Availability-aware Mapping of Service Function Chains

Jingyuan Fan, Chaowen Guan, Yangming Zhao, and Chunming Qiao
Department of Computer Science and Engineering
University at Buffalo, Buffalo, NY 14260 USA

***Abstract*—Network Function Virtualization (NFV) is a promising technique to greatly improve the effectiveness and flexibility of network services through a process named Service Function Chain (SFC) mapping, with which different network services are deployed over virtualized and shared platforms in data centers. However, such an evolution towards software-defined network functions introduces new challenges to network services which require high availability. One effective way of protecting the network services is to use sufficient redundancy.**

By doing so, however, the efficiency of physical resources may be greatly decreased. To address such an issue, this paper defines an optimal availability-aware SFC mapping problem and presents a novel online algorithm that can minimize the physical resources consumption while guaranteeing the required high availability within a polynomial time. Simulation results show that our proposed algorithm can significantly improve SFC mapping request acceptance ratio and reduce resource consumption.

I. Introduction

Network Function Virtualization (NFV) is a driving force behind implementing network functions on a virtualized and shared platform, and it can significantly reduce the hardware cost and investment, as well as greatly improve the efficiency and flexibility of utilizing hardware resources. In NFV, network functions are deployed through a process called *Service Function Chain* (SFC) mapping. An SFC consists of a set of *Virtual Network Functions* (VNFs) interconnected by logical links. Multiple SFCs from distinct clients may share the computing and networking resources in order to improve the resource utilization.

However, service chaining aggravates the availability problem faced by the cloud industry. Even if the availability of each VNF is high, the availability of a service chain may be unacceptable. For example, assume we have a linear chain which consists of 6 VNFs, and the availability of each VNF is 0.95, therefore the availability of this chain is 0.95^6, that is about 0.74, which cannot meet most applications' requirements. To mask failures, redundancy is a *de-facto* technique [14], [19], [7]. However, when deploying redundancy, careful planning is necessary to avoid waste of resources for service chains. As shown in Fig. 1, we have a service chain with 4 primary VNFs and the availability requirement is 0.82. The number near each VNF is its availability. Here we use the traditional active/standby (i.e., 1+1) redundancy model such that the primary VNF can be failed over to the standby entity in case it fails. The solid line and the dashed line represent one redundancy deployment strategy respectively.

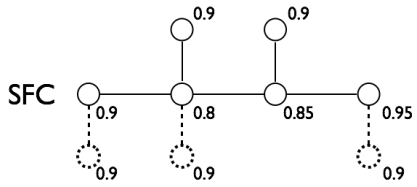


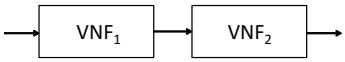
Fig. 1: Two redundancy deployment strategies

While both strategies can achieve the availability requirement (their availabilities are 0.825 and 0.8205, respectively), it is clear that the solid one uses less backup VNFs, and may save resources for other chain requests.

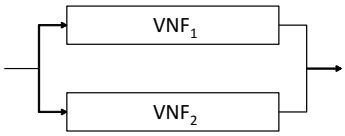
In this paper, we take the first step by addressing the following problem of availability-aware SFC mapping with off-site redundancy: *what is the minimum number of off-site backup VNFs service provider needs to provision to guarantee a certain degree of availability of a service chain?* In particular, we are interested in providing off-site redundancy. It is recommended [4] that such off-site resources should be available. The objective of our work is to meet each request's heterogeneous availability requirement such that a higher SFC request acceptance ratio can be achieved, while reducing resource consumption for service providers. To the best of our knowledge, none of the existing works has considered similar problems.

In order to solve the problem, we develop a novel algorithm that improves a service chain's availability in an iterative way, and in each iteration, we try to solve the following two sub-problems: how to efficiently and accurately evaluate availability of a service chain with off-site redundancy? What is the optimal strategy to deploy off-site backup VNFs? To answer the former one, we propose a novel method which can evaluate a service chain's availability incrementally in a polynomial time with negligible error. For the latter one, we propose a greedy algorithm with a theoretical lower bound and also show that it is optimal under some circumstances. By simulation, we show that our proposed algorithm can significantly improve SFC mapping request acceptance ratio and reduce resource consumption.

The rest of the paper is organized as follows. Section II describes our availability and redundancy models. Section III defines the problem of availability-aware SFC mapping and shows its complexity. Section IV introduces a polynomial



((a)) Linearly Connected Two VNFs



((b)) Parallelly Connected Two VNFs

Fig. 2: Two ways of combining VNFs

running time algorithm for availability evaluation and an approximation algorithm with a theoretical lower bound for backup VNF selection. We evaluate the performance of the proposed algorithms in Section V, followed by a conclusion in Section VI.

II. Availability and Redundancy Models

In this section, we briefly discuss the end-to-end availability model and various redundancy models considered in our work.

A. Availability Model

To estimate the availability for a given service chain deployment, we first need to model the logical structure. In our work, we assume that the failure of each VNF is independent.

1) Availability of one single component: The availability of a complex system such as a service chain deployment can be modelled by decomposing it into constituent components [4], of which the availability are known. The availability of a component is the relative share of time the component is functioning, and thus the probability to find the component working if checking it at a random point in time. Therefore, the availability of a VNF can be expressed using uptime followed by downtime, which can be characterized in terms of *Mean Time Between Failures* (MTBF) and *Mean Time To Repair* (MTTR), respectively. In general, the availability of a VNF can be characterized as

$$A = \frac{Uptime}{Uptime + Downtime} = \frac{MTBF}{MTBF + MTTR} \quad (1)$$

2) Availability of composed system: An SFC is generally composed of a number of VNFs. In order to estimate the availability of such composite system, which is derived from the individual components it consists of, two basic ways of combining components, serial and parallel, need to be understood. If individual components are connected in a serial manner, all components that the SFC comprises need to function at the same time. For example, as shown in Fig. 2 (a), in order to have packets processed by all the functions provided by this SFC, both VNF_1 and VNF_2 need to be available at a given time. Therefore, the availability of this SFC request is:

$$A_{SFC} = A_{VNF_1} \times A_{VNF_2} \quad (2)$$

where A_{VNF_1} and A_{VNF_2} are the availabilities of VNF_1 and VNF_2, respectively.

If two individual components are connected in a parallel way, as shown in Fig. 2 (b), and if both VNF_1 and VNF_2 provide the same function, the requested service is available when at least one of these two independent components can function, assuming there is no service disruption due to fail over. Thus, the availability of this SFC request can be described as:

$$A_{SFC} = 1 - ((1 - A_{VNF_1}) \times (1 - A_{VNF_2})) \quad (3)$$

3) Computing end-to-end availability: In this paper, we only consider VNF failures for simplicity. Therefore we define *end-to-end availability of a service chain as the probability to find all functions provided by this chain are available at a given time*. Using these two basic models mentioned above, we can model and evaluate the end-to-end availability of a more complicated SFC request with different redundancy models.

B. Redundancy Models

In this paper, we mainly consider three off-site redundancy models mentioned in [12]. For dedicated protection (DP), the backup VNF carries no traffic, and assumes the identity of the primary VNF only in case of a failure. For shared protection (SP), one backup VNF can take over the traffic when any one of the primary VNFs it protects fails by allocating the maximum amount of resources required among all primary VNFs. The third one is joint protection (JP), which is a variation of SP and its effectiveness has been shown in [12]. In general, JP requires a backup VNF to reserve resources that are sufficient for all primary VNFs it protects. In a wide area network, when the VNFs of a service chain are mapped to multiple data center sites connected by fiber links, *Optical Orthogonal Frequency Division Multiplexing* can be used to carry the huge amount of traffic flow between these sites, and JP can effectively save link resources compared to the other two redundancy modes. For simplicity, for both SP and JP, we assume one VNF can backup at most two primaries, and leave the generalization to more primary VNFs to the future work. However, for such a simple case, we have the following theorem:

Theorem 1. *Verifying if the availability of a given deployed service chain with backups is above a given threshold is PP-complete.*

Detailed proof can be found in the **appendix**.

