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

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

***Abstract*—The fast development of virtual network functions (VNFs) brings new opportunities to network service deployment on edge networks. For complicated services, VNFs can chain up to form service function chains (SFCs). Despite the promises, it is still not clear how to backup VNFs to minimize the cost while meeting the SFC availability requirements in an online manner. In this paper, we propose a novel self-adapting scheme named SAB to efficiently backup VNFs over both the edge and the cloud. Specifically, SAB uses both static backups and dynamic ones created on the fly to accommodate the resource limitation of edge networks. For each VNF backup, SAB determines whether to place it on the edge or the cloud, and if on the edge, which edge server to use for load balancing. SAB does not assume failure rates of VNFs but instead strives to find the sweet point between the desired availability of SFCs and the backup cost. Both theoretical performance bounds and extensive simulation results highlight that SAB provides significantly higher availability with lower backup cost compared with existing baselines.**

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

I. Introduction

The emerging of virtual network functions (VNFs) decouples network functions from dedicated hardware so commercial servers can be used. In order to improve the elasticity, flexibility and scalability of running VNFs over commercial servers, much research has been conducted, e.g., [1]–[4]. With the rapid development of edge computing and 5G networks, it is promising to deploy VNFs over edge networks to further realize the potentials of VNFs together with the upcoming 5G technologies. See [5]–[9] as examples.

Despite such benefits mentioned above, detaching the network function software from the underlying dedicated hardware makes it challenging to guarantee high availability [10]–[12]. Specifically, in most existing 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 renders 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].

Adding VNF backups is an effective way to improve the availability of SFCs and has been widely used for availability in the cloud [11]–[15]. However, effective schemes for the cloud face new challenges when SFCs are deployed on the edge and therefore may not work well in edge environments. 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 which are dynamic over time. A thoughtful SFC backup scheme for edge networks need to consider all the trade-offs to decide when to backup 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 [16], [18]. In addition, 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 hard to predict due to various failure types and causes [10].

In this paper, we propose a novel self-adapting backup scheme named SAB consisted of two steps to find the sweet spot between backup cost and availability without assuming the failure rate of each VNF. In the first step, SAB deploys one static backup for each VNF and determine where to place the backups to minimize the backup cost with edge resource constraints. For those SFCs backed up on the cloud, their availability is taken care of by cloud service providers through Service Level Agreements [12]. For those SFCs backed up on the edge, one static backup for each VNF may not be sufficient to meet the availability requirements [19]. This problem is solved in the next step.

In the second step of SAB, we deploy dynamic backups to achieve the required availability of SFCs. A dynamic backup is deployed as soon as an original VNF or its static backup fails. It is released when the corresponding VNF recovers. The fast creation of a dynamic backup has been verified by existing VNF platforms, e.g., [20], and can be accelerated by methods such as early VNF failure detection [10]. In addition, with the existence of the static backup for each VNF in the first step, a dynamic backup has enough time to set up unless both the original VNF and the static backup fail successively within a very short time. This makes the dynamic backup method more effective than creating copies only when original VNFs fail.

As most VNFs recover after some short time [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 edge.

In the case of sudden failure rate spikes, SAB moves backups of SFCs with lower backup cost from the edge to the cloud, thus releasing more resources to accommodate additional dynamic backups in the edge. On the contrary, the scheme backs up more SFCs on the edge to reduce cloud backup cost when failure rates decrease. The 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], [21], [22], to the best of our knowledge, no existing method optimizes both static and dynamic backups over both the edge and the cloud without assuming the failure rate of each VNF.

Our main contributions are summarized as follows.

- We design an efficient scheme SAB which reduces backup costs and guarantees the availability of SFCs on the edge without assuming future VNF failures. This scheme contains both the static backup placement and the dynamic backup deployment.
- For the static backup placement problem which is at least as complex as an NP-complete problem, we propose an algorithm with low computational complexity and prove its approximation ratio.
- As dynamic backups need to be deployed in real time, we propose an online algorithm and prove its competitive ratio compared to the offline optimal solution. Our scheme further adjusts VNF backups between the edge and the cloud based on the feedback from the online algorithm.
- We conduct real-world trace-driven numerical simulations and small-scale experiments. Results highlight that our proposed scheme provides availability guarantee for SFCs on the edge and reduces the backup cost at the same time, even without assuming the VNF failure rates.

The remainder of this paper is organized as follows. Section II presents an overview of the self-adapting backup scheme. Section III proposes the model and algorithm for the static backup deployment while Section IV continues with the dynamic backup deployment and the SFC backup adjustment. Numerical results are presented in Section V. Section VI briefly reviews the related work and Section VII concludes this paper.

II. Overview of the Self-adapting Backup Scheme

We use Fig. 1 as a brief demonstration of SAB. In the first step, we propose a static backup deployment method to deploy one static backup for each VNF. We do not consider cases when some VNFs of an SFC are backed up on the edge and others backed up in the cloud. The reason is, in our application scenario, the delay between the edge and the cloud is larger than that between edge servers, especially in the emerging 5G environment [16], [18]. Partially backing up an SFC in the cloud may cause the service flow traveling up and down between original VNFs on the edge and backups in the cloud multiple times. This process will incur large delay and add significant traffic to the network.

