A Framework for Provisioning Availability of NFV in Data Center Networks

Jingyuan Fan, Meiling Jiang, Ori Rottenstreich, Yangming Zhao, Tong Guan, Ram Ramesh, Sanjukta Das, and Chunming Qiao, *Fellow, IEEE*

***Abstract*— Network function virtualization 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 network functions are deployed over virtualized and shared platforms in data centers. However, failures are quite common in data centers. Therefore, a practical and yet theoretically challenging issue in SFC mapping in such an environment is to manage the availability of the requests. In this paper, we present a framework to provision availability of SFC requests in a data center with multiple layers of connected devices, and the devices follow heterogeneous failure processes with the objective of minimizing resource usage. To expedite the process, we further propose an optimization problem of request mapping and backup estimation and solve it efficiently with an approximation algorithm. With simulations, we demonstrate the effectiveness of our proposed framework.**

***Index Terms*— Data centers availability, network function virtualization, service function chaining.**

I. Introduction

AS AN advanced application of virtualization technologies, Network Function Virtualization (NFV) eliminates the need for dedicated middleboxes and enables the allocation of computing and networking resources in a flexible manner. In the NFV environment, traditional hardware-based network appliances are replaced by software-based virtual network functions (VNFs), which implement network functions in software. These VNFs are deployed on virtual platforms where clients (each associated with a sequence (chain) of required funcions) may share the same physical hardware through a process named Service Function Chain (SFC) mapping.

However, one of the most important issues of a successful NFV deployment is to manage the availability of the requests. Compared to traditional IT application with availability of the order of two 9s to three 9s (i.e., 99% and 99.9% of the time), telecom service requires their network being "always on" (i.e., five 9s or six 9s of the time [22]).

Managing availability, in practice however, is not easy. The challenge is multifold: 1) a data center, which holds VNFs, includes connected devices, such as servers, switches and links, each having multiple layers (e.g., software and hardware), and can have various topologies; 2) device failures are quite common and the failure and repair processes of each device in a data center can experience heterogeneous probability distributions; 3) failures of different devices can have different influences on service availability, e.g., a virtual machine (VM) vs. a core switch; 4) the availability of an SFC request can vary with its location in the data center; 5) redundancy may need to be provisioned in order to meet the availability requirement; 6) determining the optimal amount of backup resources in such a dynamic data center environment, which is important to reduce resource consumption, makes the problem even harder. Therefore, there is a need for a framework to easily estimate the availability of SFC requests as well as to allocate the minimum amount of resources needed to meet heterogeneous availability requirements, for any given data center topology with heterogeneous failure and repair processes of each device.

Despite the importance of provisioning availability in NFV, few solutions have been suggested. Fan *et al.* [16], [17] proposed online algorithms for availability guarantee with on and off-site backups. However, they did not consider the heterogeneous processes of a device's failure and repair, and they assumed that only VMs would experience failures and ignore others, such as switches and links, which also have been identified as a major contributing factor to downtime [18]. On the other hand, Schlinker *et al.* [42] provided a framework to evaluate fault-tolerance in user-defined data center topologies while considering various types of devices, but a mechanism to guarantee availability was not described.

This work presents an framework to iteratively provision availability of SFC requests in a data center. Devices in the data center have heterogeneous failure and repair processes. We focus on the case where SFC requests are implemented with the widely used consolidated middlebox structure [5], [19], [43], in which each VNF is an application that can be realized with certain number of VMs installed on the physical machines and an entire service chain running as a native process at a single physical machine [23], [26], [38], [45]. To the best of our knowledge, none of the existing works have addressed similar problems. Specifically, when a request

Manuscript received May 3, 2018; revised August 19, 2018; accepted August 28, 2018. Date of publication September 13, 2018; date of current version November 28, 2018. This work was supported by NSF under Grant EFRI 1441284. *(Corresponding author: Yangming Zhao.)*

J. Fan, Y. Zhao, T. Guan, and C. Qiao are with the Department of Computer Science and Engineering, University at Buffalo, The State University of New York, Buffalo, NY 14260 USA (e-mail: jfan5@buffalo.edu; yangming@buffalo.edu; tongguan@buffalo.edu; qiao@computer.org).

M. Jiang, R. Ramesh, and S. Das are with the Department of Management Science and Systems, University at Buffalo, The State University of New York, Buffalo, NY 14260 USA (e-mail: mjiang5@buffalo.edu; rramesh@buffalo.edu; sdsmith4@buffalo.edu).

O. Rottenstreich is with the Department of Computer Science, Technion-Israel Institute of Technology, Haifa 3200003, Israel, and also with the Department of Electrical Engineering, Technion-Israel Institute of Technology, Haifa 3200003, Israel (e-mail: ori.rot@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2018.2869960

comes, we map it to the data center, evaluate its availability and provision backups until its requirement is met. To this end, we first show a mechanism to calculate the downtime of an SFC request in such an environment via a Markov Chain Monte Carlo (MCMC) like procedure. Even though this method is very accurate, it suffers from low efficiency [48]. Such problem is further exacerbated by the iterative nature of our algorithm. To address the challenge, we propose an optimization problem that estimates the number of backups required and aims at minimizing the resource usage upon the arrival of a request, while considering both request placement and backup provisioning, thereby significantly reducing the number of iterations needed of our framework. To efficiently solve the optimization problem, we propose an approximation algorithm with a lower bound on the availability that can be achieved. The performance of our framework is validated through simulations.

In summary, our contribution is multifold: 1) to the best our knowledge, no work has considered the end-to-end availability problem with each device's state modeled as an alternating renewal process. Such a model is closer to reality, but considering it makes our problem significantly harder; 2) to resolve the complexity, we proposed an optimization problem to estimate the number of backups needed while reducing the resource usage; 3) we gave an approximation algorithm to solve the problem.

The rest of the paper is organized as follows. Section II describes the availability model of a single device as well as VNFs used in the paper. In Section III, we describe the overall structure and design of our proposed framework and show how it works with a case study. Section IV defines the problem of request mapping and backup estimation in NFV, which is part of the framework, and presents an algorithm with a theoretical bound on the availability that can be achieved. We evaluate its performance in Section V, survey related work in Section VI and briefly discuss the usage issues of our framework and future works in Section VII, followed by conclusions VIII.

II. Availability Model

A. Availability Model of a Single Device

The availability of a component is often considered as 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, existing works [15], [16] considering availability assume that the availability of a device can be calculated using uptime followed by downtime:

$$A = \frac{Uptime}{Uptime + Downtime} = \frac{MTBF}{MTBF + MTTR} \tag{1}$$

where MTBF and MTTR are *Mean Time Between Failure* and *Mean Time To Repair*, respectively.

However, such model fails to capture the non-deterministic characteristics of devices in practice [6], [14]. To have a more accurate analysis, in this paper, we model a device's failure as an alternating renewal process (ARP), as shown in Fig. 1. An ARP models a system that alternates between two states (*on* and *off*) over time. $\boldsymbol{W} = ((U_1, V_1), (U_2, V_2), \ldots)$ are the pairs of times successively spent in the two states, where the sequence $\boldsymbol{U} = (U_1, U_2, \ldots)$ and $\boldsymbol{V} = (V_1, V_2, \ldots)$ are the successive lengths of time that the system is in state *on* and *off*, respectively and both of them are assumed to be independent and identically distributed random variables. In other words, the system starts in state *on* and remains in that state for a period of time U_1, then goes to state *off* and stays in this state for a period of time V_1, then back to state *on* for a period of time U_2, and so forth. With such a model, even computing availability becomes a hard problem, which will be described and addressed in the next section.

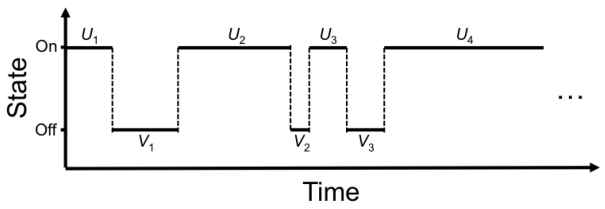


Fig. 1. Alternating renewal process.

B. Availability Model of Consolidated Middleboxes

Since any device can fail at any time, redundancy is necessary to maintain a certain level of availability of a device. In this paper, we consider the active/passive failure configuration [51]. In the active/passive setting, the backups are kept idle when not in use, while they can assume the role of the primaries when they fail. The backup VMs, in order to do so, should pre-load images of VNFs in memory without running these images. Note that we assume in this work the switching time from primaries to their backups, including the time to transfer states for stateful VNFs, is zero, and will extend our analysis in a future work.

Therefore, given an SFC with M VNFs, each of which requires $n_1, n_2, \ldots, n_M$ VMs respectively, we consider the service as available if 1) the i-th VNF should have at least n_i VMs available, where $i = 1, 2, \ldots, M$, and the physical machine holding these VMs is available; 2) there should be at least one available path (including links and switches/routers along the links) between the physical machine and any one of the access switches (the highest level switches) of the data center.