III. Problem formulation & Complexity

In this section, we formally describe the availability-aware SFC mapping problem and show the hardness of the problem.

A. Availability-aware SFC Mapping Problem

Given a Physical Network $P = (\mathbb{N}, \mathbb{E})$, where $\mathbb{N}$ donates a set of nodes, including data center sites $\mathbb{N}_D$ where VNFs can be deployed[1] and flow access/exit points $\mathbb{N}_F$, and $\mathbb{E}$ is the

[1]For the rest of the paper, terms "site" and "data center" are used interchangeably.

set of physical links (optical fibers) connecting $\mathbb{N}$. Each site $n \in \mathbb{N}_D$ is associated with a set of k types of resources $S_n^k = \{s_n^i | i \in [1,k]\}$, where s_n^i denotes the capacity of resource of type i. Each physical link $e \in \mathbb{E}$ has different amount of bandwidth b_e, and the communication delay of each link e is d_e. Given the set of resources available at a site n, it can provide a set of VNFs denoted by F_n (i.e., function constraint). $F = \bigcup_{i=1}^{\mathbb{N}_D} F_i$ is the set of all VNFs.

Assume that there is a set of m service chain requests in the network, denoted by $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_m\}$. Each request can be described by $\gamma_i = (s_{\gamma_i}, d_{\gamma_i}, f(\gamma_i), F_{\gamma_i}, l_{\gamma_i}, \alpha_{\gamma_i})$. s_{γ_i} and d_{γ_i} represent the ingress and egress of the chain, respectively, and they are fixed in the network. F_{γ_i} is the set of primary VNFs (corresponding to backup VNFs) of chain γ_i and the z-th VNF ($1 \leq z \leq |F_{\gamma_i}|$) in chain is denoted by $f_{\gamma_i}^z$. Each VNF $f_{\gamma_i}^z$ incurs a processing delay, denoted as $d_{\gamma_i}^z$. Thus, a chain is logically represented as $(|F_{\gamma_i}|+2)$ nodes, including ingress and egress, and there are $(|F_{\gamma_i}|+1)$ logical links between nodes. $f(\gamma_i)$ is the bandwidth required for the request, and $n_{\gamma_i}^{zj}$ is the amount of resource of type j that VNF $f_{\gamma_i}^z$ requires where $j \in [1,k]$. Each request requires that the end-to-end delay from ingress to egress is within l_{γ_i} and the availability is above α_{γ_i}. To map a chain request onto the physical network, we not only need to map all VNFs onto $\mathbb{N}_D$ but logical links onto $\mathbb{E}$. To map a VNF to a data center site, we need to reserve an appropriate amount of resources at that site. In a wide area network, primary VNFs of a service chain may be implemented in one single data center for low latency [20], [18] or distributed at geographically different locations [25] for reasons [2], [1], [21], [8], such as 1) some data centers may only implement limited types of network functions to reduce OPEX, and 2) 3rd party VNFs can be hosted in public cloud, places like Amazon AWS rather than service providers' infrastructure. However, each VNF can only be mapped to one single site. To map logical link between data centers, we need to allocate an appropriate amount of bandwidth along each and every physical link along the chosen path to carry the traffic flow from one VNF to another VNF, if these two VNFs are mapped to different data center sites. Since we assume the data center sites are connected with fibers, two constraints needs to be considered [22]: 1) the wavelength/spectrum continuity, and 2) the transmission reach of the light-path of a logical link with a specific modulation format. Link mapping/optimization inside a data center is beyond the scope of this work, and plenty of research have been conducted in that area [17]. When mapping VNFs, their ordering should also be considered.

When no redundant VNFs are provisioned, the availability of a service chain request γ_i can be obtained as $A_{\gamma_i} = \prod_{f \in F_{\gamma_i}} A_f$, where A_f is the availability of the VNF $f \in F_{\gamma_i}$. Here we assume that the ingress and egress nodes and all the data center sites are always available (i.e., availabilities are 1), and the service provider can know the availability of a VNF after it is deployed. Note that all VNFs have heterogeneous availabilities. However, when there are redundancy, evaluating availability becomes a hard problem (discussed in Section II-B). Upon mapping, we also need to consider four key constraints:

1) Site capacity: the total load of resource type i across all chain requests and all VNFs, including primary and backup, at each data center site $n \in \mathbb{N}_D$ should be less than or equal to its capacity s_n^i.
2) Link capacity: the total load across all chain request and all logical links at each physical link $e \in \mathbb{E}$ should be less than or equal to its bandwidth capacity b_e.
3) Delay: for a wide area service chain, the end to end delay of each request γ_i should be less than or equal to l_{γ_i}. The delay includes both VNF processing delay at data center sites and communication delay along the links.
4) Availability: The availability of one chain request should meet the client's requirement with redundant VNFs if necessary.

If any of the above requirements cannot be met, we consider this chain request as being blocked. Note that we only consider the delay constraint when mapping primary VNFs, and will extend it to both primary and backup VNF mapping in our future work. Therefore, we can define the SFC availability-aware mapping problem as follows. Given a set of SFC requests, each with a specific availability requirement, we need to find out the minimum number of backup VNFs needed, efficiently place primary and backup VNFs to the data center sites and map logical links to physical links that can satisfy all the constraints mentioned above. A decision algorithm can be embedded in a centralized system, such as NFV Management & Orchestration (MANO), that manages all the incoming requests.

B. Problem Complexity

In this section, we will briefly discuss the complexity of our problem. Due to the limit of pages, we defer more details of the complexity analysis to the **appendix**. Concretely, below we list the conclusions we draw from the analysis.

Even with an oracle to compute availability given a mapped chain request with backups (**Theorem 1**), we still cannot optimally decide if there exists a solution for this service chain request, and finding a local optimal is difficult.

Theorem 2. *Determining if there exists a solution for a service chain request is NP^{PP}-complete.*

Theorem 3. *Finding a local optimal solution for a chain request is co-NP^{PP}-complete.*

The objective of globally minimizing the number of backup VNFs further elevates the complexity.

Theorem 4. *Finding the optimal solution for one service chain request belongs to $NP^{NP^{PP}}$.*

The complexity classes we mention satisfy these containment properties and relations to other classes [16]:

$$\text{P} \subseteq \begin{matrix}\text{NP}\\ \text{co-NP}\end{matrix} \subseteq \text{PP} \subseteq \begin{matrix}\text{NP}^{\text{PP}}\\ \text{co-NP}^{\text{PP}}\end{matrix} \subseteq \text{NP}^{\text{NP}^{\text{PP}}} \subseteq \text{PSPACE}$$

Hence the availability-aware SFC mapping problem is believed to be intractable.

IV. Algorithm Design

In this section, we propose an online algorithm to provide off-site redundancy for availability-aware wide area service chaining.

Our design is independent of the specific topology used by the physical network or service. Our goal is to find the most resource-efficient mapping for each SFC request while meeting all four constraints mentioned in III-A. The metric of interest is the SFC acceptance ratio, defined by the number of accepted SFC request by the decision algorithm over the total number of SFC request. The redundancy model used is JP mentioned in Section II-B.

A. Overview

Based on our complexity analysis in Section III-B, we know finding the optimal solution is challenging. Hence, to address the challenges, we can decompose the mapping problem into two phases: primary mapping and backup mapping. In primary mapping, we need to map all primary VNFs and the associated logical links to the physical network. In backup mapping, we only consider the availability constraint in order to select backup VNFs. As we decompose our solution in this way, one can always add more constraints to the problem, which will only change how primary mapping is done, while our backup mapping solution can still work; furthermore, it makes the backup mapping process a "patch" to improve the availability in other SFC mapping works. The algorithm is summarized as the follows.