The static backup deployment determines the placement of each static backup aiming to minimize the backup cost without violating the resource limitation of any edge server. The detailed deployment method is illustrated in Section III. The first step with the static backup deployment serves as the premise of the second step, i.e., dynamic backup deployment and SFC backup adjustment. It provides one static backup to each VNF on the edge, which allows the creation of dynamic backups when original VNFs fail.

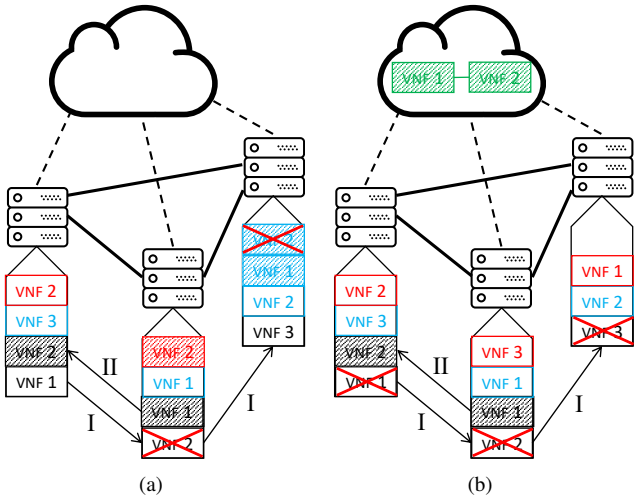


Fig. 1. The demonstration of SAB. The black VNFs connecting one another represent SFCs currently backed up on the edge. SFC I has 3 VNFs and SFC II has 2 VNFs (shadowed). SAB deploys a static backup (blue) for each VNF. Whenever an original VNF or static backup fails (red cross), SAB deploys a dynamic backup (red) for that VNF. If more failures occur (VNF 1, 2, and 3 of SFC I), the remaining resources are not sufficient for more dynamic backups in (a). SAB then backs up SFC II (shadowed) from the edge to the cloud and free more resources for the dynamic backups of SFC I. The result is shown in (b).

With the first step, some SFCs are backed up onto the cloud. For these SFCs, their availability 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 original 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 I, it is difficult to decide how many backups a 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.

In the second step, we use a dynamic backup deployment method which creates a dynamic backup whenever a VNF or its static backup just fails. Each dynamic backup is placed to an edge server using an online algorithm. Online optimization has been recently applied in edge computing [23], [24] and cloud computing [25], [26]. The algorithm balances the load

among each edge server in the long term to mitigate resource contention which is a main cause of VNF failures [27]. When both the original VNF and the static backup resume, the dynamic backup is released. However, if the current availability (previous working time/total running 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 usage 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 terminated. We then apply an SFC backup adjustment method to move the 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 move 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 IV.

III. Static Backup Deployment

In this section, we present the static backup deployment which deploys each VNF a static backup while minimizing the backup cost. Important notations used in this paper are summarized in Table I.

TABLE I

Notation	Definition
V	Set of servers on the edge, $V = \{1, ..., v, ..., \|V\|\}$
F	Set of SFCs on the edge, $F = \{1, ..., f, ..., \|F\|\}$
I_f	Set of VNFs of SFC f, $I_f = \{1, ..., i, ..., \|I_f\|\}$
$x_{f,i,v} \in \{0, 1\}$	Decision variable whether static backup i of SFC f is deployed on server v.
X	Set of all $x_{f,i,v}$, where $f \in F, i \in I_f, v \in V$
R_v	Resources on edge server v for VNFs
$U(X)$	Total cost of backing up SFCs
$\mathcal{R}(X) \leq \mathcal{D}$	Resource constraint of the edge network
w_f	Backup cost of SFC f
$\beta_{f,i}$	Resource demand of VNF i of SFC f
a_v	Resource demand on v before deploying static backups
$o_{f,i}$	Server holding the original VNF i of SFC f
$y_f \in \{0, 1\}$	Variable whether SFC f is backed up in the cloud.
$\tilde{x}, \tilde{y} \in [0, 1]$	Relaxed solution of $x_{f,i,v}$ and y_f
K	Set of dynamic backups $K = \{1, ..., k, ..., \|K\|\}$
$x_{k,v} \in \{0, 1\}$	Decision variable whether dynamic backup k is deployed on server v.
γ_k	Resource demand of the dynamic backup k
W_v	Load on v after deploying $\|K\|$ dynamic backups
b_v	Resource demand on v before dynamic backups
$o_{k,1}, o_{k,2}$	Servers holding original VNF and static backup of k

A. Model of the Static Backup Deployment

We suppose that the edge network consists of a set of servers denoted by $V = \{1, ..., v, ..., |V|\}$. Denote by R_v the resources available for VNFs on server v. We define the set of SFCs already deployed on the edge as $F = \{1, ..., f, ..., |F|\}$. All VNFs of SFC f form a set I_f with each VNF $i \in I_f$. For the deployment of static backups, we need to determine $x_{f,i,v}$, a binary variable denoting whether the backup of VNF i of SFC f is placed on server v. We use $U(X)$ to denote the total backup cost, where X is the set of all $x_{f,i,v}$. We minimize $U(X)$ subject to constraints of resources, reliability, and indivisibility of VNFs as follows.