As we consider a consolidated middlebox model in this work, the availability of an SFC can be bounded by the availability of the physical machines where the VMs implementing the SFC reside. The failure of the physical machine can result in a violation of the availability requirement no matter how many backup VMs are allocated on it. For example, as shown in Fig. 2, even though a datacenter network is by design tolerant to failures of switches and links, with resource redundancy that contribute to the multi-path connections between a server (e.g., s_{11}) and the highest level switches (as shown by the dashed lines), it may still not be enough to achieve certain availability requirements. In other words, the probability that there is at least one path available from s_{11} to any one of the core switches is already lower than the required one. Thus, VMs should also be allocated at physical machines

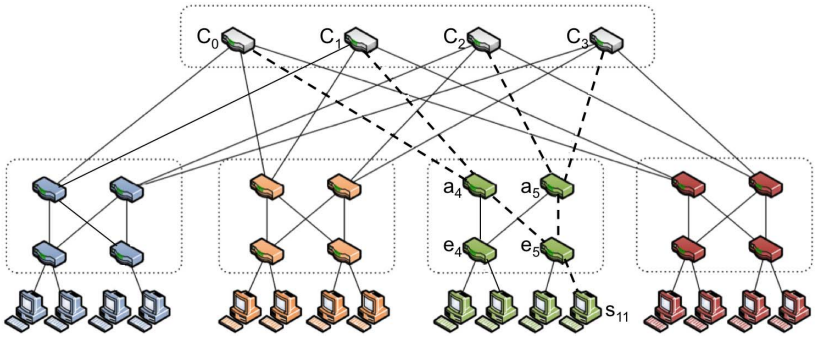


Fig. 2. Case study: a 4-ary Fat-tree data center topology [4].

Algorithm 1 The Framework for Provisioning Availability for the Request r

1: Map the request r with an availability requirement α_r onto the network
2: **while** true **do**
3: **if** CALCULATE_AVAILABILITY$r < \alpha_r$ **then**
4: Provision backups to the request r
5: **if** there is not enough resource **then**
6: **break**
7: **else**
8: Go to the next request

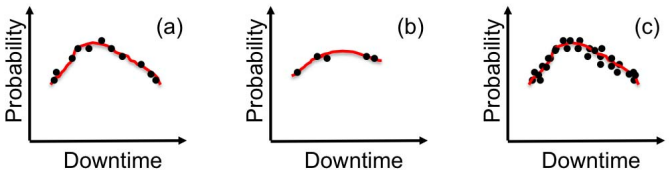


Fig. 3. Maximum likelihood estimation of the downtime distribution with: (a) an appropriate sample size; (b) undersampling; (c) oversampling. The X-axis is downtime and the Y-axis is probability.

besides the primary one; and therefore, the number of VMs allocated at a secondary physical machine should be at least $\sum_{i=1}^{M} n_i$, and multiple secondary physical machines may be required. When there are multiple physical machines used, at least one of them should satisfy the conditions aforementioned for this service to be available.

C. Assumptions

In this paper, we assume that the failure and repair processes of each and every device (including physical machines, VMs, routers, switches, etc.) is modeled by an ARP with heterogeneous $\boldsymbol{U}$ and $\boldsymbol{V}$ sequences, and the distributions from which each ARP's $\boldsymbol{U}$ and $\boldsymbol{V}$ are drawn are stationary. In this work, we assume that only VMs (i.e., VNFs) have backups and there is no correlated failure between devices and the failure of each device is independent for simplicity.

III. Framework for Provisioning Availability

A. Overview

As provisioning availability of an SFC request with the failure probability of each device being a constant is already an NP-hard problem [15], using an ARP model for evaluating availability further increases the complexity, and analysis on how to guarantee availability in the existing works [15], [16] cannot be applied directly.

To address the problem, we apply an iterative approach to increase the availability of an SFC request by provisioning backups until either the availability requirement is met or no more resource is available, as shown in **Algorithm 1**. Specifically, given the topology of a data center as well as the failure and repair probability distributions of each device, and an SFC request r with availability requirement α_r, the framework will first map the request onto the topology (line 1, see more details in Section IV), calculate its availability (function CALCULATE_AVAILABILITYr of line 3). If the availability of the request cannot meet the user's requirement, it will provision some backups (line 5) to the request and repeat the above process.

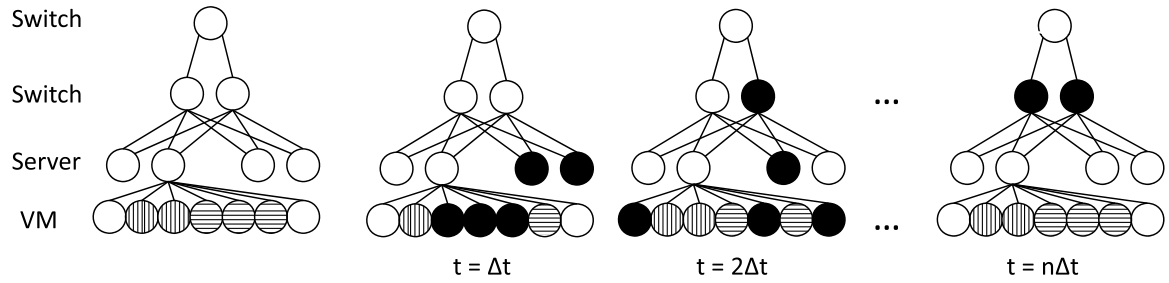
As in [42], a data center topology can be described as an abstract tree with a root node, where if a father node fails, all its children cannot be reached via this father node. Some nodes can have multiple layers of failures and they may have different influences. For example, a switch can contain multiple ports. The failure of a switch can be seen as the failure of the whole switching fabric and consequently none of the devices connected to the switch are reachable from the switch, while the failure of a port does not affect the devices connected to other ports of the switch.

For the rest of the section, we will focus on how to calculate the availability in Section III-B (line 6), and how to add backups in Section III-C (line 8), while leaving request mapping to the next section.

B. Calculating Availability for SFC Requests

To estimate the availability of a mapped request over its contract period duration, the main idea is to sample the potential states (on/off) of each component that may fail at different time intervals.

To this end, we divide the contract period duration T into small slots Δt such that within a Δt, we assume that the state of a physical component (i.e., *on* or *off*) keeps unchanged. Since the failure of a device can be modeled as an alternating



(a) Mapping an SFC request (b) Estimating downtime distribution of a mapped SFC request

Fig. 4. A toy example of how to estimate availability of a mapped SFC request (circles filled with vertical and horizontal lines represent VMs belonging to two different requests; black circle means the device is unavailable).

renewal process, the state of a device at time $n\Delta t$ only depends on the state at time $(n-1)\Delta t$. Therefore, we can emulate the state transition of each device by sampling its corresponding failure and repair probability distributions at each time slot.

Then at each time slot, the breadth-first search (BFS) algorithm is used to see how many physical machines used for an SFC request we can "reach" from the top of the constructed abstract tree. Here a reachable physical machine means all components on at least one path from the top of the tree to this physical machine are in *on* state. For a reachable physical machine, if any one of the VNFs of this SFC request does not have the required number of VMs, we consider this physical machine as being down at this time slot for this request. If all reachable physical machines are down, we consider this time slot as downtime for this request.

By running the above BFS T consecutive times and counting the number of time slots that are down for the request, we can obtain the total downtime during the time interval $[0, T]$, which can be considered as one sample from the unknown downtime distribution. Then we repeat the above procedure for N times (so that we have N samples), and calculate the frequency of each downtime that occurs in our samples S, where $|S| = N$:

$$P_{MLE}(X = s) = \frac{\text{COUNT}(s)}{N}, \quad s \in S \tag{2}$$

which equals the maximum likelihood estimation (MLE) of the downtime distribution of the request. Then we apply kernel density estimation (KDE) [39] to generate the probability density function (PDF) of f of the downtime, as shown in Fig. 3(a). Thus the availability can be estimated as $1 - \int_0^\infty f \, \mathrm{d}t$.

A toy example is shown in Fig. 4. An SFC request with two VNFs, each of which requires 1 and 2 VMs indicated by vertical and horizontal lines (the rest can be considered as backups) respectively, is mapped on a data center with two layers of switches, as shown in Fig. 4a. Fig. 4b depicts the procedure of estimating its availability. A black circle indicates the device being unavailable at that time slot. At $t = \Delta t$, the number of available VMs is less than the requirement; while at $t = n\Delta t$, the VMs are not reachable from the top switch even there are enough available VMs. Therefore, the service should be considered as unavailable during these two time slots. On the other hand, at $t = 2\Delta t$, the service is available as both conditions discussed in Section II-B can be met. Clearly with more backup VMs and physical machines, the chance that a mapped request can meet its availability requirement is higher.

In order to get the downtime distribution estimation, we first need to understand how many samples are needed. The estimated distribution cannot converge if it is under sampled (Fig. 3(b)), but oversampling (Fig. 3(c)) can lead to too much computational overhead. To this end, we empirically evaluate this by setting a relatively large initial sample size, taking progressively larger sample sizes (e.g., 1000 more each time), and deriving the estimated PDF for each until the difference between the estimated downtime distribution and the reference one, which can be acquired by evaluation with a very large sample size, is smaller than ϵ, where ϵ is a small number. The difference between two distributions can be measured in various ways, such as Kolmogorov-Smirnov test [10], and Kullback-Leibler divergence [29]. Note that simply comparing the estimated availability (i.e., the integral of the estimated downtime distribution) is not good enough as a small sample size may result in a similar expected availability to the reference one while a distribution looks quite different.

C. Provisioning Backups

If the estimated availability is smaller than the required one, backups are provisioned. Meantime, we want to minimize the number of backups allocated to 1) save resources for other requests; and 2) reduce costs. Assume an SFC request r that needs M_r VNFs, each of which requires $\langle n_1^r, n_2^r, \ldots, n_{M_r}^r \rangle$ VMs respectively, is given. We use **Algorithm 2** to allocate backups (line 8 in **Algorithm 1**). Each time a backup needs to be provisioned, the function ADD_BACKUPS is called. For ease of presentation, we denote $\langle n_1^r, n_2^r, \ldots, n_{M_r}^r \rangle$ as V_r.