Algorithm 1 Availability-aware SFC mapping

for each pair of ingress $n_{in} \in \mathbb{N}_F$, egress $n_e \in \mathbb{N}_F$ **do**
  compute K-shortest path offline, denoted as $K_{n_{in}n_e}$
  sort $K_{n_{in}n_e}$ in descending order based on the delay
end for
for each service chain request $\gamma_i \in \Gamma$ **do**
  for each path $e \in K_{s_{\gamma_i}d_{\gamma_i}}$ **do**
    map all VNFs F_{γ_i} to the data centers along the path e while balancing load across all the data centers subject to the function constraint (similar to copying books problem)
    `delay` = compute the end-to-end delay
    if `delay` $< l_{\gamma_i}$ & primary mapping succeeds **then**
      `backup` = Ø
      while EVAL($F_{\gamma_i} \bigcup$ `backup`) $< \alpha_{\gamma_i}$ **do**
        `backup` = SELECT($F_{\gamma_i} \bigcup$ `backup`)
      end while
      if a valid backup plan is found **then**
        map the backup VNFs and links
      end if
    end if
  end for
end for

For primary mapping, we first offline compute K-shortest path [23] between each pair of ingress and egress $n_{in}, n_e \in \mathbb{N}_F$ and sort K paths for each set of paths $K_{n_{in}n_e}$ in descending order based on the end-to-end communication delay. Obviously, there are data center sites along each path. Given a request with fixed ingress s_{γ_i} and egress d_{γ_i}, we map it to the physical network by iterating over all K-shortest path $K_{s_{\gamma_i}d_{\gamma_i}}$. The primary VNFs of the request are mapped to the sites along the selected path while balancing load across all these sites, subject to the delay, site capacity, link capacity and function constraints. Note that without the function constraint, our algorithm can still work.

Backup mapping consists of two components: a 'backup picker' (**Theorem 6** in Section IV-B) that proposes backup VNFs selection which will maximize the availability, subject to the resource availability for each site and physical link (SELECT in **Algorithm 1**), and a 'backup validator' (Section IV-C) that confirms or rejects the proposal by evaluating the availability and comparing the evaluation with the SFC requirement (EVAL in **Algorithm 1**). The process repeats until either the availability requirement is met, confirmed by the validator, or no more site and/or link resource is available to improve the availability of the request, output by the picker. In each iteration, the picker chooses one backup VNF for two primary VNFs. If a request is rejected in any one of the above steps, we redo primary mapping until all K paths are checked. Note that our backup mapping algorithm can work, regardless of whether primary VNFs are mapped to one or multiple data center sites since our algorithm works with the granularity of VNF.

Next, we in detail discuss the backup mapping procedure. Since how we choose backup in each iteration depends on how we evaluate the availability, we first describe the validator.

B. Validating Backup Choice

The validator determines whether the backups proposed by the picker are enough to meet the availability requirement. As seen from **Theorem 1**, the problem is PP-complete, so there's no polynomial time solution to solve the problem. Computing the exact availability requires one to go over exponential possible states [10]. In order to address this problem, previous research [15], [24] proposed solutions using *Monte Carlo* related methods, but it's hard to determine how many steps are needed to converge to the stationary distribution within an acceptable error, and the procedure is time consuming. In this subsection, we will show how to estimate availability in an easier way. In a nutshell, since we choose backups in an iterative way, the availability of a service chain can be built incrementally. Compared to other approximation approaches used in [10], our method does not require one to re-evaluate the availability of the whole service chain every time the picker selects.

Let us explain the rationale first. We consider a whole chain as a composition of several independent sub-chains, and thus, the availability of the request is the multiplication of the availability of each independent sub-chains. Here we say a

sub-chain is independent if any VNFs in this sub-chain does not share backups with VNFs in other sub-chains. Therefore, before any backup is provisioned, all VNFs are considered as independent sub-chains, so the availability of a service chain is the multiplication of the availability of all primary VNFs. Based on our algorithm, until the availability requirement is met, two primary VNFs with a backup VNF are selected in each iteration by the picker (explained in Section IV-C). Note that these two primary VNFs may or may not belong to the same sub-chain (i.e., the sub-chain(s) they belong to may or may not be independent). Then, we will have a new sub-chain which includes the backup VNF and two sub-chains, each of which contains one of the two selected primary VNFs. For example, assume we have a service chain with three sub-chains N_1, N_2 and N_3. If the picker selects one VNF from N_1 and another one from N_2, and provide them with a backup, the availability of the whole chain would be $A_{12} \times A_3$, where A_{12} is the availability of the new sub-chain including sub-chains N_1, N_2 and the backup and A_3 is the availability of N_3 if N_3 is an independent sub-chain. Therefore, how to efficiently compute A_{12} is important. Next, we discuss it with four mutually excluded cases.

Assume the availabilities of the two selected VNFs n_i, n_j are A_i, A_j respectively, and the availability of the backup b is A_b. n_i and n_j provide functions f_{n_i}, f_{n_j} respectively. Upon provisioning b as backup there are totally four possible cases:

1) Neither n_i nor n_j has backups: If neither n_i nor n_j has backups, then both n_i and n_j are considered as independent sub-chains. Therefore if a backup is provisioned to them, the availability A_{ij} of the new sub-chain, including n_i, n_j and b, equals to the probability that the backup is available plus the probability that both n_i and n_j are available while the backup b is not. Therefore

$$A_{ij} = 1 - (1 - A_b) \times (1 - A_i A_j) \tag{4}$$

2) Only one of n_i or n_j has backups: Without loss of generality, we assume n_j is the VNF with backups, and denote the sub-chain containing n_j as N. With a backup b using JP, we can see that the function required by n_j can be provisioned either by the sub-chain N or the new backup b, and these two cases are mutually excluded. Hence the new sub-chain is considered available if the sub-chain N functions properly and at least one of the backup b and n_i works properly (to provide function f_{n_i}), or backup b is available and all the VNFs in sub-chain N except for the ones providing f_{n_j} are available. Therefore, the availability A_{ij} of the sub-chain, including n_i, N and b, is

$$A_{ij} = A_N \times (1 - (1 - A_b)(1 - A_i)) + A_b \times A_{N\backslash j} \tag{5}$$

where $A_{N\backslash j}$ is the probability that sub-chain N can provide all the functions except f_{n_j}. To compute $A_{N\backslash j}$, we decompose $A_N = A'_N - A_{N\backslash j} = A'_N - (1 - A_j)\prod_{k=1}^{M}(1 - A_{j_k})A''_N$, where A'_N is the probability that the sub-chain N may or may not be able provide f_{n_j}, while all the other VNFs are available, A''_N is the probability that all VNFs except the one providing f_{n_j} in sub-chain N are available, A_{j_k} is the availability of the k-th backup for n_j excluding backup b, and M is the total number of backups that n_j has. We then define $\tau = \frac{A'_N}{A''_N} \approx 1 + \epsilon$, where ϵ is a small constant. Note that as the sub-chain contains more VNFs, τ gets closer to 1. So

$$A_{N\backslash j} = \frac{(1 - A_j)\prod_{k=1}^{M}(1 - A_{j_k})A_N}{\tau - (1 - A_j)\prod_{k=1}^{M}(1 - A_{j_k})} \tag{6}$$

3) Both n_i and n_j have backups and they belong to different sub-chains: We denote the sub-chains containing n_i and n_j using W and N respectively. Applying similar strategy described in the previous case, the probability that all the VNFs in the new sub-chain are available can be decomposed into two cases: 1) both sub-chains W and N work properly, and 2) backup b is available and at least one of the sub-chain W and N fails to provide f_{n_i} and/or f_{n_j} while other VNFs are available. Hence, the availability A_{ij} of the sub-chain, including W, N and b, is

$$A_{ij} = A_N \times A_W + A_b(A_{N\backslash j}A_{W\backslash i} + A_{N\backslash j}A_W + A_{W\backslash i}A_N) \tag{7}$$

4) Both n_i and n_j have backups and they belong to the same sub-chain: Similarly, when all the VNFs in the sub-chain N containing both n_i and n_j are available, whether backup b is available has no influence on the availability of N. On the other hand, when backup b is available, at most one of the VNFs providing function f_{n_i} and f_{n_j} should be available. Thus, the availability A_{ij} of the sub-chain, including N and b, is

$$A_{ij} = A_N + A_b \times A_{N\backslash ij} \tag{8}$$

where $A_{N\backslash ij}$ denotes the probability that sub-chain N can provide all functions except for f_{n_i} and f_{n_j}. Here, verifying if n_i and n_j have common backup VNFs is necessary to avoid double calculating.