$$\begin{aligned} \min \quad & U(X) \\ \text{s.t.} \quad & \mathcal{R}(X) \leq \mathcal{D}, && (1) \\ & x_{f,i,v} = 0, \quad \forall f \in F, \forall i \in I_f, v = o_{f,i}, && (2) \\ & x_{f,i,v} \in \{0, 1\}, \ \forall f \in F, \forall i \in I_f, \forall v \in V. && (3) \end{aligned}$$

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 I, the backup cost of SFC f is the extra cost incurred by deploying f in the cloud instead of on the edge. Denote the backup cost of SFC f by w_f, which depends on types and quantity of VNFs, the resource demand and the delay requirement of SFC f. We thus formulate

$$U(X) = \sum_{f \in F} w_f \cdot \max_{i \in I_f} \left(1 - \sum_{v \in V} x_{f,i,v}\right)^+.$$

Here, if any VNF i of SFC f is not backed up on the edge, i.e., $\sum_{v \in V} x_{f,i,v} = 0$, the whole SFC f is backed up onto the cloud, and the backup cost w_f is thus counted.

For resource constraint (1), we denote $\beta_{f,i}$ as the resource demand of VNF i of SFC f. The overall resource demand of VNFs on v cannot exceed the total VNF resources, R_v. Since original VNFs are running on edge servers, we denote a_v as the resource demand on v before deploying static backups. In this way, we formulate Constraint (1) as

$$\sum_{f \in F} \sum_{i \in I_f} \beta_{f,i} \cdot x_{f,i,v} \leq R_v - a_v, \ \forall v \in V.$$

For Constraint (2), denote by $o_{f,i}$ the node holding original 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 $x_{f,i,v} = 0$ for $v = o_{f,i}$. Constraint (3) simply means a VNF cannot be split.

The optimization problem can be transformed into an integer linear programming (ILP) problem as illustrated in Section III-B). In particular, if we ignore the reliability constraint and suppose each SFC only has one VNF, the problem becomes a 0-1 multiple knapsack problem which is NP-complete [28]. Therefore, when the edge network consists of a large number of servers and SFCs which is often the case, the computational overhead of solving the problem directly may be unaffordable. Motivated by this observation, we propose a static backup deployment algorithm to solve the problem with much lower complexity while guaranteeing a theoretical bound compared to the optimal solution.

B. Static Backup Deployment Algorithm

The detailed static backup deployment algorithm is presented in Algorithm 1. In the algorithm, we define a binary variable y_f and let $y_f = \max_{i \in I_f}(1 - \sum_v x_{f,i,v})^+$. The variable y_f represents whether an SFC 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 $\max_{i \in I_f}(1-\sum_v x_{f,i,v})^+$ in the objective function and formulate an equivalent ILP problem with both $x_{f,i,v}$ and y_f and an extra constraint

$$1 - \sum_{v \in V} x_{f,i,v} \leq y_f \quad \forall f \in F, \forall i \in I_f \tag{4}$$

By relaxing $x_{f,i,v} \in \{0,1\}$ and $y_f \in \{0,1\}$ to $\tilde{x}_{f,i,v} \in [0,1]$ and $\tilde{y}_f \in [0,1]$, we then get a linear programming (LP) problem. We solve the LP problem with an LP solver [29] to obtain relaxed solutions $\{\tilde{x}_{f,i,v}\}$ and $\{\tilde{y}_f\}$.

Next, we need to determine whether each $x_{f,i,v}$ and y_f should be 0 or 1 based on the relaxed solutions. In Algorithm 1, we first determine y_f and let the value of y_f determines the corresponding set of $x_{f,i,v}$ for the same SFC. If $y_{f'} = 1$ for a particular f', there must be at least one $i \in I_{f'}$ with $(1-\sum_v x_{f',i,v})^+ = 1$. Then we can make $(1-\sum_v x_{f',i,v})^+ = 1$ for all $i \in I_{f'}$. Since this will not change the objective but reduce the left side of Constraint (1) as all $x_{f',i,v} = 0$. If $y_{f'} = 0$, it is clear that $(1 - \sum_v x_{f',i,v})^+ = 0$ for all $i \in I_{f'}$. For a particular $i' \in I_{f'}$, since Constraint (1) is linear, we only choose one $v \in V$ and make $x_{f',i',v} = 1$ while satisfying Constraint (2). This is because multiple $x_{f',i',v} = 1$ will not reduce the objective function but increase the left side of Constraint (1). Based on the fractional result $\{\tilde{x}_{f,i,v}\}$, when $\sum_v x_{f',i',v} = 1$, we look for the $x_{f',i',v}$ with the largest $\tilde{x}_{f',i',v}$ among all v and make it 1. By following steps above, whenever the rounding of y_f is given, all corresponding $x_{f,i,v}$ are determined.

To determine $\{y_f\}$, we first sort all $\tilde{y}_f$ in a decreasing order. Denote by θ the threshold for rounding which always equals 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, $x_{f,i,v}$ with the largest $\tilde{x}_{f,i,v}$ among all v are set to 1. In each iteration, any y_f with $\tilde{y}_f = \theta$ is set to 1 and corresponding $x_{f,i,v}$ are set to 0. This means that Algorithm 1 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 continues until all constraints are satisfied for the first time. The total number of iterations is limited by $|F|$. 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 $f \in F$ and $y_f = 1$, we set y_f back to 0 and corresponding $x_{f,i,v}$ back to 1, if no constraint is violated by doing so.

