) that larger h, i.e., the number of static backups for each VNF, leads to higher SFC availability. It is possible for STA-h to have comparable SFC availability to RAD if h is larger than 4. However, such availability improvement is at the cost of much higher backup cost according to Fig. 6(b)). In this way, we can conclude from Fig. 6 that our RAD scheme achieves the highest SFC availability with moderate backup cost and outperforms both baselines.

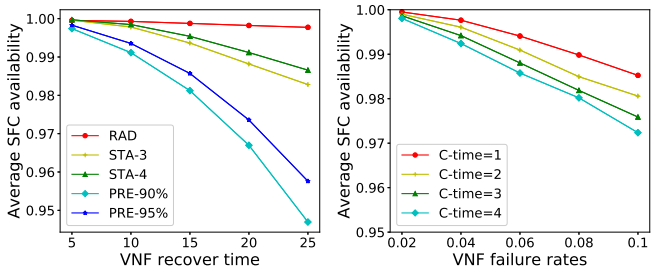


Fig. 7. Impacts of the recover time and the creation time to RAD. (a) The average availability of SFCs with RAD and the baselines when the recover time grows. (b) The average availability of SFCs applying RAD with different creation time.

We further evaluate how the recover time and the creation time affect the RAD scheme. Fig. 7(a) shows the performance of RAD and the baselines when the recover

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2020.3048885, IEEE Transactions on Mobile Computing

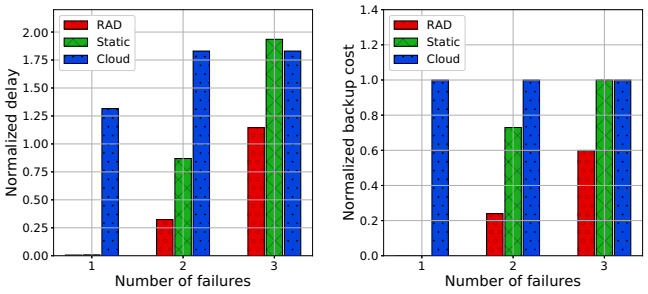
time increases. Failure rates of VNFs are 0.02 in this set of simulations. Although availability of SFCs achieved by all backup schemes decreases when the recover time increases, RAD degrades with the slowest speed. This indicates that RAD can tolerant slower VNF recoveries and guarantee high SFC availability compared to static and predictive backup schemes. Since the creation time of a VNF is influenced by multiple factors (e.g., VNF type, resource demand, and load of the edge server), we define a C-time representing the range within which the creation time is uniformly distributed. For instance, C-time = 4 means the creation time of each dynamic backup is uniformly distributed between 1 and 4 slots. Fig. 7(b) shows the performance of RAD with different creation time. With longer creation time, the average SFC availability of RAD decreases. However, even with larger creation time, the average SFC availability of our scheme is still much better than that of the baselines according to Fig. 6(a). In summary, RAD is applicable and performs much better than the baselines when VNF recover and creation time is relatively large.

6.5 Experiment

We conduct small-scale experiments with two types of real-world service function chains. The first SFC (SFC 1) consists of two functions, i.e., an image transcoder and an image classifier. The image transcoder transforms input images from png to jpg. The processed images then go through the image classifier to recognize their content. On the second SFC (SFC 2), a video transcoding function is followed by an action recognition function. Input videos are first transcoded from mjpeg to h.264 then processed by the action recognizer to identify actions in the videos. Details of these functions can be found in [33]–[35].

For each service chain, we apply three different backup strategies, our RAD scheme, the baseline with static backups at the edge (Static), and the baseline with only cloud backups (Cloud). For the RAD strategy proposed in this paper, the SFC, static backups, and dynamic backups are supported by three LXC containers [36]. Containers for the SFC and static backups can hold both VNFs while the container for dynamic backups has limited resources only for one dynamic backup, i.e., it can either support VNF 1 or VNF 2. For the baseline Static, two containers hold the SFC and its two static backups, respectively. For the baseline Cloud, only one container holds the SFC. If any failure happens, the service is forward to static backups in the cloud. All containers are running on an edge server with an Intel Core i7-9700k processor and 32 GB memory. All cloud backups are supported by VMs from the Google cloud platform and each VM has 4 cores and 16GB memory.