Accordingly, the availability of the VNFs selected by the picker needs to be updated as well. As seen from the methods described in this subsection, for each iteration the computation complexity of computing availability is polynomial time with respect to the number of backup that the selected VNF has. We will later show in the simulation that the estimation error is small enough to be neglected.

C. Choosing Backup

The primary job of the picker is to select minimum number of backups that a service chain requires to meet the availability requirement. Hence, the *Proposition* immediately follows [16].

Proposition 1. *Selecting minimum number of backups cannot be approximated within any fixed factor in polynomial time unless P=NP.*

The intractability result holds in general, i.e., when no further constraints are put on the problem instances. Therefore, solving this problem is also hard. In this subsection, we propose a greedy heuristic to select backup VNFs (SELECT in **Algorithm 1**) to maximize the availability that a service chain

can achieve in each iteration assuming all backups have the same availabilities and we can compute the exact availability of a chain in polynomial time. Before presenting the detailed algorithm we first define *improvement ratio* of availability.

Definition 1. *Define an* **improvement ratio** *as the ratio of the improvement of the end-to-end availability to the end-to-end availability before provisioning a backup.*

Then we can have the following *Theorem*.

Theorem 5. *Provisioning a backup VNF to two primary VNFs whose availabilities are among the lowest maximizes the improvement ratio for each case described in Section IV-B.*

Here we only prove **Theorem 5** for the second case in Section IV-B. Note that we can similarly prove **Theorem 5** for other cases as well (in fact, case 1 is simpler, and cases 3 and 4 are based on case 2).

Proof. Given two VNFs n_i and n_j selected, b as a backup VNF, and assume n_j already has backups. As n_j already has backups, we must have computed the availability of the sub-chain N which contains n_j according to Section IV-B. Then the end-to-end availability before provisioning backup b is $A_{before} = A_1A_2 \dots A_kA_iA_NA_p \dots A_q$, and the availability after adding the backup is $A_{after} = A_1A_2 \dots A_k(A_N(1 - (1 - A_b)(1 - A_i)) + A_bA_{N\setminus j})A_p \dots A_q$ where $A_1A_2 \dots A_k$ and $A_p \dots A_q$ are the availabilities of the rest independent sub-chains, and A_N is dependent on A_j. Therefore u is,

$$u = \frac{A_{after} - A_{before}}{A_{before}} = -A_b + \frac{A_b}{A_iA_N}(A_N + A_{N\setminus j}) \quad (9)$$

Let's substitute Eq. (6) for $A_{N\setminus j}$, and let $S = (1 - A_j)\prod_{k=1}^{M}(1 - A_{j_k})$ then calculate the partial derivatives with respect to A_i and A_j respectively,

$$u_{A_i} = \frac{\partial u}{\partial A_i} = -\frac{A_b}{A_i^2A_N}(A_N + A_{N\setminus j}) \quad (10)$$

$$u_{A_j} = \frac{A_b}{A_i}\frac{(S'A_N^2 + 2\frac{\partial A_N}{\partial A_j}S \times A_N)(\tau - S) - (\tau - S)'S \times A_N^2}{(\tau - S)^2} \quad (11)$$

As the availability is always greater than or equal to 0, and $A_i \in [0, 1]$, from Eq. (10) we can easily tell that u decreases monotonically as A_i increases. While the monotonic property of Eq. (11) is not as obvious as Eq. (10). To see that, we let $u_{A_j} = 0$ to compute the critical point, and get

$$2\frac{\partial A_N}{\partial A_j} = (\frac{S}{\tau - S} + 1)\frac{A_N}{1 - A_j} \quad (12)$$

Solve this partial differential equation, and get

$$A_N = \sqrt{\frac{\frac{\tau}{\prod_{k=1}^{M}(1-A_{j_k})} - (1 - A_j)}{1 - A_j}} \quad (13)$$

which means that when the equation holds true, we get the critical point. However, $\frac{\tau}{\prod_{k=1}^{M}(1-A_{j_k})} \gg 1$ since $\tau > 1$, $M \geq 1$ and both $A_N \in [0, 1]$ and $A_j \in [0, 1]$. Therefore this equation can never hold, which means u is a monotone function respect to A_j in its domain. Also we can easily check $u_{A_j=1} < u_{A_j=0}$, so we know that u decreases monotonically as A_j increases. Therefore, selecting two VNFs with lowest availabilities leads to the largest improvement ratio. □

Define $AE(B)$ as the function to accurately calculate the availability of a service chain with a set of backup VNF B, and $\rho_b(B) = AE(B \cup \{b\}) - AE(B)$ as the availability improvement when adding a backup VNF b. So $AE(\emptyset)$ is the availability of the service chain without any backups. Define ρ_b^i as the availability improvement when a backup b is provisioned and the relationship of the two VNFs that the picker selects belongs to case i. i is the case number as defined in Section IV-B. Then we have the following *Lemma*.

Lemma 1. $\rho_b^1 > \rho_b^2 > \rho_b^3 > \rho_b^4$

Proof. Here we only prove the first inequality. One can prove the rest using the same method. Given a service chain which consists of three independent sub-chains, N_1, N_2 and N_3. In N_1 and N_2 all VNFs are primary while N_3 is composed of primary and backup VNFs. Either we can provide a backup b for VNF i and j or VNF i and p, where VNF i, j and p are primary VNFs in sub-chain N_1, N_2 and N_3, respectively. If VNF i and j are selected, $\rho_b^1 = (A_b - A_bA_{N_1}A_{N_2}) \times A_{N_3}$; if VNF i and p are selected, $\rho_b^2 = (A_{N_3} - A_{N_3}A_{N_2} + A_{N_3\setminus p}) \times A_bA_{N_1}$. For $\rho_b^1 > \rho_b^2$, the following inequality must hold

$$\frac{1 - A_{N_1}}{A_{N_1}} > \frac{A_{N_3\setminus p}}{A_{N_3}} \quad (14)$$

We argue that this inequality should hold in practice where failure of a VNF happens relatively rarely, which makes $1 - A_{N_1}$ is at least one order of magnitude larger than $A_{N_3\setminus p}$, while A_{N_1} and A_{N_3} are about the same order. □

Together with **Theorem 5**, we outline how picker selects backup VNFs with the following *Theorem*.

Theorem 6. *Selecting two VNFs whose relationship belongs to the category in Section IV-B with the smallest case number, and settling ties by choosing the VNFs whose availabilities are among the lowest would maximize the availability improvement for each iteration (Line 11-13 in Algorithm 1).*

Proof. Based on **Theorem 5** and **Lemma 1**, the theorem follows immediately. □

Next, we analyze how close the availability derived from the backup plan selected by the picker is to the one achieved by the optimal backup solution. Assume the request has n primary VNFs, and K_b is the number of backup VNFs that can be provisioned.

Theorem 7. *When $K_b \leq \lceil \frac{n}{2} \rceil$, our greedy backup selection method can achieve the optimal solution.*

Proof. When $K_b < \lceil \frac{n}{2} \rceil$, or $K_b = \lceil \frac{n}{2} \rceil$ and n is even, according to **Lemma 1**, the VNFs that each one of K_b backup protects should not overlap. To prove the optimality of this

greedy algorithm, we need to prove the greedy choice property and the optimal structure property [11].