In Algorithm 1, the complexity of getting corresponding $x_{f,i,v} = 1$ for every y_f is $\mathcal{O}(|F||I||V|)$. The process of determining y_f with the withdraw procedure totally takes $\mathcal{O}(|F||I|)$. Thus, the computational complexity of Algorithm 1 except solving an LP is $\mathcal{O}(|F||I||V|)$. We then prove that $\frac{1}{\theta}$ is a bound between the result of Algorithm 1 and that of the optimal solution.

Algorithm 1 Static Backup Deployment Algorithm

Input: $\{\tilde{x}_{f,i,v}\}$ and $\{\tilde{y}_f\}$ from the LP solver.
Output: Binary output $\{x_{f,i,v}\}$, $\{y_f\}$ and threshold θ
1: Initialize all $x_{f,i,v} = 0$.
2: **for all** $f \in F, i \in I_f$ **do**
3: **for all** $v \in V \setminus o_{f,i}$ **do**
4: **if** $\tilde{x}_{f,i,v} = \max_{v \in V \setminus o_{f,i}} \{\tilde{x}_{f,i,v}\}$ **then**
5: $x_{f,i,v} \leftarrow 1$, break.
6: **end if**
7: **end for**
8: **end for**
9: **while** Constraints are not satisfied **do**
10: $\theta \leftarrow \max_{f \in F}\{\tilde{y}_f\}$
11: **for all** $\tilde{y}_f = \theta$ **do**
12: $\tilde{y}_f \leftarrow 0$
13: $y_f \leftarrow 1$ and corresponding $x_{f,i,v} \leftarrow 0$
14: **end for**
15: **end while**
16: **for all** $f \in F$ **do**
17: **if** $y_f = 1$ and setting corresponding $x_{f,i,v}$ set to 0 in line 13 back to 1 does not violate any constraint **then**
18: $y_f \leftarrow 0$ and corresponding $x_{f,i,v} \leftarrow 1$.
19: **end if**
20: **end for**

Theorem 1. *Suppose the result of Algorithm 1 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 $y_f = \max_{i \in I_f}(1 - \sum_{v \in V} x_{f,i,v})^+$, the goal is to minimize $\sum_{f \in F} w_f \cdot y_f$. First, since the optimal solution of a relaxed problem is at least as good as the original ILP problem, we have

$$Z^* \geq \sum_f w_f \cdot \tilde{y}_f.$$

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

$$\sum_f w_f \cdot \tilde{y}_f \geq \sum_{\{\tilde{y}_f | \tilde{y}_f \geq \theta\}} w_f \cdot \tilde{y}_f \geq \theta \cdot \sum_{\{y_f | y_f = 1\}} w_f \cdot y_f.$$

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

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

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

IV. Dynamic Backup Deployment

When the first step of SAB is finished, 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 SFCs need more backups in an online manner. SAB deploys dynamic backups for SFCs with failed VNFs or static backups.

As mentioned in Section II, when deploying dynamic backups, we need to balance the load of each server to mitigate

resource contention. In addition, since VNF failures may occur successively over time, we need to deploy a dynamic backup as soon as a failure happens to guarantee the availability. 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 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.

A. Model of Dynamic Backup Deployment

For the deployment of dynamic backups, denote by K the set of dynamic backups arriving over time, where $K = \{1, ..., k, ..., |K|\}$. For simplicity, we assume no VNF recovers during this time which leads to the most resource contention. Dynamic backup k has a resource demand γ_k. We define a decision variable $x_{k,v}$ which denotes whether the dynamic backup k is deployed on server v. We also define a variable W_v representing the load (resource demand/resource capacity) on server 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. We formulate the dynamic backup deployment problem as follows.

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

$$\text{s.t.} \quad \frac{b_v + \sum_{k \in K} \gamma_k \cdot x_{k,v}}{R_v} = W_v, \quad \forall v \in V, \tag{5}$$

$$W_v \leq 1, \quad \forall v \in V, \tag{6}$$

$$\sum_{v \in V} x_{k,v} = 1, \quad \forall k \in K, \tag{7}$$

$$x_{k,v} = 0, \quad \forall k \in K, v \in \{o_{k,1}, o_{k,2}\}, \tag{8}$$

$$x_{k,v} \in \{0, 1\}, \ \forall k \in K, \forall v \in V. \tag{9}$$

With the deployment of static backups, we denote b_v as the current resource demand on v. Constraint (5) makes sure that W_v is the final load on node v after deploying all dynamic backups, which is the sum of original VNFs, static backups and dynamic backups over resources. Constraint (6) serves as the resource constraint and ensures that the load on any server is restricted by 1. Constraint (7) restricts that there must be one edge server holding each dynamic backup. Constraint (8) is the reliability constraint that dynamic backup k cannot be deployed on servers holding its original VNF or static backup represented by $o_{k,1}$ and $o_{k,2}$.

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 as follows.