Fig. 8 presents the performance of SFC 1 applying the three backup strategies when different numbers of synchronous failures occur. In our experiments, when neither static nor dynamic backups cover the VNF failures, the service is forwarded to the cloud. Backups on the cloud are thus initialized on the fly to cover this failure at the cost of longer delay. Extra delay is also introduced by transmitting data from VNFs to backups or initializing dynamic backups on the edge. In this way, the availability of SFCs applying different backup methods are measured by the extra delay

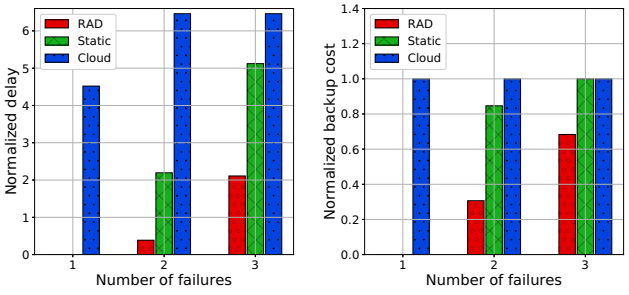


(a) (b)

Fig. 8. Performance of SFC 1 with different backup strategies when the number of synchronous failures increases. (a) shows the average delay introduced by different backup schemes. The delay time is normalized by the average execution time of the SFC with no failure. (b) shows corresponding backup costs for each backup strategy, normalized by the cost of deploying static backups in the cloud.

introduced by them. Fig. 8(a) shows the delay introduced by RAD and baselines when different numbers of failures happen at the same time. The value is normalized by the execution time of SFC 1 when no failure happens. It is clear that our RAD introduces the least delay under different failure conditions. Even when three failures happen at the same time on SFC 1 and its edge backups, RAD still reduces 40% delay compared to the baselines.

Fig. 8(b) presents backup costs of different backup strategies with varying numbers of failures. Since we apply pay-to-use cloud resources, the backup cost incurs in the cloud can be represented by the execution time of backups. The value is normalized by the cost of deploying static backups in the cloud, which run all the time when an SFC is in service. When two failures happen synchronously, RAD can save 67% backup cost compared to the static backup scheme by simply adding one dynamic backup on the edge. Even when rare cases happen with three failures at the same time, RAD achieves 40% lower backup cost compared with the baselines.



(a) (b)

Fig. 9. Performance of SFC 2 with different backup strategies when the number of synchronous failures increases. (a) shows the average delay introduced by different backup schemes. The delay time is normalized by the average execution time of the SFC with no failure. (b) shows corresponding backup costs for each backup strategy, normalized by the cost of deploying static backups in the cloud.

Fig. 9 shows corresponding performance of the backup strategies applied to SFC 2 processing videos. Advantages of RAD are also evident. Moreover, by comparing Fig. 8 with Fig. 9, we find that our RAD strategy introduces even less delay when applied to SFCs with larger input data size. For instance, in the same failure condition (two failures at the same time), RAD can save 63% delay for SFC 1 dealing with

images while saving at least 83% delay for SFC 2 processing videos of larger sizes. Experiment results from both figures highlight that the RAD scheme can work efficiently in real-world cases and significantly reduce the backup cost while guaranteeing the availability of SFCs on the edge compared to the baselines.

7 Related Work

Despite the popularity of VNF and SFC, how to guarantee the availability is considered as a key issue and has drawn much research attention [10]–[15], [23], [24], [37]. Much related work is committed to improving the availability of VNFs in the cloud and the majority of them focuses on using redundancy. Fan. et al. [11], [12] propose a scheme to minimize the total number of backups while meeting availability requirements. Zhang et al. [23] propose a novel method pursuing a similar goal while considering the heterogeneous resource demands of VNFs. Kanizo et al. [14], instead, maximize the availability with a given number of backups taking advantage of the resource-sharing ability of VNFs. All the work assumes the knowledge of failure rates is known, while our RAD scheme does not assume it. In addition, RAD provides performance guarantees for both approximation ratio and competitive ratio.