Greedy Choice Property: The greedy algorithm selects two VNFs with the lowest availabilities among all primary VNFs as the first pair of VNFs to be provisioned with a backup. Say these two VNFs are i and j, and the backup VNF is b. We have to show that there exists an optimal backup strategy that also contains a backup to protect i and j. There are four possible cases:

1) The optimal backup strategy contains a backup to protect VNFs i and j, then we are done.
2) The optimal backup strategy doesn't contain any backup to protect either i or j. Then we can remove any one backup from the optimal strategy and add a backup to protect i and j. By doing so, we get a higher availability. Contradiction.
3) The optimal backup strategy contains a backup to protect one of i and j. Similarly, we can remove this backup from the optimal strategy and add a backup to protect i and j so that we can get a higher availability. Contradiction.
4) The optimal backup strategy contains two backups which protect i and j respectively. Assume VNFs i and p is one pair and VNFs j and q is the other pair. Without loss of generality, we assume $A_i < A_j < A_p \leq A_q$, then we need to show when i and j form a pair and p and q form another pair, we have a higher availability. Based on case 1 in Section IV-B, if we can get a higher availability, then

$$\begin{aligned}(A_b + (1-A_b)A_iA_j)(A_b + (1-A_b)A_pA_q) > \\ (A_b + (1-A_b)A_iA_p)(A_b + (1-A_b)A_jA_q) \\ \Leftrightarrow A_iA_j + A_pA_q > A_iA_p + A_jA_q \\ \Leftrightarrow A_p(A_q - A_i) > A_j(A_q - A_i) \quad (15)\end{aligned}$$

Since $A_q > A_i$ and $A_p > A_j$ by our assumption, the inequality holds. Contradiction.

Optimal Structure property: Let P_1 be the subproblem obtained from the original problem P by removing VNFs i and j. Let S be an optimal backup strategy for the original problem P, in which VNF i and j are the first pair of VNFs that is selected by the picker. Let S_1 be obtained from S by deleting b. Then S_1 is a backup strategy for the subproblem P_1. We need to show that S_1 is an optimal solution for P_1. Towards a contradiction, suppose this is not the case and we have S_1' as the optimal solution for P_1. Then we can build a backup strategy $S' = S_1' \bigcup b$ for P with a higher availability. This contradicts the fact that S is an optimal strategy of P.

Since both properties hold, the greedy algorithm is correct.

When $K_b = \lceil \frac{n}{2} \rceil$ and n is odd, after we choose $\lfloor \frac{n}{2} \rfloor$ pairs of VNFs to provide with a backup each, we achieve the maximum availability possible. Then we greedily find the VNF with a backup protected and the lowest availability and pair it with the only left primary VNF which doesn't have a backup to provide them a with backup, which can maximize the availability based on **Theorem 5**. This concludes the proof. □

With $K_b > \lceil \frac{n}{2} \rceil$, we can prove our algorithm is near-optimal.

Theorem 8. *when $K_b > \lceil \frac{n}{2} \rceil$, the algorithm computes a backup scheme which maximizes the availability with a normalized relative error of $\frac{e-1}{e}AE(OPT) + \frac{1}{e}AE(\emptyset)$, where $AE(OPT)$ and $AE(\emptyset)$ are the availabilities with an optimal backup solution and without backups, respectively.*

Proof. For arbitrary backup set S and T with $T - S = \{j_1, j_2, \ldots, j_\tau\}$ and $S - T = \{k_1, k_2, \ldots, k_\nu\}$, we have

$$\begin{aligned}&AE(S \cup T) - AE(S) = \\ &\sum_{t=1}^{\tau}[AE(S \cup \{j_1, j_2, \ldots, j_t\}) - AE(S \cup \{j_1, j_2, \ldots, j_{t-1}\})] = \\ &\sum_{t=1}^{\tau} \rho_{j_t}(S \cup \{j_1, j_2, \ldots, j_{t-1}\}) \leq \sum_{v \in T-S} \rho_v^{\varphi(S')}(S') \quad (16)\end{aligned}$$

where $\varphi(S')$ is the smallest case number of calculating ρ_{j_1}, and this can be realized by dynamically changing S'. When adding an element, say j_p, into the backup set, if the way of calculating the availability is the same as the way of calculating availability when adding the 1^{st} element, $S' = S \cup \{j_1, j_2, \ldots, j_{\nu-1}\}$; otherwise, $S' = S \cup \{j_1, j_2, \ldots, j_{\nu-1}\} - \{Q\}$, where $Q \subseteq S \cup \{j_1, j_2, \ldots, j_{\nu-1}\}$ so as to ensure the availability is calculated using the same case number when adding the 1^{st} backup VNF. Based on **Lemma 1**, this inequality holds. Similarly,

$$\begin{aligned}&AE(S \cup T) - AE(T) = \\ &\sum_{t=1}^{\nu}[AE(T \cup \{k_1, k_2, \ldots, j_t\}) - AE(T \cup \{k_1, k_2, \ldots, k_{t-1}\})] = \\ &\sum_{t=1}^{\nu} \rho_{k_t}(T \cup \{k_1, k_2, \ldots, k_{t-1}\} - k_t) \geq \sum_{v \in S-T} \rho_v^{\varphi(\chi)}(\chi) \\ &\quad (17)\end{aligned}$$

where $\chi = T \cup S - \{v\}$ and $v \in S - T$. Subtracting Eq. (17) from Eq. (16), we get

$$\begin{aligned}&AE(T) - AE(S) \leq \\ &\sum_{v \in T-S} \rho_j^{\varphi(S')}(S') - \sum_{v \in S-T} \rho_j^{\varphi(\chi)}(\chi) \leq \sum_{v \in T-S} \rho_j^{\varphi(S')}(S') \\ &\quad (18)\end{aligned}$$

since the improvement of availability is always greater than or equal to 0. Taking T as the optimal solution, S to be the set S^t generated after t iterations of the greedy algorithm, and using

$$AE(S^t) = AE(\emptyset) + \sum_{i=0}^{t-1} \rho_i \quad (19)$$

and $AE(OPT) = AE(T)$, $|T - S^t| \leq K_b$, we have

$$AE(OPT) \leq AE(\emptyset) + \sum_{i=0}^{t-1} \rho_i + \sum_{v \in T-S^t} \rho_v^{\varphi(S'')}(S'') \quad (20)$$

where S'' is constructed from S^t in the same way as constructing S' from S. Now take $t = 0$, we have

$$AE(OPT) \leq AE(\emptyset) + K_b \times \rho_{max} \tag{21}$$

where $\rho_{max} = \max(\rho_v^{\varphi(S'')}(S''))$, $\forall v \in T - S^t$. So,

$$AE(OPT) - AE(\emptyset) \leq K_b \rho_{max} \leq K_b(AE(\Phi) - AE(\emptyset)) \tag{22}$$

where $AE(\Phi)$ is the availability of a backup solution picked by the picker. $\rho_{max} \leq AE(\Phi) - AE(\emptyset)$ holds based on **Lemma 1** and **Theorem 7**. So we have,

$$\frac{AE(OPT) - AE(\Phi)}{AE(OPT) - AE(\emptyset)} \leq \frac{K_b - 1}{K_b} \tag{23}$$

Since $(\frac{K_b-1}{K_b})^{K_b} \leq e^{-1}$ [13], we have the approximation ratio

$$\frac{e-1}{e} AE(OPT) + \frac{1}{e} AE(\emptyset) \leq AE(\Phi) \tag{24}$$

□

V. Evaluation

In this section, we use synthetic policy to evaluate our algorithm in terms of (i) SFC request acceptance ratio, (ii) backup resource consumed by requests, and (iii) accuracy of availability evaluation by the validator (Section IV-B).