B. Online Algorithm for Dynamic Backup Deployment

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 $\eta_{1,v} = 0$ for all v. M_1 is the largest load on any v with the current deployment of VNFs and 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 IV-D is triggered. However, this termination is very rare unless failure spikes happen as shown in Section V. We define an increment parameter $\delta^k_{n,v} = \frac{\gamma_k}{R_v \cdot M_n}$ for each remaining server v. We sort remaining servers in the 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 pick the first server v' in V' that $v' \notin \{o_{k,1}, o_{k,2}\}$. 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 $M_{n+1} = 2M_n$. The detailed process is presented in Algorithm 2.

Algorithm 2 Dynamic Backup Deployment Algorithm

Input: Set of edge servers V, current deployment of VNFs and backups, dynamic backup set K, ϵ and ρ.
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 request arrives, $overflow \leftarrow True$ and do lines 3-23.
3: **for all** $v \in V$ **do**
4: **if** $\frac{b_v}{R_v} + \sum_{j=1}^{k-1} \frac{\gamma_j}{R_v} \cdot x_{j,v} + \frac{\gamma_k}{R_v} > 1$ **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 the 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 first 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**

Denote the result of Algorithm 2 (the offline optimal solution) by $M^\dagger$ (M^*, respectively). We now prove that the result of Algorithm 2 is bounded by a competitive ratio to the offline optimum in Theorem 2.

Theorem 2. *Assume that Algorithm 2 has not excluded the optimal server for dynamic backup k when deploying it. We*

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

When $|V| > 1$, the bound in Theorem 2 is minimized if $\epsilon = \frac{\rho+1}{\rho}$ and ρ satisfies $\frac{\rho-1}{\rho} \cdot e^{\frac{\rho+1}{\rho-1}\log_\epsilon(\frac{\rho+1}{\rho})} = |V|$. The detailed value of ϵ and ρ can be obtained by well-established numerical methods. To prove Theorem 2, we first prove the following lemma.

Lemma 1. *If* $M^* \leq M_n$*, Algorithm 2 can finish the deployment of* $|K|$ *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 2 in iteration n til deploying backup k, $\Phi^*_v(k)$ is the set of backups deployed by the offline optimum on v til deploying backup k. Define $\eta^*_{n,v}(k) = \frac{\sum_{j \in \Phi_n(k) \cap \Phi^*_v(k)} \gamma_j}{R_v \cdot M_n}$. 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 2, we denote by $V(k)$ the set V when deploying dynamic backup k and have $V(k+1) \subseteq V(k)$. We first have

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

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

$$\begin{aligned} F_n(k+1) - F_n(k) \leq &\sum_{v \in V(k+1)} \epsilon^{\eta_{n,v}(k+1)} \cdot (\rho - \eta^*_{n,v}(k+1)) \\ &- \sum_{v \in V(k+1)} \epsilon^{\eta_{n,v}(k)} \cdot (\rho - \eta^*_{n,v}(k)) \end{aligned} \tag{10}$$

We define $v^\dagger$ and v^* as corresponding servers chosen by Algorithm 2 and offline algorithm to place dynamic backup $k+1$. According to the assumption in Theorem 2, we have $v^* \in V(k+1)$. When $v^\dagger$ and v^* are different, we simplify (10) and get

$$\begin{aligned} (10) = &(\epsilon^{\eta_{n,v^\dagger}(k+1)} - \epsilon^{\eta_{n,v^\dagger}(k)}) \cdot (\rho - \eta^*_{n,v^\dagger}(k)) \\ &-\epsilon^{\eta_{n,v^*}(k)} \cdot \delta_{n,v^*}(k+1) \end{aligned}$$

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

$$\begin{aligned} (10) = &(\epsilon^{\eta_{n,v}(k+1)} - \epsilon^{\eta_{n,v}(k)}) \cdot (\rho - \eta^*_{n,v}(k)) \\ &-\epsilon^{\eta_{n,v}(k+1)} \cdot \delta_{n,v}(k+1) \end{aligned}$$

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

$$\begin{aligned} (10) \leq &(\epsilon^{\eta_{n,v^\dagger}(k+1)} - \epsilon^{\eta_{n,v^\dagger}(k)}) \cdot (\rho - \eta^*_{n,v^\dagger}(k)) \\ &-\epsilon^{\eta_{n,v^*}(k)} \cdot \delta_{n,v^*}(k+1) \end{aligned} \tag{11}$$

We further have

$$(11) \leq \rho \cdot (\epsilon^{\eta_{n,v^\dagger}(k+1)} - \epsilon^{\eta_{n,v^\dagger}(k)}) - \epsilon^{\eta_{n,v^*}(k)} \cdot \delta_{n,v^*}(k+1)$$

Due to lines 16 and 17, we have $v^\dagger, v^* \notin \{o_{k,1}, o_{k,2}\}$ 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

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

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 1. 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 $\eta_{n,v}(k) = \log_\epsilon(\epsilon^{\eta_{n,v}(k)}) \leq \log_\epsilon(\frac{1}{\rho-1}F_n(k)) \leq \log_\epsilon(\frac{1}{\rho-1}\sum_{v \in V} \epsilon^0 \cdot (\rho - 0)) \leq \log_\epsilon(\frac{\rho}{\rho-1} \cdot |V|)$. In this way, line 18 of Algorithm 2 will not be violated and thus Lemma 1 is proved. □

With Lemma 1, we now prove Theorem 2.