The key idea of the algorithm is trying to ensure that the number of backups allocated for the i-th VNF is proportional to the number of VMs it initially requires (i.e., n_i^r) while keeping at most one backup VM allocated to each VNF each time. The intuition here is that if a VNF requires more VMs, more backups should be allocated as it has a higher chance to have at least one VM down while keeping the increase of VM usage to a minimum.

To this end, we build **Algorithm 2** based on iterative water-filling algorithm [52]. First V_r is scaled down with the

Algorithm 2 Provisioning Backups

```
Initialize:
n_min = min (n^r_1, ..., n^r_{M_r})
unit = [⌊n^r_1 / n_min⌋, ⌊n^r_2 / n_min⌋, ..., ⌊n^r_M / n_min⌋]
backup = [0, 0, ..., 0]
times = 1
function ADD_BACKUPS(r)
    if backup_i ≥ unit_i × times   ∀i ∈ [1, M_r]   then
        times++
    for i = 1 to M_r do
        if backup_i < times × unit_i then
            backup_i = backup_i + 1
    return backup
```

scale factor $\min_i n_i^r$, denoted as `unit`, and it (multiplied by `times`) is used as the "water level" for deciding if more backups should be allocated to a VNF with `times` increases as the "level" is evened. For example, assume that $V_r = \langle 10, 3, 7 \rangle$, then $\mathtt{n_{min}} = 3$ and $\mathtt{unit} = \langle 3, 1, 2 \rangle$. In this case, `times` is increased every three times the function is called. When the function is called the first time, one backup is allocated to each of the VNF; one is allocated to the first and third VNFs respectively the next time the function is called; if the function is invoked again, one VM is provisioned to the first VNF. This loop continues if more backups are needed with increasing `times` by one.

As mentioned in the previous section, the availability of an SFC is bounded by the availability of the physical machines that holds the VMs implementing this SFC. Thus sometimes multiple physical machines are needed for one request. To minimize the usage of physical machines, only when adding backups to a physical machine cannot increase the availability, we select a new physical machine, and allocate $\sum_{i=1}^{M_r} n_i^r$ VMs on the new physical machine for the request. Backups, if still required, are only provisioned to this new physical machine until a new physical machine is added.

D. Case Study

In this subsection, we show how our framework works through an example. Consider an SFC request r and a 4-ary Fat-tree data center topology [4] as shown in Fig. 2. The request has an availability requirement of 99% for a contract period duration of 30 days. It requires 4 VNFs, each of which requires 3, 10, 21, 15 VMs, respectively. Note that our framework also works for other types of topologies, such as DCell [21] and BCube [20], which are recursively defined. Δt is set to 1 minute, and a Gaussian kernel is applied for KDE. In this example, the request is mapped initially to a randomly selected physical machine in the network.

Fig. 5 presents how the availability improves with different number of backups through the changes in downtime distribution. At the very beginning, no backup (red line) is added to the request. Based on the distribution, the availability can be estimated as 0.902. When adding more backups, the availability keeps increasing (0.981 for blue line and 0.989 for green line) until achieving the availability requirement (0.992 for purple line). Note that after we add 7 backups to the request (green line), the availability cannot be further improved[1] by adding more backups to the same physical machine (which is not shown in the figure); and therefore, we add one more physical machine with 49 VMs (0 backup VM), re-estimate the availability and get the purple line. When there are two physical machines, a time slot is count as uptime if at least one of the machines has at least 49 VMs and that machine is reachable from one of the core switches. In this study, the sample size $|S|$ is at least 4,000 for the estimated downtime distribution to converge. In general, we find that the sample size needs to be quite large.

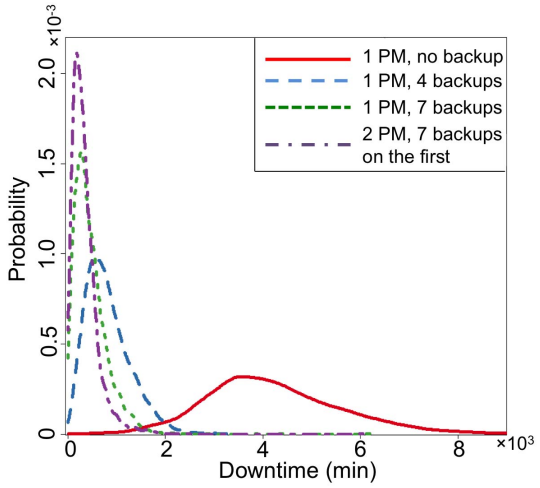


Fig. 5. Downtime estimation with different number of backups (PM: physical machine).

E. Increasing Estimation Efficiency

One drawback of the above method is the extremely low efficiency. The total running time of a naive implementation of a solution for the example in the previous subsection, written in non-optimized Python code, takes more than 1,100 hours on a Linux machine with 3.60GHz Intel i7-4790 cores and 12GB memory. To improve the computational efficiency, we made a few optimizations to our implementation:

Mapping requests with selected physical machines. Since the availability that an SFC achieves is bounded by the availability of the physical machines implementing this SFC, we can always choose physical machines with higher availability to implement SFC requests with stricter requirement. To this end, we define the *path availability* of a physical machine as the probability that all components of at least one path from the top of the hierarchy network to this physical machine are up. Take the upside down tree shown by dashed lines in Fig. 2 as an example, we should have the server s_{11} to be available, the edge switch e_5 to be available and at least one the two options: 1) a_4 is available together with at least one

[1]More precisely, with 7 backups, the availability of this VNF is very close to the availability of the physical machine, which is the availability upper limit of the VNF.

of c_0 or c_1; or 2) a_5 is available together with at least one of c_2 or c_3. Therefore, the *path availability* of physical machine s_{11} can be calculated as

$$G(s_{11}) = \alpha_{s_{11}} \times \alpha_{e_5} \times (1 - (1 - \alpha_{\theta_1}) \times (1 - \alpha_{\theta_2}))$$

where

$$\begin{aligned} \alpha_{\theta_1} &= \alpha_{a_4} \times (1 - (1 - \alpha_{c_0}) \times (1 - \alpha_{c_1})) \\ \alpha_{\theta_2} &= \alpha_{a_5} \times (1 - (1 - \alpha_{c_2}) \times (1 - \alpha_{c_3})) \end{aligned} \tag{3}$$

where α_d is the availability of device d. Note that in this example, we just consider the failures of switch and server, but one can extend it to include links, ports, etc. With the path availability, one can estimate how many physical machines are needed at least.

Pruning the search space. Consider the same example. Assume that an SFC request is mapped to the physical machine s_{11}, then instead of including all nodes in the abstract tree in the search space, only searching over nodes that are used in calculating its *path availability* are necessary and sufficient. In our case, these nodes are s_{11}, e_5, a_4, a_5, c_0, c_1, c_2, and c_3. With a data center topology with a very large number of switches and physical machines, eliminating unnecessary nodes in the search space can lead to a significant performance improvement.

Parallelizing the search. Clearly, the BFS algorithm can be easily parallelized and/or distributed. Each core can run fewer iterations, and all the samples can be merged together at last to get the estimation. By default, all available cores are used by our framework.

With parallelism and the other two optimizations, we run the same experiments as the one in the case study on Dell PowerEdge R720 with Intel Xeon E5-2600 cores and 128 GB memory. It took around 40 hours on 40 non-dedicated cores. The computational efficiency can be further improved with more cores/machines.

However, even though the efficiency is greatly improved, our framework is still not ideal in the sense that the process can be further prolonged with 1) an even higher availability requirement, which is normal in a carrier-class network [22] (usually requires an availability of 99.999% or 99.9999% (five 9s or six 9s)) and 2) longer contract period duration. A question arises naturally: *can we further reduce the running time?*

IV. Optimizing Availability Provisioning in NFV

To answer the question raised in the previous section, we first need to understand what are the major contributing factors. Recall that in order to estimate the downtime distribution, we need to run simulations similar to MCMC [48], and for the estimated downtime distributed to converge, a large sample size is needed. Such procedure, in spite of being accurate, is time consuming by nature. Even with a much simpler model, the work in [42] still requires a long time to evaluate availability/reliability. In addition, iterations are needed to improve availability to meet requirements.

Fortunately, there is still room to alleviate the situation and our insight here is that if we can accurately estimate the number of backups that each request needs, instead of just mapping the required VMs when a request arrives, we can map them together with backups (line 1 in **Algorithm 1**). Therefore, when we evaluate the availability (line 6 in **Algorithm 1**), a mapped request can have an availability very close to its requirement, thereby reducing the total running time. In this section, we will model the request mapping and backup provisioning as an optimization problem with the objective of minimizing the total amount of resources usage and show an approximation algorithm to efficiently solve it.