A. Experimental Workloads

1) Physical Network: For physical networks, we use the network map and delay statistics of a large ISP network [5]. Each node of the network represents one single data center, which can provide three types of resources, namely CPU, memory and storage, with the capacity between [1500, 2500] units each. Each data center is associated with an ingress and an egress. The delay between the ingress/egress with their associated data center site is assumed to be between [1, 3] ms. We assume there are 10 types of functions in the network, and each of the data center sites can provide four to six functions. Each of the fiber links between the data center sites has a spectrum capacity of 16THz with a spacing of 12.5GHz per spectrum slot. The availability of each mapped VNF is randomly distributed within [0.9, 0.99].

2) Service Chain Requests: Each service chain request consists of two to six VNFs interconnected. Each VNF demands three types of resources and can provide one function, and the demand for each kind of resource is uniformly distributed between 0 and 30. Each logical link has a bandwidth demand among {10, 40, 100, 200} Gb/s with equal probability. For each service chain request, we select the availability requirement among {95%, 99%, 99.9%}, similar to the ones used by Google Apps [3]. The processing delay of a VNF is set to 50-150μs [9], and the end-to-end delay budget of each service chain request is set to 50-300ms [6].

We evaluate our algorithm using a Macbook with OS X 10.9 with 1.7 GHz Intel Core i7 processor and 8GB memory. Our algorithms are implemented in C++. The statistics are the average results.

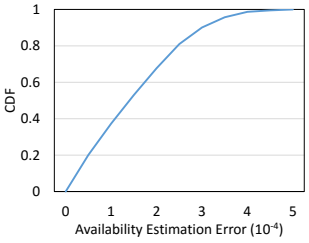


Fig. 3: CDF of the estimation error

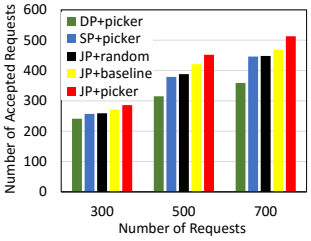


Fig. 4: Number of accepted requests

B. Availability Evaluation Accuracy

We first evaluate the availability evaluation accuracy achieved by validator proposed in Section IV-B. Table I summarizes the median estimation error when the number of VNF is randomly chosen from two to six and τ is varied.

TABLE I: Median Estimation Error Achieved by Validator

τ	0.03	0.05	0.07	0.09	0.11
Median Error (10^{-4})	1.63	1.55	1.38	1.71	2.2

From the table, we can see that when τ is 0.07, the estimation error is the smallest, and we fix τ to 0.07 for the rest simulations. With $\tau = 0.07$, we evaluate the *Cumulative Distribution Function* (CDF) of the estimation error of validator as shown in Fig. 3 when the number of request is 100 to ensure that the request acceptance ratio is 100%, and the availability threshold is set to 99.9%. We can observe that 90% of the error is smaller than 3.5×10^{-4}, which demonstrates the effectiveness of using the validator to predict the service availability.

C. SFC Request Acceptance Ratio

To understand how the picker and JP works, we compare the number of requests which can be accepted with different redundancy models and backup selection methods. Since a request can be accepted if and only if there is enough resource and the availability requirement can be met, the rationale behind this experiment is that an algorithm with better resource efficiency can accept more requests. From Fig. 4, we can see that JP + picker achieves the best acceptance ratio performance, and in particular, it outperforms SP and DP, both of which adopt the picker, by 15.1% and 42.8%, respectively when the number of request is 700. To show the effectiveness of our algorithm in picking backups, we compare the picker with a random selector (JP+random) which randomly selects two VNFs for each iteration to add a backup, as well as a baseline VNF selection algorithm (JP+baseline) in [12] which in each iteration the picker selects two primary VNFs whose availabilities are among the lowest. JP+random achieves almost the same performance as SP + picker, and JP+baseline performs about 9% worse than the optimal. Another interesting thing we can find from the figure is that when the number of requests is small (i.e., 300), all four algorithms have similar performance; while as the number of

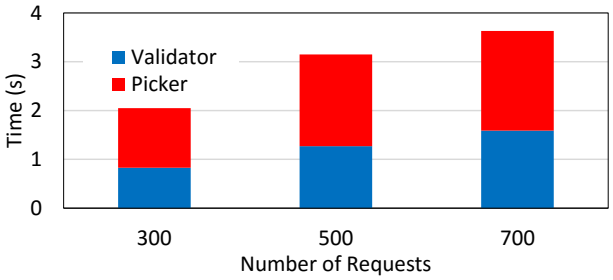


Fig. 5: Running time of picker and validator

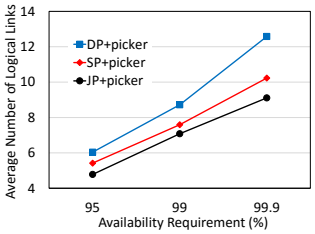


Fig. 6: Average logical links number w.r.t. different availability requirement

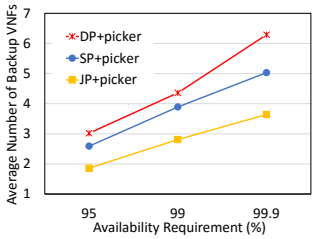


Fig. 7: Average backup VNF number w.r.t. different availability requirement

requests increases, the other three algorithms saturate faster than JP + picker. Therefore we can know that compared with other methods, JP + picker can meet availability requirement while consuming less resources. Furthermore, we analyze the running time of our algorithm as shown in Fig. 5. With 700 service chain requests, the total running time is less than 4 seconds, while the validator never uses more than 2 seconds.

D. Backup Resource Consumption

To further understand how JP + picker can save resources, we compare the average number of backup VNFs and logical links used for each service chain request w.r.t. different availability requirement. Here logical links refer to the links connecting a backup and their associated primary VNFs. The number of request used in this experiment is 700, and only the accepted requests are considered. As shown in Fig. 6, JP + picker uses 27.6% and 10.9% fewer links compared with the other two methods respectively when the availability requirement is "three nines" (i.e., 99.9%). Similar observations can be made when comparing the number of backup VNFs as illustrated in Fig. 7. We can see that JP + picker requires fewer number of backup VNFs. In particular, JP + picker can save up to 42.1% of backup VNFs.

VI. Conclusion

NFV explores the virtualization technologies to offer network-as-as-Services through connected/chained VNFs. Since telecom networks must be always on, it is critical to provide effective and efficient protection and resource allocation schemes for guaranteeing network service availability. In this paper, we propose an online algorithm for availability-aware SFC mapping for wide area service chaining, which can minimize the resources allocated to service chain requests while meeting clients' heterogeneous availability requirement. In addition we have developed a lower bound for our backup VNFs picking algorithm. Furthermore, we have shown that our design is able to evaluate service availability with a negligible estimation error in polynomial time. We have also validated our design through extensive simulations and demonstrated that it can achieve a significant performance improvement compared to traditional redundancy models and baseline backup VNFs picking method.

References

[1] Aryaka. www.aryaka.com.

[2] At&t domain 2.0 vision white paper. https://www.att.com/Common/about_us/pdf/AT&T\%20Domain\%202.0\%20Vision\%20White\%20Paper.pdf.

[3] Google apps service level agreement. http://www.google.com/apps/intl/en/terms/sla.html.

[4] Network functions virtualisation (nfv) resiliency requirements, 2015. http://www.etsi.org/deliver/etsi_gs/NFV-REL/001_099/001/01.01.01_60/gs_nfv-rel001v010101p.pdf.

[5] U.s. network latency. http://ipnetwork.bgtmo.ip.att.net/pws/network_delay.html.

[6] Verizon network infrastructure planning. http://innovation.verizon.com/content/dam/vic/PDF/Verizon_SDN-NFV_Reference_Architecture.pdf.

[7] vsphere. https://www.vmware.com/products/vsphere/features/fault-tolerance.

[8] A. Abujoda and P. Papadimitriou. Midas: Middlebox discovery and selection for on-path flow processing. In *COMSNETS*, pages 1–8, 2015.