Proof. 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 $\Delta_n \leq \log_\epsilon(\frac{\rho}{\rho-1} \cdot |V|) \cdot 2^{n-1} \cdot M_1$. According to lines 3-7, excluded servers have less load increment. 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 2 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 2 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 $M^* \leq M_1 + \frac{|K| \cdot \gamma_{max}}{r_{min}}$. So N is bounded by $\mathcal{O}(\log(M_1 + \frac{|K| \cdot \gamma_{max}}{r_{min}}))$. In addition, the complexity of each iteration is $\mathcal{O}(|K||V|\log(|V|))$, so the computational complexity of Algorithm 2 is $\mathcal{O}(|K||V|\log(|V|) \cdot \log(M_1 + \frac{|K| \cdot \gamma_{max}}{r_{min}}))$.

C. Availability Reinforcement and Dynamic Backup Release

If two of the three copies (i.e., the original 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 any time for each VNF. If the availability of a particular SFC does not reach its 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.

D. SFC Backup Adjustment

During the deployment of dynamic backups, there exist situations that edge resources are not enough for the next dynamic backup. In these cases, we terminate Algorithm 2 and start a SFC backup adjustment method. We sort SFCs backed up on the edge and pick the SFC f with the smallest w_f. We then back up SFC f into the cloud instead. After the backup of SFC f is established in the cloud, all static and dynamic backups of f are released from the edge. Since SFC f is now backed up in the cloud, its availability is taken care of by the cloud. Meanwhile, more resources are available to create dynamic backups for SFCs backed up on the edge. Therefore, the availability of all SFCs is increased. Above steps are repeated until that Algorithm 2 can start again.

The adjustment method also maintains t, the running time of Algorithm 2 without termination. When t exceeds a predefined threshold τ_1, the method tries to resume static backups of SFC f back to edge servers. f is the SFC adjusted into the cloud previously with the largest w_f. If there is no enough resource for this resumption, $t = 0$. If t keeps increasing and exceeds another threshold τ_2, all copies of SFC f in the cloud are released and f is backed up on the edge again. In this way, unnecessary backup cost is saved. The detailed choices of τ_1 and τ_2 are discussed in Section V.

V. Performance Evaluation

A. Simulation Setup

For the simulation setup, we refer to basic settings of the VNF platform on commercial servers in [20]. 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). We suppose that there are totally 20 servers in the edge network connecting to one another and all servers connected to the cloud. In the edge network, each SFC is randomly chained up by one to five VNFs. We focus on CPU utilization to represent the VNF resource demand (e.g., a VM uses 0.6 CPU core). In our simulation, we use the real-world trace of VM CPU utilization in MS Azure [30]. The backup cost of an SFC depends on its delay requirement and resource demand. We assign each SFC a random number uniformly distributed in $[1, 5]$ 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. To mitigate the influence of VNF migration between the edge and the cloud, we choose large thresholds in backup adjustment with $\tau_1 = 100$ and $\tau_2 = 150$.

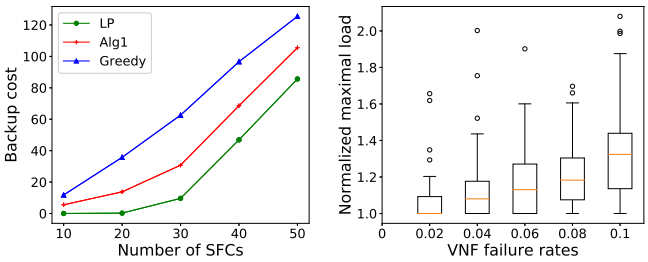
B. Baselines

We compare the performance of our algorithms with that of a greedy baseline algorithm (Greedy) and the solution of a relaxed LP problem (LP) [29]. The cost of LP serves as a lower bound on the cost 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.

Regarding the greedy baseline, as existing algorithms for SFC deployment vary in the objective functions optimized [1]–[4], 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 a SFC arrives, a server with enough available resource 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, the SFC will be placed on the cloud. With the original 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 place it on the server with the lowest load without violating resource or reliability constraints. If any VNF of an SFC cannot be backed up on the edge, the whole SFC is backed up in the cloud.

C. How Much Can Static and Dynamic Backup Placement Algorithms Help?

Fig. 2 (a) shows the average backup cost of different algorithms when the number of SFCs in the network increases. The cost is average over 100 simulations. It is evident that our Algorithm 1 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 a smaller number of SFCs, e.g., 10-30) on the edge, Algorithm 1 reduces cost by 61.1%-50.9%. Even with limited edge resources for backups (e.g., with 50 SFCs and an average of 88.1% load on the edge before deploying backups), Algorithm 1 can still save 15.9%.



(a) (b)

Fig. 2. Effectiveness of the proposed algorithms. (a) illustrates the average backup cost of LP, Algorithm 1 and Greedy with different numbers of SFCs deployed on the edge. (b) shows the maximal load on edge servers from Algorithm 2 normalized by the offline optimum when failure rates increase.

We continue to evaluate the performance of Algorithm 2 with 30-50 SFCs deployed in the network. Each box in Fig. 2 (b) concludes the maximal load on edge servers from Algorithm 2 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. 2 (b) highlights that, in the majority of simulations, Algorithm 2 achieves near optimal performance. Even with a large failure rate (10%), about 75% of simulations are below 1.4 times of the offline optimal solution. This means Algorithm 2 is effective in balancing the

load on edge servers, and thus reducing resource contention in an online manner.