When considering the availability of SFCs chained up by multiple VNFs, Beck et al. [37] propose algorithms to provide backup VNFs and links parallel to the VNF. Shang et al [13], [15] propose rerouting strategies to guarantee the availability of SFCs in case of node and link failures and take network congestion into consideration. Instead of introducing redundancy, Taleb [10] propose a novel alternative framework which ensures the resilience of SFCs using efficient and proactive restoration mechanisms. These schemes create replacing VNFs at the early detection of failures. Martins et al. [24] realize a virtual VNF platform called ClickOS which enables fast creation of VNFs. In this paper, we combine the ideas of backups and fast creation of dynamic backups and propose the RAD scheme.

When it comes to deploying VNF on the edge, work in [5]–[9] has proposed algorithms and systems to explore its potential in multiple aspects. To guarantee the availability of VNFs on the edge, Zhu et al. [25] propose methods to track the availability and cost impact and place VNFs on edge networks considering both resource cost and application availability. Yala et al. [26] propose a VNF placement scheme between the edge and the cloud to optimize the trade-off between availability and latency. The work above only considers placing VNFs without deploying backups, thus may leading to infeasible solutions with longer service chains and lower availability. Dinh et al. [17] propose a cost-efficient redundancy allocation scheme for VNFs based on measuring the availability improvement potential of each VNF. However, as far as we are concerned, none of them consider the trade-off between the availability of SFCs and the backup cost when failure rates of VNFs are unknown and vary over time.

8 Conclusion

In this paper, we propose a reliability-aware adaptive deployment scheme RAD to find the sweet spot between the SFC availability and backup costs without assuming VNF failure rates. RAD first deploys SFCs over the edge and the cloud to maximize edge resource utilization. It simultaneously runs two sub-algorithms with constant theoretical bounds and picks the better output as the SFC deployment. RAD then uses a low complexity algorithm with a theoretical bound to deploy one static backup for each VNF while minimizing the backup cost. It further deploys dynamic backups using an online algorithm with a provable competitive ratio to guarantee the desired availability of SFCs backed up on the edge. RAD also adjusts backups between the edge and the cloud dynamically to accommodate the fluctuations in VNF failure rates and edge resource availability. Simulation results highlight that the proposed scheme fully utilizes edge resources, significantly reduces the backup cost, and guarantees the availability without predicting VNF failure rates. Small-scale experiments also show that RAD is feasible and efficient in real-world cases.

Acknowledgments

This research work was supported in part by the U.S. National Science Foundation under grant numbers CCF-1526162 and CCF-1717731.

References

[1] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive vnf scaling and flow routing with proactive demand prediction," in *IEEE INFOCOM 2018-IEEE International Conference on Computer Communications*. IEEE, 2018, pp. 486–494.

[2] X. Zhang, C. Wu, Z. Li, and F. C. Lau, "Proactive vnf provisioning with multi-timescale cloud resources: Fusing online learning and online optimization," in *IEEE INFOCOM 2017-IEEE International Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[3] S. Agarwal, F. Malandrino, C.-F. Chiasserini, and S. De, "Joint vnf placement and cpu allocation in 5g," in *IEEE INFOCOM 2018-IEEE International Conference on Computer Communications*. IEEE, 2018, pp. 1943–1951.

[4] C. Pham, N. H. Tran, S. Ren, W. Saad, and C. S. Hong, "Traffic-aware and energy-efficient vnf placement for service chaining: Joint sampling and matching approach," *IEEE Transactions on Services Computing*, vol. 13, no. 1, pp. 172 – 185, 2020.

[5] R. Cziva and D. P. Pezaros, "Container network functions: Bringing nfv to the network edge," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 24–31, 2017.

[6] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic, latency-optimal vnf placement at the network edge," in *IEEE INFOCOM 2018-IEEE International Conference on Computer Communications*. IEEE, 2018, pp. 693–701.

[7] A. Boubendir, E. Bertin, and N. Simoni, "On-demand, dynamic and at-the-edge vnf deployment model application to web real-time communications," in *CNSM 2016 International Conference on Network and Service Management*. IEEE, 2016, pp. 318–323.