A. How to Deal With Heterogeneous Failure Processes of Devices?

The first problem to deal with is the heterogeneous failure processes, which are modeled as alternating renewal processes, of devices. Let $\mu = \mathbb{E}(\boldsymbol{W})$ denote the mean of a time period that the device is *on* and $\nu = \mathbb{E}(\boldsymbol{U})$ denote the mean of a time period that the device is *off*. Let $p(t) = \mathbb{P}[\text{device is } \textit{on}]$ be the probability that the device is *on* at time $t \in [0, \infty)$. Since the uptime (downtime) of a device is non-arithmetic, we have the following two relationships by the alternating renewal theorem [53]:

$$\begin{aligned} \lim_{t\to\infty} \mathbb{P}[\text{device is } \textit{on} \text{ at } t] &= \frac{\mu}{\mu+\nu} \\ \lim_{t\to\infty} \mathbb{P}[\text{device is } \textit{off} \text{ at } t] &= \frac{\nu}{\mu+\nu} \end{aligned} \tag{4}$$

Thus, as Δt is very small compared to the contract period duration, we approximate each device's state as a Bernoulli variable with the success probability $p = \frac{\mu}{\mu+\nu}$.

B. Computing Availability of SFCs

An SFC request r is defined as a tuple $r = \langle \alpha_r, V_r, t^r_{start}, t^r_{end}, b^r \rangle$, where α_r is the availability requirement, $V_r = \langle n^r_1, n^r_2, \ldots, n^r_{M_r} \rangle$ is the minimum number of VMs required of each VNFs for the service to be available. t^r_{start} and t^r_{end} specify the contract start and end time, respectively. b^r is the bandwidth requirement. Here we assume the egress and ingress traffic are the same, and since we assume a consolidated middlebox structure, the traffic between VNFs is ignored.

Given such an SFC request r, and assume that $\langle k^r_1, k^r_2, \ldots, k^r_{M_r} \rangle$ backups are allocated, the probability that the i-th VNF is available is:

$$\mathbb{P}[i\text{-th VNF is available}] = \mathbb{P}_{n^r_i + k^r_i}[X^r_i \geq n^r_i] \tag{5}$$

where X^r_i is a variable indicating the number of available VMs. Since the VMs' states are modeled as independent Bernoulli variables with different success probability, the sum of these variables, denoted as N^r, follows a Poisson binomial distribution [12]. Let $\xi^r_k = \mathbb{P}[N^r_i = k^r]$, $k^r = 0, 1, \ldots, n^r_i + k^r_i$ be the probability mass function (PMF) for the Poisson binomial random variable N^r_i, then the cumulative distribution function (CDF) of N^r_i, denoted by $F_{N^r_i}(k^r) = \mathbb{P}[N^r_i \leq k^r]$, $k^r = 0, 1, \ldots, n^r_i + k^r_i$, gives the probability of having at most

TABLE I
SYMBOLS AND THEIR DEFINITIONS

Symbol	Definition
t^r_{start}	The start time of the r-th request
t^r_{end}	The end time of the r-th request
n^r_i	The required number of VMs of the i-th VNF of the r-th request
α_r	The availability requirement of the r-th requirement
M_r	The total number of VNFs of the r-th request
$F_{N^r_{ij}}(x)$	The probability of having at most x VMs are available of all VMs mapped to the j-th physical machine
$\mathcal{R}_m$	The set of all subsets of m VMs of all VMs

k^r successes out of a total of $n^r_i + k^r_i$, and can be written [50] as:

$$F_{N^r_i}(k^r) = \sum_{m=0}^{k_r} \xi^r_m = \sum_{m=0}^{k_r} \left\{ \sum_{\mathcal{A}\in\mathcal{R}_m} \prod_{j\in\mathcal{A}} p_j \prod_{j\in\mathcal{A}^c} (1-p_j) \right\} \tag{6}$$

where $\mathcal{R}_m$ is the set of all subsets of m integers that can be selected from $\{1, 2, \ldots, n^r_i + k^r_i\}$, and $\mathcal{A}^c$ is the complement of set $\mathcal{A}$. Note that when all p_j are identical, the above equation is just the CDF of a Binomial distribution. Therefore, we have

$$\mathbb{P}_{n^r_i+k^r_i}[X^r_i \geq n^r_i] = 1 - F_{N^r_i}(n^r_i - 1). \tag{7}$$

Thus, the probability that all the VNFs are available can be expressed as

$$\prod_{i=1}^{M_r} \mathbb{P}_{n^r_i+k^r_i}[X^r_i \geq n^r_i] = \prod_{i=1}^{M_r} \big(1 - F_{N^r_i}(n^r_i - 1)\big). \tag{8}$$

However, computing such a function is not easy, which requires one to enumerate all elements in $\mathcal{R}_m$, which requires to consider an exponential number of scenarios. It is very large even when $n^r_i + k^r_i$ is small. In general, we show the following theorem:

Theorem 1: Verifying if the availability of a given deployed SFC request with backups is above a given threshold is PP-complete.

Proof: Rewrite Eq. (6) as a boolean formula, where x_j and $\bar{x_j}$ represent if the j-th VM is available or not, while $\prod$ and $\sum$ are replaced by $\wedge$ and $\vee$, respectively. By the definition of language in PP, it is clear this problem is in PP. To show PP-completeness, we can reduce MAJSAT problem [30] to this problem. MAJSAT is a decision problem where one needs to decide if half of all assignments can satisfy a given Boolean formula, and it is PP-complete. 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 the instance of this problem 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 this problem. To verify the correctness,

$$\begin{aligned} \phi \in MAJSAT &\iff \Pr[\phi(x)] > \frac{1}{2} \text{ with } x \in \{0,1\}^n \\ &\iff \text{the probability that } \phi \text{ can be satisfied} \\ &\quad > \quad \text{the given threshold } \frac{1}{2} \\ &\iff (\phi, \frac{1}{2}) \in \text{this problem,} \end{aligned}$$

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

To efficiently compute the availability of an SFC if only one physical machine is used, we adopt [25]. Interested readers can refer to the paper for more details. When multiple physical machines are used, the availability is the probability that at least one of the physical machines as well as all the VNFs deployed on the physical machine is available.

C. Problem Formulation

In this subsection, we formulate the request mapping and backup provisioning problem as an integer program. For ease of reading, Table I provides a list of symbols defined in previous sections.

As mentioned, the data center topology can be described as an abstract tree with N_s physical machines (i.e., nodes on the second layer from the bottom), and each physical machine $j \in [1, N_s]$ can contain up to c_j VMs. We are also given R SFC requests.

To formulate the problem, two binary variables are defined. $x^r_j = 1$ if the r-th request uses VMs of the j-th physical machine; otherwise, $x^r_j = 0$. Likewise, y^{rt}_{jk} equals 1 if the r-th request uses the k-th VM of the j-th physical machines at time slot t, where $t^r_{start} \leq t \leq t^r_{end}$ and $1 \leq k \leq c_j$; otherwise $y^{rt}_{jk} = 0$. Therefore, we can have the following constraints:

$$\sum_r y^{rt}_{jk} \leq 1 \tag{9}$$

which means that at any time slot, at most one SFC request can use the k-th VM of the j-th physical machine. It also ensures that the maximum number of VMs of any physical machine that are used at any time slot is less than the capacity of the physical machine. Thus

$$x^r_j \left(\sum_{i=1}^{M_r} n^r_i \right) \leq \sum_k y^{rt}_{jk} \leq 2 \times x^r_j \left(\sum_{i=1}^{M_r} n^r_i \right) \tag{10}$$

which guarantees that if r-th request uses VMs of the j-th physical machines (i.e., $x_j^r = 1$), at least $\sum_{i=1}^{M_r} n_i^r$ VMs are allocated at this physical machine for this request (Section II-B); otherwise, it should be zero, bounded by the left and right hand sides of the equation. The right hand side of the constraint sets the upper bound on the number of the backups to be $\sum_{i=1}^{M_r} n_i^r$, and one can change the constant given different settings.

It is also important to have bandwidth constraint to fulfill the service rate of an SFC. Given there are multiple paths between a physical machine to a core switch, when mapping a request, we also need to reserve bandwidth resource on the selected paths between the physical machine that implements the request and any of the core switches. Accordingly, $G(\cdot)$ should only include those selected paths.

Last but not least, the availability of an SFC request should always be greater than or equal to its requirement during the contract period duration:

$$1 - \prod_j \left(1 - \left(\prod_{i=1}^{M_r} \left(1 - F_{N_{ij}^r}(n_i^r - 1)\right) \times G(j)\right) x_j\right) \geq \alpha_r \tag{11}$$

where $1 - F_{N_{ij}^r}(n_i^r - 1) = \mathbb{P}_{n_i^r + k_i^r}[X_i^r \geq n_i^r]$ and the r-th request is mapped to the j-th server. $F_{N_{ij}^r}(\cdot)$ is defined as in Eq. (6). A VM $e \in \mathcal{R}_m$ if and only if $y_{je}^{rt} = 1$. To show this in the formulation, we can rewrite the availability p_k of a VM k as $p_k y_{jk}^{rt}$.

The objective of the problem is to minimize a weighted-sum of the number of VMs and physical machines that are needed:

$$\rho \sum_r \sum_j \sum_k y_{jk}^{rt} + (1 - \rho) \sum_j \mathbb{1}\left(\sum_r x_j^r \geq 1\right) \tag{12}$$

where $0 \leq \rho \leq 1$ is of one's choice and $\mathbb{1}(\cdot)$ is the indicator function. A higher value of ρ puts more emphasis on the number of VMs, e.g., when one tries to decide how much to charge users as with the current VM pricing models [1]; while a lower one puts more emphasis on the number of physical machines that need to be used, e.g., when energy consumption is more of a concern as we try to pack VMs into fewer physical machines and put the rest into hibernation [34].