[9] A. Basta, W. Kellerer, M. Hoffmann, H. J. Morper, and K. Hoffmann. Applying nfv and sdn to lte mobile core gateways, the functions placement problem. In *Proceedings of the 4th workshop on All things cellular: operations, applications, & challenges*, pages 33–38. ACM, 2014.

[10] C. J. Colbourn and C. Colbourn. *The combinatorics of network reliability*, volume 200. Oxford University Press New York, 1987.

[11] T. H. Cormen. *Introduction to algorithms*. MIT press, 2009.

[12] J. Fan, Z. Ye, C. Guan, X. Gao, K. Ren, and C. Qiao. Grep: Guaranteeing reliability with enhanced protection in nfv. In *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, pages 13–18. ACM, 2015.

[13] M. Feldman, J. Naor, and R. Schwartz. A unified continuous greedy algorithm for submodular maximization. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 570–579. IEEE, 2011.

[14] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 350–361. ACM, 2011.

[15] M. Jerrum and A. Sinclair. The markov chain monte carlo method: an approach to approximate counting and integration. *Approximation algorithms for NP-hard problems*, pages 482–520, 1996.

[16] J. Kwisthout. Most probable explanations in bayesian networks: Complexity and tractability. *International Journal of Approximate Reasoning*, 52(9):1452–1469, 2011.

[17] X. Meng, V. Pappas, and L. Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

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

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

[20] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. Design and implementation of a consolidated middlebox architecture. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 323–336, 2012.

[21] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24, 2012.

[22] Z. Ye, A. N. Patel, P. N. Ji, C. Qiao, and T. Wang. Virtual infrastructure embedding over software-defined flex-grid optical networks. In *2013 IEEE Globecom Workshops (GC Wkshps)*, pages 1204–1209. IEEE, 2013.

[23] J. Y. Yen. Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716, 1971.

[24] Q. Zhang, M. F. Zhani, M. Jabri, and R. Boutaba. Venice: Reliable virtual data center embedding in clouds. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 289–297. IEEE, 2014.

[25] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patneyt, M. Shirazipour, R. Subrahmaniam, C. Truchan, et al. Steering: A software-defined networking for inline service chaining. In *IEEE ICNP*, pages 1–10, 2013.

APPENDIX

In the appendix, we prove the availability-aware SFC mapping problem belongs to $\text{NP}^{\text{NP}^{\text{PP}}}$, which is commonly believed to be intractable. We first formally formulate the problem as a Boolean formula. Since a data center can provide a set of functions, and based on the joint protection method we propose, we need several auxiliary variables. Note that using SP instead cannot reduce the time complexity, and we can prove the same properties by changing the formulation slightly; using DP only simplifies the evaluation process, while this problem still remains hard. Given that JP can potentially brings more advantages in terms of effectiveness and resource consumption, here we only prove the case using JP .

For a data center n, p_n represents if n is used for mapping for the request, and q_n and $\bar{q_n}$ denote if n is used as a mapping site for a primary VNF or a backup VNF, respectively. Assume a site n can provide a set of functions F_n, then we use $\{x_n^i | n \in \mathbb{N}_D\}$ to show the set of functions site n provides. For example, given a physical network with 3 data centers n_1, n_2 and n_3, which can provide functions f_1, f_2, f_3, f_2 and f_1, f_3 respectively, and a service chain request which needs functions f_2, f_3, we can write the Boolean formula as

$$\begin{aligned}\varphi &= C(n_1) \wedge C(n_2) \wedge C(n_3) \wedge func(n_1, n_2, n_3) \\ &= (((p_1 \wedge q_1) \wedge (x_1^1 \wedge \bar{x_1^2} \wedge \bar{x_1^3}) \wedge (\bar{x_1^1} \wedge x_1^2 \wedge \bar{x_1^3}) \\ &\quad \wedge (\bar{x_1^1} \wedge \bar{x_1^2} \wedge x_1^3)) \vee ((p_1 \wedge \bar{q_1}) \wedge (x_1^1 \wedge x_1^2 \wedge \bar{x_1^3}) \\ &\quad \wedge (\bar{x_1^1} \wedge x_1^2 \wedge x_1^3) \wedge (x_1^1 \wedge \bar{x_1^2} \wedge x_1^3)) \\ &\quad \vee (\bar{p_1} \wedge \bar{x_1^1} \wedge \bar{x_1^2} \wedge \bar{x_1^3})) \wedge ((p_2 \wedge x_2^2) \vee (\bar{p_2} \wedge \bar{x_2^2})) \\ &\quad \wedge (((p_3 \wedge q_3) \wedge (x_3^1 \wedge \bar{x_3^3}) \wedge (\bar{x_3^1} \wedge x_3^3)) \\ &\quad \vee ((p_3 \wedge \bar{q_3}) \wedge (x_3^1 \wedge x_3^3)) \vee (\bar{p_3} \wedge \bar{x_3^1} \wedge \bar{x_3^3})) \\ &\quad \wedge (((x_1^2 \wedge x_1) \vee (x_2^2 \wedge x_2)) \wedge ((x_1^3 \wedge x_1) \vee (x_3^3 \wedge x_3)))\end{aligned}$$

where $C(n_i)$ shows the constraints for site i, and $func(n_1, n_2, n_3)$ indicates the functions provided by each site. x_1, x_2 and x_3 is 1 if this site can function normally at a given time and 0 otherwise. We are trying to find the minimum number of backup VNFs needed to be selected, which is equivalent to finding the maximum set $A \in \mathbb{N}_D$, and the value of each element p_n in this set can be set to 0 such that the rest can satisfy the Boolean formula with probability greater or equal to certain threshold. As the first step, we show that verifying if the availability is above clients' requirement in a polynomial time is not a viable option.

Theorem 9. *Verifying if the availability of a given deployed service chain with backups, donated as problem VA, is above a given threshold is PP-complete.*

Proof. By the definition of language in PP, it is clear problem VA is in PP. Note that MAJSAT [16] is a PP-complete problem. To show PP-completeness, we can reduce MAJSAT problem to problem VA. Note that, for an instance ϕ with n variables of MAJSAT, the number of all possible assignments to ϕ is 2^n. Thus, we have

$$\begin{aligned}\phi \in MAJSAT &\iff \text{the number of assignments that} \\ &\qquad \text{satisfies } \phi \text{ is greater than } 2^{n-1} \\ &\iff \Pr[\phi(x)] > \frac{1}{2} \text{ with } x \in \{0,1\}^n\end{aligned}$$

Given that problem VA's instance is a pair (ϕ, θ) consisting of a Boolean formula ϕ and a threshold θ. Hence, with a MAJSAT instance ϕ, we can set instance $(\phi, 1/2)$ for problem VA. To verify the correctness,

$$\begin{aligned}\phi \in MAJSAT &\iff \Pr[\phi(x)] > \frac{1}{2} \text{ with } x \in \{0,1\}^n \\ &\iff \text{the probability that a given formula} \\ &\qquad \phi \text{ can be satisfied is greater than} \\ &\qquad \text{a given threshold } \frac{1}{2} \\ &\iff (\phi, \frac{1}{2}) \in VA,\end{aligned}$$

$$\begin{aligned}\phi \notin MAJSAT &\iff \Pr[\phi(x)] \leq \frac{1}{2} \text{ with } x \in \{0,1\}^n \\ &\iff \text{the probability that a given formula} \\ &\qquad \phi \text{ can be satisfied than a given} \\ &\qquad \text{threshold } \frac{1}{2} \\ &\iff (\phi, \frac{1}{2}) \notin VA.\end{aligned}$$

Thus, it is a valid many-one reduction from MAJSAT problem to problem VA. Therefore, problem VA is also PP-complete. □

Even with an oracle to the problem VA, we still cannot optimally decide if there exists a solution for a certain request, and finding a local optimal is difficult.