D. Benefits of the SAB Scheme

Now we evaluate the performance of the entire SAB scheme. Again, 30-50 SFCs are deployed in the network. In each time slot, we assume each VNF instance has a failure rate, and the VNF is randomly available or not according to the rate [11]–[15]. By default, we assume a VNF recovers after 10 timeslots (the creation of a VNF takes 1 timeslot) and we vary the recover and create time in Fig. 5 to evaluate their 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 SAB scheme with an h-backup greedy scheme which provides each VNF with h static backups. Its deployment method is similar to that of the greedy algorithm in Section V-B but deploys h static backups instead of one.

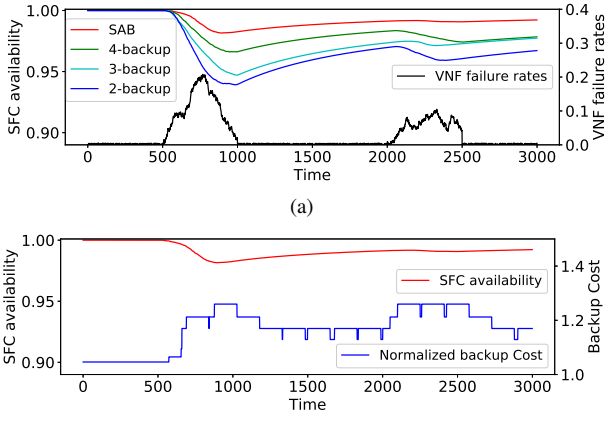


Fig. 3. Self-adapting features of SAB. (a) The availability of SFCs with different backup schemes when failure rate spikes occur. (b) The change of backup cost with SAB when failure rate spikes happen.

We first demonstrate how our backup scheme adaptively balances the trade-off between the backup cost and the availability without knowing failure rates. Fig. 3 (a) shows the availability of SFCs (averaged from the beginning to the current timeslot) with SAB and greedy baselines during 3,000 timeslots. There are two failure rate spikes shown by the black line. Clearly, the availability curve of SAB outperforms the baselines, meaning our scheme is more robust against bursts of VNF failures comparing with baselines using a fixed number of static backups.

Fig. 3 (b) illustrates how SAB reacts to sudden failure rate spikes. In particular, when a spike happens, SAB creates more backups to handle the failures. When the spike ends, SAB 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 SAB 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 dynamic backups. When failure rates fall back, the scheme does the opposite which releases unnecessary dynamic backups and reduces the backup cost.

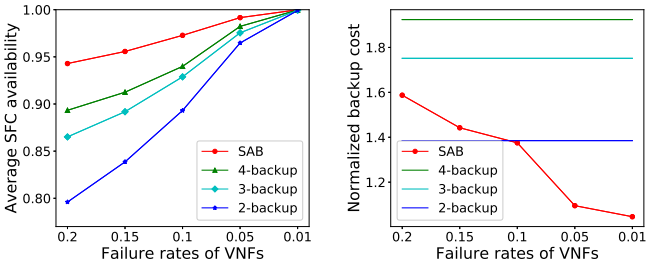


Fig. 4. Performance of different backup schemes in guaranteeing the availability of SFCs and reducing the backup cost. (a) The average availability of SFCs with SAB and the h-backup greedy when failure rates of VNFs change. (b) The average backup cost applying different backup schemes with different failure rates of VNFs. The backup cost is normalized by the static backup cost from Algorithm 1.

We further evaluate the average performance of our scheme with a large number of simulations. Fig. 4 (a) shows the average availability of SFCs applying different backup schemes when the failure rates of VNFs decrease. We find that our scheme always preserves higher SFC availability compared to baseline schemes. We also observe that applying more fixed backups for each VNF can increase the availability. However, this increment is at the cost of higher backup cost.

Fig. 4 (b) shows the average backup cost using different schemes with different failure rates. All results are normalized by the cost of deploying one static backup from Algorithm 1. We observe that the backup cost of the greedy h-backup scheme is fixed with decreasing failure rates and larger h incurs higher backup cost. The backup cost of SAB reduces significantly when failure rates of VNFs decrease. Moreover, the backup cost of our scheme is always lower than the greedy scheme with h larger than 2. Compared with the 2-backup greedy scheme, our scheme has lower cost when failure rates are below 0.1 and preserves much higher SFC availability.

Then, we evaluate how the recover time and the creation time affect the self-adapting backup scheme. Fig. 5 (a) shows the performance of SAB and the greedy baseline when the recover time increases. Failure rates of VNFs are 0.1 in this simulation. Although the availability of SFCs decreases when the recover time increases with both schemes, SAB decreases relatively slower. This indicates that SAB can better tolerant slow recovery of VNFs, thus providing higher availability.