D. Algorithm Design and Analysis

In this subsection, we propose an online algorithm to provision availability while mapping requests. Based on our complexity analysis above, we know that finding the optimal solution is challenging. Hence, to address the challenges, we decompose the problem into two phases: request mapping and backup provisioning. In request mapping, we need to select the physical machines to hold the request and reserve requested bandwidth from the machines to the core switches. In backup provisioning, we only consider the availability constraint in order to add backup VMs.

First recall the availability of a request when there is only one one used physical machine

$$\prod_{i=1}^{M_r} \left(1 - F_{N_i^r}(n_i^r - 1)\right) \tag{13}$$

As the availability requirement of an SFC request is usually very close to 1, $F_{N_{ij}^r}(n_i^r - 1)$ should be close to 0. Therefore, we can rewrite Eq. (13) as

$$\begin{aligned}
&\prod_i \left(1 - F_{N_{ij}^r}(n_i^r - 1)\right) \\
&= 1 - \sum_i \left(1 - F_{N_{ij}^r}(n_i^r - 1)\right) \\
&\quad + \sum_{w<v} \left(1 - F_{N_{wj}^r}(n_w^r - 1)\right)\left(1 - F_{N_{vj}^r}(n_v^r - 1)\right) \\
&\quad - \sum_{w<v<u} \left(1 - F_{N_{wj}^r}(n_w^r - 1)\right)\left(1 - F_{N_{vj}^r}(n_v^r - 1)\right) \\
&\quad \times \left(1 - F_{N_{uj}^r}(n_u^r - 1)\right) + \ldots + (-1)^{M_r} \prod_i F_{N_{ij}^r}(n_i^r - 1) \\
&\doteq 1 - \sum_i F_{N_{ij}^r}(n_i^r - 1)
\end{aligned} \tag{14}$$

Now, recall the definition of a supermodular function and a submodular function:

*Definition 1: A function $f : X \to R$ is **supermodular** if $f(x \uparrow y) + f(x \downarrow y) \geq f(x) + f(y), \forall x, y \in X$, where $x \uparrow y$ denotes the component-wise maximum and $x \downarrow y$ the component-wise minimum of x and y.*

*Definition 2: If $-f$ is supermodular, f is **submodular**.*

Therefore, we can have the following theorem:

Theorem 2: Fixing k, $F_N(k)$ is a decreasing supermodular function in the context of NFV and the decrement is proportional to the availability of the element added to $\mathcal{R}_m$.

Proof: We first prove the second half of the theorem and use the conclusion to prove the first half.

Given are two sets $\mathcal{R}_m = \{1, 2, \ldots, n\}$ and $\mathcal{R}'_m = \mathcal{R}_m \bigcup \{n+1\}$. Let $F_n(k)$ and $F_{n+1}(k)$ denote the probability of having at most k out of a total of n and $n+1$, respectively. p_i denotes the availability of the i-th component. Then

$$\begin{aligned}
F_{n+1}(k) &= \sum_{m=0}^{k} \left\{ \sum_{\mathcal{A} \in \mathcal{R}'_m} \prod_{j \in \mathcal{A}} p_j \prod_{j \in \mathcal{A}^c} (1 - p_j) \right\} \\
&= (1 - p_{n+1}) \sum_{m=0}^{k} \left\{ \sum_{\mathcal{A} \in \mathcal{R}_m} \prod_{j \in \mathcal{A}} p_j \prod_{j \in \mathcal{A}^c} (1 - p_j) \right\} \\
&\quad + p_{n+1} \sum_{m=0}^{k-1} \left\{ \sum_{\mathcal{A} \in \mathcal{R}_m} \prod_{j \in \mathcal{A}} p_j \prod_{j \in \mathcal{A}^c} (1 - p_j) \right\}
\end{aligned} \tag{15}$$

Therefore, we have

$$\begin{aligned}
&F_{n+1}(k) - F_n(k) \\
&= p_{n+1} \\
&\quad \times \left\{ \sum_{m=0}^{k-1} \left\{ \sum_{\mathcal{A} \in \mathcal{R}_m} \prod_{j \in \mathcal{A}} p_j \prod_{j \in \mathcal{A}^c} (1 - p_j) \right\} - F_n(k) \right\} \\
&= -p_{n+1} \times \sum_{\mathcal{A} \in \mathcal{R}_k} \prod_{j \in \mathcal{A}} p_j \prod_{j \in \mathcal{A}^c} (1 - p_j) = -p_{n+1} \xi_k
\end{aligned} \tag{16}$$

where ξ_k is the probability of having k successes out of n. As $\xi_k > 0$, we can see that fixing k, $F_n(k)$ is a decreasing function and the decrement is proportional to p_{n+1}.

Next, we show a sufficient condition for $F_{n+1}(k) - F_n(k) < F_{n+2}(k) - F_{n+1}(k)$. To satisfy the inequality, we need

$$p_n \xi_k > p_{n+1} \xi'_k \tag{17}$$

where ξ'_k is the probability of having k successes out of $n+1$. By expanding both ξ_k and ξ'_k as summations, we can find that $2 \times \zeta(\xi_k) = \zeta(\xi'_k)$, where $\zeta(x)$ denotes the number of terms in the summation form of x. To make them equal, rewrite each term φ_k in ξ_k as $\frac{1}{2} \times \varphi_k + \frac{1}{2} \times \varphi_k$. By subtracting the RHS from the LHS of Eq. (17), we can have

$$p_n \xi_k - p_{n+1} \xi'_k = \sum \prod_{i \in \mathcal{B}} p_i \prod_{i \in \mathcal{C}} (1 - p_i) \left(\frac{1}{2} \prod_{i \in \mathcal{D}} p_i \prod_{i \in \mathcal{E}} (1 - p_i) - \prod_{i \in \mathcal{G}} p_i \prod_{i \in \mathcal{H}} (1 - p_i) \right) \tag{18}$$

where $\mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E} \subseteq \mathcal{R}_m$ and $\mathcal{G}, \mathcal{H} \subseteq \mathcal{R}'_m$, and furthermore, $|E| < |H|$, $|D| \geq |G|$ and $|D| + |E| + 1 = |G| + |H|$. As in practice [3] (as well as our data from production data centers) the availability of a VM is quite high (at least greater than 0.9), Eq. (18) is greater than 0. Following a similar method, one can prove that $F_N(k)$ is a supermodular function. □

If all physical machines for a request are selected, then we can rewrite Constraint (11) as

$$1 - \alpha_r \geq \prod_j \left(1 - \left(\prod_{i=1}^{M_r} \left(1 - F_{N^r_{ij}}(n^r_i - 1) \right) \times G(j) \right) \right) \tag{19}$$

and take $\log$ on both sides,

$$c_r \geq \sum_j \log \left(1 - \left(\prod_{i=1}^{M_r} \left(1 - F_{N^r_{ij}}(n^r_i - 1) \right) \times G(j) \right) \right) \tag{20}$$

where $c_r = \log(1 - \alpha_r)$. Since $\prod_{i=1}^{M_r} \left(1 - F_{N^r_{ij}}(n^r_i - 1) \right) \times G(j) \leq 1$, the RHS of Eq. (20) can be approximated by Taylor series, and the availability constraint can be written as:

$$\sum_j G(j) \left(1 - \sum_i F_{N^r_{ij}}(n^r_i - 1) \right) \geq c'_r \tag{21}$$

where $c'_r = -c_r$, as $\log(1 - x) = -\sum_{n=1}^{\infty} \frac{x^n}{n} < -x$ when $x \approx 0$. Note that regardless of this relaxation, the solution is still valid and feasible as $\prod_{i=1}^{M_r} \left(1 - F_{N^r_{ij}}(n^r_i - 1) \right) \times G(j) > 0$.

Since the sum of supermodular functions is still supermodular, we know that Eq. (14) and the LHS of Eq. (21) is submodular. Therefore, we can have **Algorithm 3**.

The first step (line 1 to 3) is to pre-process the path availability for all N_s physical machines and sort all VMs in each physical machines by their availability defined in Section IV-A in descending order. Clearly, there is a tradeoff: selecting all paths between a physical machine and all core switches would increase the path availability of the machine, but it also means that we need to reserve bandwidth for the SFC on all paths, which would waste a lot of resources. To this end, we set a tunable parameter K as the number of paths between the physical machines and all core switches that the algorithm considers. Besides, setting a small K helps to further prune the BFS search space and reduce the running time (Section III-B). In other words, a physical machine may have multiple path availabilities, each corresponding to a set of K paths to the core switches, and a physical machine with all combinations of the K paths will be in the mapping candidates list `candidate`.

When a request comes, we first map it to the physical machines by selecting the machines with higher path availability and more available capacity, and reserve bandwidth on the corresponding K paths. Note that we only select the least number (M) of distinct physical machines, estimated as in Section III-E, from `candidate`. To map the request on these physical machines, $\sum_{i=1}^{M_r} n^r_i$ VMs with the highest availabilities are selected at each of the physical machines.