[8] C. K. Dominicini, G. L. Vassoler, L. F. Meneses, R. S. Villaca, M. R. Ribeiro, and M. Martinello, "Virtphy: Fully programmable nfv orchestration architecture for edge data centers," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 817–830, 2017.

[9] Q. Zhang, F. Liu, and C. Zeng, "Adaptive interference-aware vnf placement for service-customized 5g network slices," in *IEEE INFOCOM 2019-IEEE International Conference on Computer Communications*. IEEE, 2019, pp. 2449–2457.

[10] T. Taleb, A. Ksentini, and B. Sericola, "On service resilience in cloud-native 5g mobile systems," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 3, pp. 483–496, 2016.

[11] J. Fan, C. Guan, Y. Zhao, and C. Qiao, "Availability-aware mapping of service function chains," in *IEEE INFOCOM 2017-IEEE International Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[12] J. Fan, M. Jiang, O. Rottenstreich, Y. Zhao, T. Guan, R. Ramesh, S. Das, and C. Qiao, "A framework for provisioning availability of nfv in data center networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2246–2259, 2018.

[13] X. Shang, Z. Li, and Y. Yang, "Placement of highly available virtual network functions through local rerouting," in *IEEE MASS 2018-IEEE International Conference on Mobile Ad Hoc and Sensor Systems*. IEEE, 2018, pp. 80–88.

[14] Y. Kanizo, O. Rottenstreich, I. Segall, and J. Yallouz, "Optimizing virtual backup allocation for middleboxes," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2759–2772, 2017.

[15] X. Shang, Z. Li, and Y. Yang, "Partial rerouting for high-availability and low-cost service function chain," in *IEEE GLOBECOM 2018-IEEE Global Communications Conference*. IEEE, 2018, pp. 1–6.

[16] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[17] N.-T. Dinh and Y. Kim, "An efficient availability guaranteed deployment scheme for iot service chains over fog-core cloud networks," *Sensors*, vol. 18, no. 11, p. 3970, 2018.

[18] S. Chen, F. Qin, B. Hu, X. Li, and Z. Chen, "User-centric ultra-dense networks for 5g: Challenges, methodologies, and directions," *IEEE Wireless Communications*, vol. 23, no. 2, pp. 78–85, 2016.

[19] X. Shang, Y. Huang, Z. Liu, and Y. Yang, "Reducing the service function chain backup cost over the edge and cloud by a self-adapting scheme," in *IEEE INFOCOM 2020-IEEE International Conference on Computer Communications*. IEEE, 2020, pp. 2096–2105.

[20] Y. Sang, B. Ji, G. R. Gupta, X. Du, and L. Ye, "Provably efficient algorithms for joint placement and allocation of virtual network functions," in *IEEE INFOCOM 2017-IEEE International Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[21] X. Shang, Z. Liu, and Y. Yang, "Network congestion-aware online service function chain placement and load balancing," in *ICPP 2019 International Conference on Parallel Processing*, 2019, pp. 1–10.

[22] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *IEEE INFOCOM 2015-IEEE International Conference on Computer Communications*. IEEE, 2015, pp. 1346–1354.

[23] J. Zhang, Z. Wang, C. Peng, L. Zhang, T. Huang, and Y. Liu, "Raba: Resource-aware backup allocation for a chain of virtual network functions," in *IEEE INFOCOM 2019-IEEE International Conference on Computer Communications*. IEEE, 2019, pp. 1918–1926.

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

[25] H. Zhu and C. Huang, "Edgeplace: Availability-aware placement for chained mobile edge applications," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 11, pp. 1–20, 2018.

[26] L. Yala, P. A. Frangoudis, and A. Ksentini, "Latency and availability driven vnf placement in a mec-nfv environment," in *IEEE GLOBECOM 2018-IEEE Global Communications Conference*. IEEE, 2018, pp. 1–7.

[27] J. Kang, O. Simeone, and J. Kang, "On the trade-off between computational load and reliability for network function virtualization," *IEEE Communications Letters*, vol. 21, no. 8, pp. 1767–1770, 2017.

[28] S. Mitchell, M. OSullivan, and I. Dunning, "Pulp: A linear programming toolkit for python," *The University of Auckland, Auckland, New Zealand*, 2011.