Theorem 10. *Determining if there exists a solution for a service chain request, denoted as problem DE, is NP^{PP}-complete.*

Proof. We can construct a nondeterministic oracle Turing machine N with oracle that solves problem VA, which conducts the following three steps to solve the given instance (A, ϕ, θ) of problem DE:

1) Randomly guess a solution to set A, which takes time $O(|A|)$
2) Hardcode the guess to ϕ to obtain ϕ', which takes time $O(|\phi|)$
3) Query the VA oracle with (ϕ, θ)

Since $O(|Y|)$ and $O(|\phi|)$ are polynomials in terms of n, this satisfies the definition of NP$^{\text{PP}}$ class. Hence, problem DE is in NP$^{\text{PP}}$. To show NP$^{\text{PP}}$-completeness, we reduce E-MAJSAT [16] to problem DE. In E-MAJSAT problem, for an instance (k, ϕ) (we represent a sequence of variables x as $x_1 x_2 \cdots x_n$),

$$\begin{aligned}(k,\phi) \in E-MAJSAT &\Longleftrightarrow (\exists\ x_1x_2\cdots x_k \in \{0,1\}^k) \\ &\quad \sharp(\text{assignments to } x_{k+1}\cdots x_n \\ &\quad \text{that satisfies } \phi) > 2^{n-k} \\ &\Longleftrightarrow (\exists x_1x_2\cdots x_k \in \{0,1\}^k) \\ &\quad \Pr[\phi(x)|x_1x_2\cdots x_k] > \frac{1}{2},\end{aligned}$$

where $\sharp(Y)$ denotes the number of elements in set Y. With such an E-MAJSAT instance, we first define a set of variables as $A = x_1, x_2, \cdots, x_k$ and then set instance $(A, \phi, 1/2)$ for problem DE. To verify correctness

$$\begin{aligned}(k,\phi) \in E-MAJSAT &\Longleftrightarrow (\exists\ x_1x_2\cdots x_k \in \{0,1\}^k) \\ &\quad \Pr[\phi(x)|x_1x_2\cdots x_k] > \frac{1}{2} \\ &\Longleftrightarrow \text{there exists a solution to set} \\ &\quad \text{A such that the probability} \\ &\quad \text{that a given } \phi \text{ can be} \\ &\quad \text{satisfied is greater than} \\ &\quad \text{a given threshold} \\ &\Longleftrightarrow (A, \phi, \frac{1}{2}) \in DE,\end{aligned}$$

$$\begin{aligned}(k,\phi) \notin E-MAJSAT &\Longleftrightarrow (\nexists\ x_1x_2\cdots x_k \in \{0,1\}^k) \\ &\quad \Pr[\phi(x)|x_1x_2\cdots x_k] > \frac{1}{2} \\ &\Longleftrightarrow (\forall x_1x_2\cdots x_k \in \{0,1\}^k) \\ &\quad \Pr[\phi(x)|x_1x_2\cdots x_k] \leq \frac{1}{2} \\ &\Longleftrightarrow \text{there DOESNOT exist a} \\ &\quad \text{solution to set A such that} \\ &\quad \text{the probability that a given} \\ &\quad \phi \text{ can be satisfied is greater} \\ &\quad \text{than a given threshold} \\ &\Longleftrightarrow (A, \phi, \frac{1}{2}) \notin DE,\end{aligned}$$

Thus, this is a valid many-one reduction from E-MAJSAT problem to DE problem. Therefore, problem DE is NP$^{\text{PP}}$-complete. □

Let LM denote the problem of finding a local optimal solution for a chain request. Now we can define the decision problem form of LM.

Definition 2. *Does there exist a set of backups of size that can make the availability of a given service chain request γ_i at least θ, given a set of sites A, where each element is set to 0?*

We denote this decision problem version of LM as DLM. We can see that the input to DLM is a set of backups along with ϕ and θ, and the output is a boolean value. Apparently, we can solve DLM once we can solve LM. This means that LM is at least as hard as DLM. Moreover, LM and DLM are actually equivalent. To prove that, we can just prove the following lemma.

Lemma 2. *DLM is at least as hard as LM.*

We can prove this lemma by constructing an algorithm solving LM using a DLM problem oracle, denoted as $\mathcal{O}_{DLM}(A, \phi, \theta)$. The algorithm solving LM with a given chain request ϕ and a threshold θ as input is as follows:

Algorithm 2 Deciding local optimal

```
S ← {n_1, ··· , n_{|N_D|}}
A ← ∅
for k from 1 to n do
    A ← A ∪ {b_k}
    result ← O_DLM(A, φ, θ)
    if result = FALSE then
        A ← A\{b_k}
    end if
end for
return A
```

Note that the above algorithm is actually a polynomial-time Turing reduction. This means via the above algorithm, LM can be solved with a DLM oracle, which indicates that DLM is at least as hard as LM. Overall, we obtain that DLM and LM are equivalent.

Lemma 3. *The decision problem DLM is co-NP^{PP}-complete.*

Proof. To show co-NP$^{\text{PP}}$-completeness, we need to prove that 1) DLM is in the co-NP$^{\text{PP}}$ class and 2) DLM is co-NP$^{\text{PP}}$-hard. For 1), since the worse case happens when all the variables in A are set as 0, then by definition of co-NP, such case satisfies means all other assignments to A also satisfy the boolean function. Overall, it becomes for all assignments to A, at least half of the assignments to the rest backups satisfy the boolean function, which is computed using a PP oracle. Hence, we can see that 1) follows from the definition of class co-NP$^{\text{PP}}$. Next, to prove 2), we can construct a many-one reduction from A-MAJSAT problem to this problem. Concretely, note that the formula ϕ in the input instance (k, ϕ) is not necessarily of monotone form, while the Boolean formula in instance for problem DLM is monotone. Hence, in order to transfer an A-MAJSAT instance to a DLM instance, we need to employ the standard way of converting general Boolean formula to monotone CNF Boolean formula ϕ'. Observe that, if fixing a subset of variables A, in such a monotone CNF Boolean for-

mula, the number of satisfying assignments to ϕ' is minimum with all variables in A set to be 0. It is trivial to derive such a set A from k in A-MAJSAT instance. At this point, we set a threshold θ as $\frac{1}{2}$ and then get an instance $(A, \phi', \frac{1}{2})$. With this $(A, \phi', \frac{1}{2})$ as input to the DLM oracle, if it outputs TRUE, then it means the majority of the assignments are satisfying when all the variables in A are set as 0 (which is the minimum case). Thus, for A being set to other assignment, the majority of the assignments must satisfies ϕ'. Hence, we can use an oracle solving problem DLM to solve A-MAJSAT problem. It is easy to check the validity of this many-one reduction. Since A-MAJSAT problem is co-NP$^{\text{PP}}$-complete, therefore, problem DLM is also co-NP$^{\text{PP}}$-complete. □

Theorem 11. *Finding a local optimal solution for a chain request is co-NP^{PP}-complete.*

Proof. Based on **Lemma 2** and **Lemma 3**, this theorem immediately follows. □

The objective of globally minimizing the number of backup VNFs further elevates the time complexity.

Theorem 12. *Finding the optimal solution for one service chain request belongs to $NP^{NP^{PP}}$.*

Proof. By the definition of NP$^{\text{NP}^{\text{PP}}}$ [16], we can construct a polynomial-time bounded nondeterministic oracle Turing machine M to accept our problem with oracle to a problem in co-NP$^{\text{PP}}$ as follows:
Taken an instance (ϕ, θ) of our problem as input

1) Randomly guess a maximum set A that will meet our property
2) Use (A, ϕ, θ) to access the DLM oracle to decide the maximal set

Clearly, before verifying, the randomly generated set A by machine M is just a possible candidate for maximum set. To make sure that, M uses DLM oracle to verify the correctness. Note that we reply on the power of nondeterministic Turing machine to guess a possible solution, which takes $O(n)$ time. Also DLM is co-NP$^{\text{PP}}$-complete. Therefore, our problem is in NP$^{\text{NP}^{\text{PP}}}$. □