When provisioning backups (line 13 to 15), we try to maximize the availability that can be achieved. As a matter of fact, there exist various approximation algorithms for the maximization of such a monotone submodular function (the LHS of Eq. (21) as proved). The idea we use in **Algorithm 3** is to incrementally build a solution starting with no backups, and in each iteration the algorithm adds one backup that most improves the current availability, until the availability requirement is met. Each iteration takes $\mathcal{O}(M)$ time, as have been proved that the decrement of function $F_N(k)$ with respective to N is proportional to the availability of the newly added backup, and the algorithm only needs to search the VM with the highest availability on each of the M physical

Algorithm 3 GAP: Greedy Availability Provisioning in NFV

1: **for** each physical machine $j \in N_s$ **do**
2: Given the number of paths between the physical machines and all core switches K, compute all possible path availabilities
3: Sort VMs on physical machines by their availability (Section IV-A) in a descending order
4: **for** each request $r \in R$ **do**
5: `candidate` = N_s
6: **while** true **do**
7: Sort `candidate` by their available capacity and path availability in descending order and select the first M distinct physical machines as `C`
8: **for** physical machine $i \in$ `C` **do**
9: Map $\sum_{i=1}^{M_r} n^r_i$ VMs on each of the physical machine and reserve bandwidth on corresponding paths
10: **while** Eq. (21) is not met **do**
11: select one VM s.t. the LHS of Eq. (21) is maximized
12: **if** request r is not successfully mapped **then**
13: Reject the request

machines. Since we use such a greedy method, we have the following theorem [37]:

Theorem 3: The backup scheme $\mathcal{B}$ computed by our algorithm can achieve an availability that is at least $1-\frac{1}{e}$ times of the one with the optimal backups OPT given $|OPT| = |\mathcal{B}|$.

If there is not enough capacity (i.e., VMs) at a physical machine, we remove the machine with all possible path availabilities from the mapping candidate list `candidate`; or if there is insufficient bandwidth, we remove the machine with that particular path availability from `candidate`. The above procedure is repeated until a request is successfully mapped or considered rejected when there is no physical machine or bandwidth available.

V. Evaluation

In this section, we evaluate the performance of our proposed algorithm in terms of 1) SFC request acceptance ratio, 2) backup resource consumed by requests and 3) running time. The distribution of device downtime is generated based on [18] and data collected from the Center for Computational Research (CCR), a large high performance computing facility at SUNY Buffalo [3], [14]. The statistics shown in this section are averaged outputs of the framework.

First we introduce two methods to see how our optimization performs.

1) **Baseline**: In the baseline method, backups are allocated following **Algorithm 2**.
2) **Greedy**: We modify the method in [16] to estimate the number of backups, where for an SFC request which requires n VNF and has the availability requirement of α, we keep increasing the number of backup for each VNF on all selected physical machines until each VNF can have an availability of $\sqrt[n]{\alpha}$.
3) **GAP**: Backups are estimated following the optimization in **Algorithm 3**.

Note that after estimating the number of backups needed using **Greedy** and **GAP**, more backups will be provisioned iteratively following the backup allocation method in **Algorithm 2** until meeting the availability requirement. In all, **Algorithm 2** is applied in all three methods to provision backups as it guarantees availability while **Baseline** starts with zero backup and the other two start with an estimated number of backups.

A. Running Time

One of the main benefits of an accurate estimation is to reduce the number of iterations that our framework needs to run, and therefore reduce the total running time. Note that the running time shown in this section includes both the estimation time used by our algorithms and the iterations to add more backups to guarantee availability. With the same example in Section III-D and $K = 2$, the total running time can be reduced to less than 5 hours, on 40 non-dedicated cores, which achieves a 87.5% improvement. Fig. 6 and Fig. 7 show the absolute running time with an SFC request which requires the same number of VNFs and VMs but different contract period durations and availability requirements, respectively. First, let's

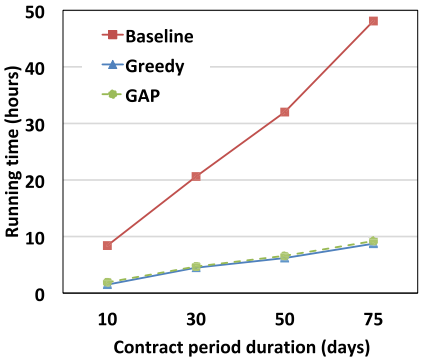


Fig. 6. Running time vs different contract period durations.

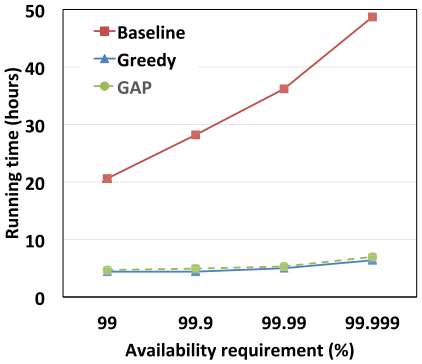


Fig. 7. Running time vs different availability requirements.

compare the performance of **Baseline** and **GAP**. It is clear that as the contract period duration increases, the running time of our framework increases almost proportionally, which is expected as the number of samples is proportional to the duration of the contract period (See Section III-B). However, if our algorithm is first applied to estimate the number of backups, the growth rate is much smaller. Similar observations can be made when the availability requirement changes. Specifically, the running time with optimization keeps almost constant as the availability requirement increases (from 99% to 99.99%). When the availability is 99.99%, the relative improvement can be near 90%. The relative improvement is even higher when the availability requirement is further increased (i.e., 99.99%). The results imply that our optimization framework can help to estimate the number of backups needed, and we can save more time with higher availability requirements and longer contract period durations. Next, we compare the performance of **Greedy** and **GAP**. **Greedy** requires slightly less time than **GAP** as its estimation returns with a higher number of VMs (will be shown in the next section) and therefore a request is mapped with more backups.

We also evaluate our algorithm with SFC requests with different availability requirements, contract period durations, number of VNFs and number of VMs for each VNF. In total there are about 30 different cases with various combinations. The availability requirement of a request is selected among {99%, 99.9%, 99.99%, 99.999%, 99.9999%} and the range of contract period duration is between 10 to 100 days.

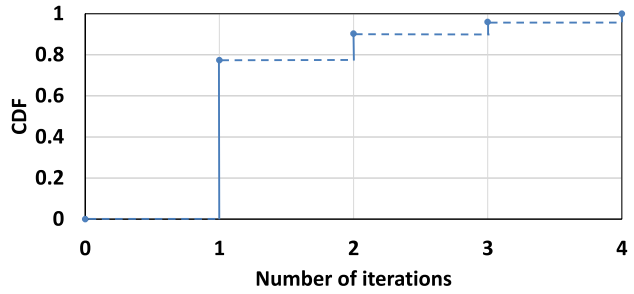


Fig. 8. CDF of the number of iterations needed.

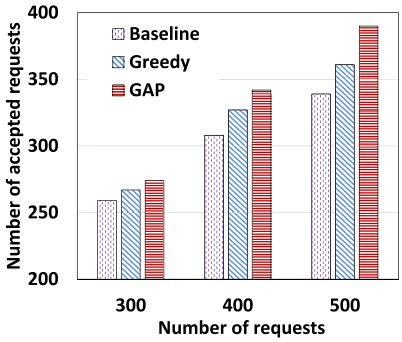


Fig. 9. Number of accepted requests with the Fat-tree topology.

Fig. 8 shows the CDF of the number of extra iterations needed by our framework to provision availability after a request is mapped with **GAP**. For over three quarters of the time, only one iteration was required. As at least we need one iteration to verify if the returned result can achieve the availability requirement, this result shows that our optimization framework can return a solution within a single iteration with enough number of backups to guarantee availability in most cases. Multiple iterations are only needed in rare cases.

B. Resource Efficiency

Next, we run simulations to understand how the proposed algorithm can save resources. In our simulations, we use the 8-ary Fat-tree and the BCube(2, 5) topology so that they have a similar number of physical machines, and assume each physical machine can support up to 1024 VMs [2]. For each SFC request, the availability requirement is randomly selected among {99%, 99.9%, 99.99%, 99.999%, 99.9999%}, and the contract period duration is exponentially distributed with an average of 10 days. Each request consists of two to six VNFs, each of which demands a certain number of VMs that is uniformly distributed between 10 and 40 [16].

Fig. 9 and Fig. 10 shows the number of accepted requests with respect to different number of requests with two different topologies, respectively. In these simulations, the number of paths between the physical machines and all core switches K is set to 2 for both topologies. Clearly, with our backup selection method (i.e., **GAP**), the number of requests that can be accepted increases as the number of the requests increases. Furthermore, as the request number increases, we have a larger performance gain compared to **Baseline** and **Greedy**, since when the request number is small, VM resources are sufficient;

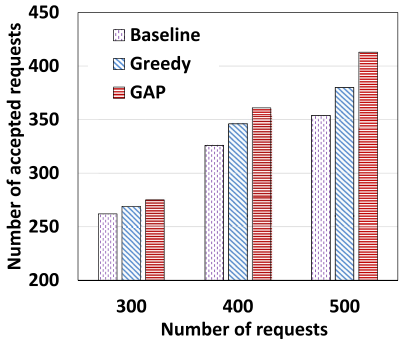


Fig. 10. Number of accepted requests with the BCube topology.

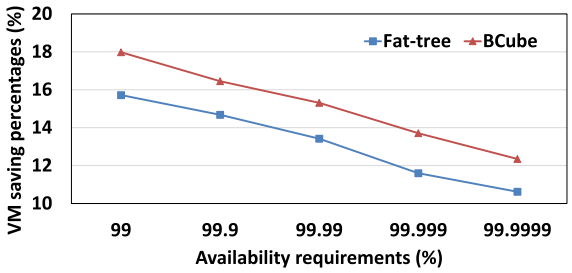


Fig. 11. Backup VM saving percentages.