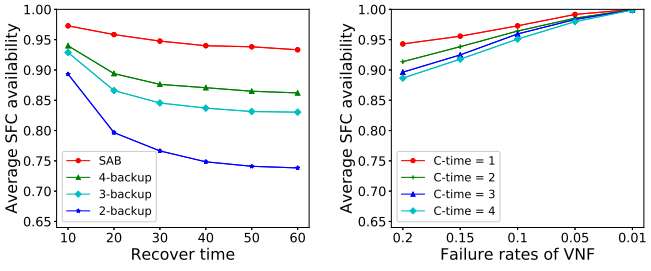


Fig. 5. Impacts of the recover time and the creation time. (a) The average availability of SFCs with SAB and the h-backup greedy scheme when the recover time grows. (b) The average availability of SFCs applying SAB with different creation time.

Since the creation time of an 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. 5 (b) shows the performance of SAB with different creation time. With longer creation time, the average availability of SFCs decreases with SAB. However, compared with Fig. 4 (a), the average availability of our scheme is still better than that of the greedy scheme even with larger creation time. In summary, SAB is applicable and performs much better than the baselines with relatively larger recover and creation time.

E. Experiment

We conduct a small-scale experiment to further evaluate our proposed scheme. We deploy echo servers as VNFs on both the edge and the cloud. For the edge scenario, we deploys the original image, static backup and dynamic backup on three personal PC equipped with Intel i5-4590 within an intranet, respectively. We use a well-known cloud service provider to deploy a static backup in the cloud. At each round, the user sends a packet to the service and we conduct 500 rounds. In the edge scenario, the original VNF fails at round 47 and recovers at 438, while the static backup fails at 182 and recovers at 379. In the cloud scenario, the original VNF fails at round 30 and recovers at round 429. Fig. 6 shows the latency of each packet. It is evident that the edge scenario with SAB provides much lower latency and latency variation compared to the cloud backup scheme.

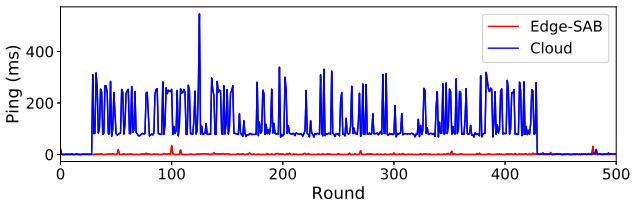


Fig. 6. A small-scale experiment of VNFs backed up on the edge and the cloud.

VI. 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], [19], [20], [31]. Much related work is committed to improving the availability of VNFs in the cloud using redundancy. Fan. et al. [11], [12] propose a scheme to minimize the total number of backups while meeting availability requirements. Zhang et al. [19] 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 SAB scheme does not assume it.

When considering the availability of SFCs chained up by multiple VNFs, Beck et al. [31] propose algorithms to provide backup VNFs and links parallel to the original VNF. Shang et al [13], [15] propose rerouting strategies to guarantee the availability of SFCs in case of node and link failures. 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. [20] 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 self-adapting backup 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. [21] 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. [22] 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 original 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.

Online load balancing problems have been widely studied in the literature and the most related work is probably [32], [33]. Compared to the existing work, our algorithm balances the load with additional reliability constraints of dynamic backups and a shrinking server set with generalized parameters. In this way, our algorithm generalizes existing ones to a wider range of applications including the one studied in this paper, while preserving the theoretical performance guarantee.

VII. Conclusion

In this paper, we propose a self-adapting backup scheme to optimize the trade-off between the SFC availability and backup costs without assuming VNF failure rates. Our scheme first uses a low complexity algorithm with a provable theoretical bound to deploy one static backup for each VNF while minimizing the backup cost. It then adds dynamic backups using an online algorithm with a provable competitive ratio to guarantee the desired availability of SFCs backed up on the edge. Our backup scheme further 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 can significantly reduce the backup cost while guaranteeing the availability without knowing VNF failure rates.

Acknowledgements

This research work was supported in part by the U.S. National Science Foundation under grant numbers CCF-1526162, CCF-1717731, CNS-1617698, CNS-1730128, and CNS-1717588.

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 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 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 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*, 2017.

[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 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 *2016 12th International Conference on Network and Service Management (CNSM)*. 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 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 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 *2018 IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. 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 *2018 IEEE Global Communications Conference (GLOBECOM)*. 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] 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 Conference on Computer Communications*. IEEE, 2019, pp. 1918–1926.

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

[21] H. Zhu and C. Huang, "Edgeplace: Availability-aware placement for chained mobile edge applications," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 11, p. e3504, 2018.

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

[23] X. Shang, Z. Liu, and Y. Yang, "Network congestion-aware online service function chain placement and load balancing," in *Proceedings of the 48th International Conference on Parallel Processing*, 2019, pp. 1–10.

[24] Y. Zeng, Y. Huang, Z. Liu, and Y. Yang, "Joint online edge caching and load balancing for mobile data offloading in 5g networks," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 923–933.

[25] J. Comden, S. Yao, N. Chen, H. Xing, and Z. Liu, "Online optimization in cloud resource provisioning: Predictions, regrets, and algorithms," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 1, p. 16, 2019.

[26] J. Maghakian, J. Comden, and Z. Liu, "Online optimization in the non-stationary cloud: Change point detection for resource provisioning," in *2019 53rd Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2019, pp. 1–6.

[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. Geng *et al.*, "The complexity of the 0/1 multi-knapsack problem," *Journal of Computer Science and Technology*, vol. 1, no. 1, pp. 46–50, 1986.

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

[30] 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 *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 153–167.

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

[32] 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.

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