[29] Y. Azar, B. Kalyanasundaram, S. Plotkin, K. R. Pruhs, and O. Waarts, "Online load balancing of temporary tasks," in *Workshop on Algorithms and Data Structures*. Springer, 1993, pp. 119–130.

[30] K. Han, Z. Hu, J. Luo, and L. Xiang, "Rush: Routing and scheduling for hybrid data center networks," in *IEEE INFOCOM 2015-IEEE International Conference on Computer Communications*. IEEE, 2015, pp. 415–423.

[31] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *ACM SOSP 2017-ACM Symposium on Operating Systems Principles*. ACM, 2017, pp. 153–167.

[32] H. Huang and S. Guo, "Proactive failure recovery for nfv in distributed edge computing," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 131–137, 2019.

[33] GluonCV, "Gluoncv: A deep learning toolkit for computer vision," Website, https://gluon-cv.mxnet.io/contents.html.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE CVPR 2016-IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[35] C. Feichtenhofer, H. Fan, J. Malik, and K. He, "Slowfast networks for video recognition," in *IEEE ICCV 2019-IEEE International Conference on Computer Vision*, 2019, pp. 6202–6211.

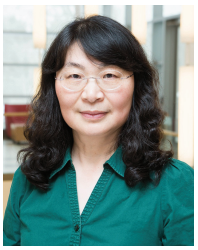
[36] LXC, "Linux containers," Website, https://linuxcontainers.org/.

[37] M. T. Beck, J. F. Botero, and K. Samelin, "Resilient allocation of service function chains," in *IEEE NFV-SDN 2016-IEEE Conference on Network Function Virtualization and Software Defined Networks*. IEEE, 2016, pp. 128–133.

Xiaojun Shang received his B. Eng. degree in Information Science and Electronic Engineering from Zhejiang University, Hangzhou, China, and M.S. degree in Electronic Engineering from Columbia University, New York, USA. He is now pursuing his Ph.D. degree in Computer Engineering at Stony Brook University. His research interests are in cloud computing and data center networks, with focus on placement and routing of virtual network functions and resilience of service function chains.

Yaodong Huang received his B.E. in Computer Science and Technology in 2015 from University of Electronic Science and Technology of China. He is now pursuing his Ph.D. degree in Computer Engineering at Stony Brook University. His research interests are in mobile edge computing, with focus on data caching, storage, and blockchain technology over edge environments.

Zhenhua Liu is currently assistant professor in the Department of Applied Mathematics and Statistics, also affiliated with Department of Computer Science and Smart Energy Technology Cluster, since August 2014. During the year 2014-2015, he is on leave for the ITRI-Rosenfeld Fellowship in the Energy and Environmental Technology Division at Lawrence Berkeley National Laboratory. Dr. Liu received his Ph.D. degree in Computer Science at the California Institute of Technology, where he was co-advised by Prof. Adam Wierman and Prof. Steven Low. Before Caltech, he received an M.S. degree of Computer Science Technology in 2009 and a B.E. degree of Measurement control in 2006, both from Tsinghua University with honor, as well as a B.S. degree of Economics from Peking University in 2009.



Yuanyuan Yang received the BEng and MS degrees in computer science and engineering from Tsinghua University, Beijing, China, and the MSE and Ph.D. degrees in computer science from Johns Hopkins University, Baltimore, Maryland. She is a SUNY Distinguished Professor of computer engineering and computer science at Stony Brook University, New York, and is currently on leave at the National Science Foundation as a Program Director. Her research interests include edge computing, data center networks, cloud computing and wireless networks. She has published more than 460 papers in major journals and refereed conference proceedings and holds seven US patents in these areas. She is currently the Editor-in-Chief for IEEE Transactions on Cloud Computing and an Associate Editor for IEEE Transactions on Parallel and Distributed Systems and ACM Computing Surveys. She has served as an Associate Editor-in-Chief for IEEE Transactions on Cloud Computing, Associate Editor-in-Chief and Associated Editor for IEEE Transactions on Computers, and Associate Editor for IEEE Transactions on Parallel and Distributed Systems. She has also served as a general chair, program chair, or vice chair for several major conferences and a program committee member for numerous conferences. She is an IEEE Fellow.