TABLE II
NUMBER OF ACCEPTED REQUESTS WITH DIFFERENT K

K	1	2	3	4
Number of accepted requests	285	327	328	302

while when there are more requests, the VM resources saved by **GAP** can be used to accommodate more requests. The reason is that with our algorithm, fewer backup VMs are required to reach a certain level of availability, as shown in Fig. 11, where the VM saving percentage is defined as $\frac{\text{backup \# w/ greedy - backup \# w/ GAP}}{\text{backup \# w/ greedy}}$ given different availability requirements, and the request number is 400. Compared with **Greedy**, **GAP** can save more resources while not increasing the running time. In addition, as depicted in these figures, more requests can be accepted with BCube compared to Fat-tree topology even though K is the same. This is because BCube is more robust to failures [20].

To understand how K affects the performance, we run simulations in the same Fat-tree topology when there are 400 requests, and the number of accepted requests with different K value are shown in Table II. As mentioned, there is a trade-off between path availability of a physical machine and bandwidth consumption when adjusting K. From the table, we can see that when K is too small (i.e., $K = 1$) or too large (i.e., $K = 4$), the number of requests that can be accepted is smaller. Even though $K = 3$ gives up a slightly better result, the framework also needs to spend a longer time (about 30% more) to evaluate availability of a mapped request.

VI. RELATED WORK

Recently there are a number of works on network function placement with various objectives following the work of resource allocation in data centers [33], [36], [44], [49], such

as energy efficiency and scalability. Bagaa *et al.* [7] not only consider the placement of VNFs but also derive the optimal resources needed to serve a given mobile traffic. Benkacem *et al.* [8] study the VNF placement problem in a CDN environment and considers both cost efficiency and QoE. Utilizing bargaining game theory, the authors show an optimal tradeoff between them. Bouet *et al.* [9] solve the virtual DPI placement problem to minimize the total cost subject to delay constraint. Cohen *et al.* [11] study a general NFV placement problem and provides an approximation algorithm. Laghrissi *et al.* [31] present VNF placement algorithms to address the problem of non-uniform service demands and the irregular nature of network topologies by transforming physical domain signaling messages to canonical domain. Laghrissi *et al.* [32] study mobile service consumption behavior and presents a enhanced predictive VNF placement to consider both QoS and cost efficiency. Martini *et al.* [35] study the problem of composing and computing VNFs to select nodes along the path with the objective of minimizing end-to-end latency. Taleb *et al.* [47] optimize VNF placement while considering operators' cost as well as service usage behavioral patterns in a mobile environment.

However only few of them have considered the availability/reliability issues [16], [17], [40], let alone how to guarantee availability, which is considered as one of the most essential topics in NFV [22]. Ding *et al.* [13] propose a new redundancy model by taking the global information of the VNF forwarding graph into consideration. Fan *et al.* [15], [16] proposed algorithms to provision availability with on- and off-site backups, but they assumed that only VMs would experience failures. Herker *et al.* [24] proposed resilient embedding algorithms in data centers for multiple backup strategies. Kang *et al.* [27] studied the tradeoff between reliability and computational due to replication of VNFs. Sun *et al.* [46] proposed algorithms to maximize the end-to-end availability while taking failures of both nodes and links into consideration. Kanizo *et al.* [28] proposed to optimize the backup scheme to maximize survivability while respecting the resource constraints. Qu *et al.* [40] jointly optimize VNF placement and traffic routing in data center networks to maximize the reliability of network services and minimize end-to-end service delay. Following this work, the authors further exploited the possibilities of backup resource sharing when provisioning SFC requests recently in [41]. However, none of these work considers the heterogeneous processes of devices' failure and repair to the best of our knowledge. In this paper, on the contrary, we propose a general framework to provision availability of NFV in data centers of arbitrary number of devices and improve the resource usage efficiency.

VII. Discussion

Before concluding, we briefly discuss the usage issues of our framework in practice.

First, even though our optimization is able to reduce the total running time of the framework significantly, this procedure still requires a few hours to finish (Section V-A). Here, we list two potential solutions as we have seen that our optimization can achieve a reasonable estimation.

1) Based on our evaluations, since *for requests with shorter contract time periods or less strict availability requirements*, our algorithm can return enough backups, we can just use the optimization to estimate backups needed for these requests; while for the rest requests, even though the estimated backups are not enough to provision availability, we know from Fig. 8 that we can achieve that by adding a few more backup VMs that would have been added during the iterations. Therefore, *for requests with long contract time periods or strict availability requirements* we can map them with a few more backup VMs in addition to the estimated ones, as a possible upper limit on the VMs needed. Then we can accurately evaluate its availability using our framework after the request is deployed. If fewer backups are needed, we can retract the over-provisioned VMs.

2) We can further improve the estimation efficiency in III-B. Instead going over all possible states when estimating the downtime distribution, we can utilize importance sampling while having a very high accuracy.

Second, we made several assumptions that are simplifications of reality when modeling the problem, which we intend to further explore as a future work. For example, we currently do not consider Byzantine failures of devices, which means devices, upon certain failures, can continue to work, but at a significantly reduced capacity. To incorporate this failure model, the definition of availability should also be modified that takes the processing rate of traffic into consideration. Besides, correlated failures such as failures due to power outages or configuration errors can have significant effects on service availability, which are not considered in this paper. One possible solution to address it is to identify/audit the independence of resources when allocating redundancy, and it requires a more detailed model of the system. Last but not least, for stateful VNFs, failovers from primaries to backups requires transferring of states, which makes the switching time not zero. We are currently working on a Generalized Stochastic Petri Net (GSPN) model to incorporate such an effect. More precisely, the switching from primaries to backups can be modeled as a timed transition, which means only after certain time (i.e., after the switching finishes), a backup can start and process traffic.

VIII. Conclusion

NFV explores the virtualization technologies to offer network-as-a-service through chained VNFs. Since telecom networks must be always on, it is critical to guarantee service availability with the minimum amount of resources. In this paper, we presented our approach to provisioning availability in NFV in data center networks where the failure and repair process of the devices follow heterogeneous distributions. To efficiently and accurately solve tihe problem, requests are first mapped with the estimated number of backups through solving an optimization problem with the objective of minimizing the resource consumption and achieving heterogeneous availability requirements, while assuming the failure probability of each device is a constant. Then the downtime distribution of a mapped request is accurately calculated and more backups would be provisioned until the

availability requirement is met. Through evaluation with real world data, we demonstrated the effectiveness and efficiency of our solutions.

REFERENCES

[1] *Amazon EC2 Pricing*. Accessed: Apr. 20, 2016. [Online]. Available: https://aws.amazon.com/ec2/pricing

[2] *FusionSphere Cloud OS*. Accessed: Mar. 11, 2016. [Online]. Available: http://e.huawei.com/us/products/cloud-computing-dc/cloud-computing/fusionsphere/fusionsphere

[3] (2008). *Reducing Data Center Cost With an Air Economizer*. [Online]. Available: http://www.intel.com/it/pdf/Reducing_Data_Center_Cost_with_an_Air_Economizer.pdf

[4] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM*, 2008, pp. 63–74.

[5] J. W. Anderson, R. Braud, R. Kapoor, G. Porter, and A. Vahdat, "xOMB: Extensible open middleboxes with commodity servers," in *Proc. ACM/IEEE ANCS*, 2012, pp. 49–60.

[6] T. Aven and U. Jensen, *Stochastic Models in Reliability*. New York, NY, USA: Springer, 1999.

[7] M. Bagaa, T. Taleb, A. Laghrissi, A. Ksentini, and H. Flinck, "Coalitional game for the creation of efficient virtual core network slices in 5G mobile systems," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 469–484, Mar. 2018.

[8] T. Taleb, I. Benkacem, M. Bagaa, and H. Flinck, "Optimal VNFs placement in CDN slicing over multi-cloud environment," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 616–627, Mar. 2018.

[9] M. Bouet, J. Leguay, T. Combe, and V. Conan, "Cost-based placement of vDPI functions in NFV infrastructures," *Int. J. Netw. Manage.*, vol. 25, no. 6, pp. 490–506, 2015.

[10] I. M. Chakravarti, J. Roy, and R. G. Laha, *Handbook of Methods of Applied Statistics*. Hoboken, NJ, USA: Wiley, 1967.

[11] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Proc. IEEE INFOCOM*, Apr./May 2015, pp. 1346–1354.

[12] C. Daskalakis, I. Diakonikolas, and R. A. Servedio, "Learning Poisson binomial distributions," *Algorithmica*, vol. 72, no. 1, pp. 316–357, 2015.

[13] W. Ding, H. Yu, and S. Luo, "Enhancing the reliability of services in NFV with the cost-efficient redundancy scheme," in *Proc. IEEE ICC*, May 2017, pp. 1–6.

[14] A. Y. Du, S. Das, Z. Yang, C. Qiao, and R. Ramesh, "Predicting transient downtime in virtual server systems: An efficient sample path randomization approach," *IEEE Trans. Comput.*, vol. 64, no. 12, pp. 3541–3554, Dec. 2015.

[15] J. Fan, C. Guan, Y. Zhao, and C. Qiao, "Availability-aware mapping of service function chains," in *Proc. IEEE INFOCOM*, May 2017, pp. 1–9.

[16] J. Fan, M. Jiang, and C. Qiao, "Carrier-grade availability-aware mapping of service function chains with on-site backups," in *Proc. IEEE IWQoS*, Jun. 2017, pp. 1–10.

[17] J. Fan, Z. Ye, C. Guan, X. Gao, K. Ren, and C. Qiao, "GREP: Guaranteeing reliability with enhanced protection in NFV," in *Proc. ACM SIGCOMM Workshop Hot Topics Middleboxes Netw. Function Virtualization*, 2015, pp. 13–18.

[18] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," in *Proc. ACM SIGCOMM*, 2011, pp. 350–361.

[19] A. Greenhalgh, F. Huici, M. Hoerdt, P. Papadimitriou, M. Handley, and L. Mathy, "Flow processing and the rise of commodity network hardware," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 2, pp. 20–26, 2009.

[20] C. Guo *et al.*, "BCube: A high performance, server-centric network architecture for modular data centers," in *Proc. ACM SIGCOMM*, 2009, pp. 63–74.

[21] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: A scalable and fault-tolerant network structure for data centers," in *Proc. ACM SIGCOMM*, 2008, pp. 75–86.

[22] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 90–97, Feb. 2015.

[23] S. Han, K. Jang, A. Panda, S. Palkar, D. Han, and S. Ratnasamy, "SoftNIC: A software NIC to augment hardware," Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2015-155, 2015.

[24] S. Herker, X. An, W. Kiess, S. Beker, and A. Kirstaedter, "Data-center architecture impacts on virtualized network functions service chain embedding with high availability requirements," in *Proc. IEEE Globecom Workshops*, Dec. 2015, pp. 1–7.

[25] Y. Hong, "On computing the distribution function for the Poisson binomial distribution," *Comput. Statist. Data Anal.*, vol. 59, pp. 41–51, Mar. 2013.

[26] J. Hwang, K. K. Ramakrishnan, and T. Wood, "NetVM: High performance and flexible networking using virtualization on commodity platforms," *IEEE Trans. Netw. Service Manage.*, vol. 12, no. 1, pp. 34–47, Mar. 2015.

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

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

[29] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, 1951.

[30] J. Kwisthout, "Most probable explanations in Bayesian networks: Complexity and tractability," *Int. J. Approx. Reasoning*, vol. 52, no. 9, pp. 1452–1469, 2011.

[31] A. Laghrissi, T. Taleb, and M. Bagaa, "Conformal mapping for optimal network slice planning based on canonical domains," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 519–528, Mar. 2018.

[32] A. Laghrissi, T. Taleb, M. Bagaa, and H. Flinck, "Towards edge slicing: VNF placement algorithms for a dynamic & realistic edge cloud environment," in *Proc. IEEE GLOBECOM*, Dec. 2017, pp. 1–6.

[33] X. Li, J. Wu, S. Tang, and S. Lu, "Let's stay together: Towards traffic aware virtual machine placement in data centers," in *Proc. IEEE INFOCOM*, Apr./May 2014, pp. 1842–1850.

[34] P. Mahadevan, S. Banerjee, P. Sharma, A. Shah, and P. Ranganathan, "On energy efficiency for enterprise and data center networks," *IEEE Commun. Mag.*, vol. 49, no. 8, pp. 94–100, Aug. 2011.

[35] B. Martini, F. Paganelli, P. Cappanera, S. Turchi, and P. Castoldi, "Latency-aware composition of virtual functions in 5G," in *Proc. IEEE NetSoft*, Apr. 2015, pp. 1–6.

[36] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.

[37] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions—I," *Math. Program.*, vol. 14, no. 1, pp. 265–294, 1978.

[38] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker, "NetBricks: Taking the V out of NFV," in *Proc. OSDI*, 2016, pp. 203–216.

[39] E. Parzen, "On estimation of a probability density function and mode," *Ann. Math. Statist.*, vol. 33, no. 3, pp. 1065–1076, Sep. 1962.

[40] L. Qu, C. Assi, K. Shaban, and M. J. Khabbaz, "A reliability-aware network service chain provisioning with delay guarantees in NFV-enabled enterprise datacenter networks," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 3, pp. 554–568, Sep. 2017.

[41] L. Qu, M. Khabbaz, and C. Assi, "Reliability-aware service chaining in carrier-grade softwarized networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 558–573, Mar. 2018.

[42] B. Schlinker *et al.*, "Condor: Better topologies through declarative design," in *Proc. ACM SIGCOMM*, 2015, pp. 449–463.

[43] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proc. USENIX NSDI*, 2012, p. 24.

[44] V. Shrivastava, P. Zerfos, K.-W. Lee, H. Jamjoom, Y.-H. Liu, and S. Banerjee, "Application-aware virtual machine migration in data centers," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 66–70.

[45] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, "NFP: Enabling network function parallelism in NFV," in *Proc. ACM SIGCOMM*, 2017, pp. 43–56.

[46] J. Sun *et al.*, "A reliability-aware approach for resource efficient virtual network function deployment," *IEEE Access*, vol. 6, pp. 18238–18250, 2018.

[47] T. Taleb, M. Bagaa, and A. Ksentini, "User mobility-aware virtual network function placement for virtual 5G network infrastructure," in *Proc. IEEE ICC*, Jun. 2015, pp. 3879–3884.

[48] Z. Taylor and S. Ranganathan, *Designing High Availability Systems: DFSS and Classical Reliability Techniques With Practical Real Life Examples*. Hoboken, NJ, USA: Wiley, 2013.

[49] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 71–75.
[50] Y. H. Wang, "On the number of successes in independent trials," *Statist. Sinica*, vol. 3, no. 2, pp. 295–312, 1993.
[51] W. Yeow, C. Westphal, and U. C. Kozat, "Designing and embedding reliable virtual infrastructures," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 2, pp. 57–64, Apr. 2011.
[52] W. Yu, W. Rhee, S. Boyd, and J. M. Cioffi, "Iterative water-filling for Gaussian vector multiple-access channels," *IEEE Trans. Inf. Theory*, vol. 50, no. 1, pp. 145–152, Jan. 2004.
[53] M. Zhao, "Availability for repairable components and series systems," *IEEE Trans. Rel.*, vol. 43, no. 2, pp. 329–334, Jun. 1994.

Jingyuan Fan received the B.Eng. degree from Fudan University, China, in 2012, and the M.S. degree from the University of California, Los Angeles, CA, USA, in 2014. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, University at Buffalo, Buffalo, NY, USA. His research interests lie in the field of computer networks.

Meiling Jiang received the M.S. degree in applied mathematics from The State University of New York at Buffalo, Buffalo, NY, USA, where she is currently pursuing the Ph.D. degree with the Department of Management Science and Systems. Her research interest is in statistical modeling of availability in computing systems, operations and economics of cloud computing markets, and healthcare analytics.

Ori Rottenstreich received the B.Sc. degree (*summa cum laude*) in computer engineering and the Ph.D. degree from the Technion, Haifa, Israel, in 2008 and 2014, respectively. From 2015 to 2017, he was a Post-Doctoral Research Fellow with the Department of Computer Science, Princeton University, Princeton, NJ, USA. He is currently with the Faculty of Computer Science, Technion, where he is also with the Faculty of Electrical Engineering. His main research interest is computer networks.

Yangming Zhao received the B.S. degree in communication engineering and the Ph.D. degree in communication and information system from the University of Electronic Science and Technology of China in 2008 and 2015, respectively. He is currently a Research Scientist with SUNY Buffalo, Buffalo, NY, USA. His research interests include network optimization, data center networks, edge computing, and transportation systems.

Tong Guan received the B.S. degree in electrical engineering from the Huazhong University of Science and Technology in 2011 and the Ph.D. degree in computer science and engineering from the University at Buffalo, Buffalo, NY, USA, in 2018. His research interests are in the areas of networking, mobile networks, and social networks with emphasis on mathematical modeling and performance analysis.

Ram Ramesh is currently a Professor of management science and systems with SUNY Buffalo, Buffalo, NY, USA. His research has been funded by NSF, AFOSR, ARL, ARI, Google, Raytheon, Samsung, and Westinghouse among others. He has authored extensively in ISR, INFORMS JoC, the IEEE TKDE, and many others. His current research focuses on cloud infrastructure availability analytics. He serves as an Editor-in-Chief for the *Information Systems Frontiers* and an Area Editor for the *INFORMS Journal on Computing* for the area "Knowledge Management and Machine Learning."

Sanjukta Das received the Ph.D. degree in operations and information management from the University of Connecticut, Mansfield, CT, USA, in 2007. She is currently an Associate Professor and the Department Chair with the Department of Management Science and Systems, SUNY Buffalo, Buffalo, NY, USA. Her research has been funded by Google and NSF. She has authored extensively in journals such as the *INFORMS Journal on Computing* and the *Information Systems Research*. Her research interests include resource allocation and contract design in cloud computing. She serves as an Associate Editor for the *Information Systems Research*. She served as an Associate Editor for the *INFORMS Journal on Computing*, as a Coordinating Editor for the *Information Systems Frontiers*, and as a Guest Associate Editor for the *MIS Quarterly*.

Chunming Qiao (F'10) is currently a SUNY Distinguished Professor and also the Chair of the Computer Science and Engineering Department, University at Buffalo, Buffalo, NY, USA. He was elected as an IEEE fellow for his contributions to optical and wireless network architectures and protocols. His research has been funded by a dozen major IT and telecommunications companies, including Cisco and Google, and more than a dozen NSF grants. He has authored extensively with an h-index of over 69 (according to Google Scholar). He holds seven U.S. patents. His current focus is on connected and autonomous vehicles. Two of his papers have received the best paper awards from the IEEE and joint ACM/IEEE venues. He has been serving as a consultant for several IT and telecommunications companies since 2000.