Service Function Chain Deployment with Guaranteed Resilience

Yang Chen and Jie Wu
Center for Networked Computing, Temple University, USA
Email: {yang.chen, jiewu}@temple.edu

***Abstract*—Network Function Virtualization (NFV) transforms the implementation of network functions from expensive hardwares to software middleboxes. These software middleboxes, also called Virtual Network Functions (VNFs), are executed on virtualization platforms, which makes them more prone to error compared to dedicated hardwares. One effective way of ensuring VNF robustness is to provision redundancy in the form of deploying backup instances. Flows usually request to be processed by a service chain, consisting of multiple chained VNFs in some order. This paper considers both the resilient VNF deployment and the routing of flows. We deploy both active and backup VNF instances while guaranteeing the required VNF service resilience. Our objective is to minimize the total expected transmission delay of all flows when the probabilistic prior failure information is given. We first formulate our problem mathematically. We discuss the solution for the general network setting. Then, we simplify some setups and propose performance-guaranteed solutions in various scenarios with detailed approximation ratio analysis: for the case of a single flow without backup instances, we propose four optimal solutions corresponding to the settings of link transmission delay and server capacity; for the case of a single flow with backup instances, we extend the algorithms; for the case of two flows without backup instances, we propose efficient algorithms with approximation ratios inspired by the solutions for a single flow.**

***Index Terms*—Delay, deployment, resilience, VNFs.**

I. Introduction

In Software Defined Networks (SDNs), flows usually request to be processed by a service chain, an ordered set of Virtualized Network Functions (VNFs). Fig. 1 illustrates an example of a service chain, consisting of four VNFs: NAT, Firewall, IDS, and Monitor [1]. Network Function Virtualization (NFV) is the driving force behind the change in implementing network functions from expensive hardwares to software middleboxes, which are deployed at switch-connected servers [2]. With the probabilistic prior failure information of software middleboxes [3], we consider the resilient VNF deployment and routing of flows with the objective of minimizing the maximum expected transmission delay of all flows [4]. Beyond the challenges of transitioning to NFV, extra challenges lie in several aspects as follows: 1) With prior failure information, there is always a need to meet a certain level of service chain availability. Backup VNF instances are deployed in order to increase the availability of the provided service chain. 2) The setting of link transmission delay and server capacity can be homogeneous (uniform) or heterogeneous (non-uniform), which complicates our problem. 3) Multiple flows may compete for the same link bandwidth resource [5]. In addition, various VNF active and backup instances can share a server's capacity. 4) When an active instance of a network function fails, a flow needs to reroute to the backup instance of the function to get processed, which adds extra transmission delay.

This research was supported in part by NSF grants CNS 1824440, CNS 1828363, CNS 1757533, CNS 1629746, CNS-1651947, and CNS 1564128.

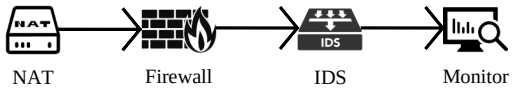


Fig. 1: A service chain example.

Considering all of these challenges leads to our problem: given the probabilistic prior failure information and the service chain availability requirement of flows, how we can efficiently deploy VNF active and backup instances on switch-connected servers and route flows in order to minimize the total expected flow transmission delay, when the service chain availability requirements of flows are satisfied [6]. Therefore, we handle our problem by proposing effective solutions under some simplified network settings, such as the topology and the required server resource for deploying a network function instance. We not only consider the deployment of network functions on specific topologies, but also study the service chain resilience issue. We guarantee a certain level of resilience while reserving resources for future use by selecting several network functions to have backup instances. In contrast to the existing resilient resource management schemes for NFV [7–9], we consider the joint design of resource management schemes accounting for both normal and failure scenarios.

We use a motivation example to illustrate our problem in Fig. 2. We are given a toy leaf-spine data-center topology [10], including spine and leaf switches and server racks. There is one flow f, whose source and destination are shown in the figure. It needs to be processed by a service chain, consisting of four middleboxes in sequence: NAT, Firewall, IDS, and Monitor. Their active instances are denoted as white boxes marked with their names. In order to increase the availability of the provided service chain, three of them have backup instances, denoted as left-slashed boxes. The red solid line shows the routing path of f when there is no failure. If the primary instance of Firewall fails, because of no Firewall

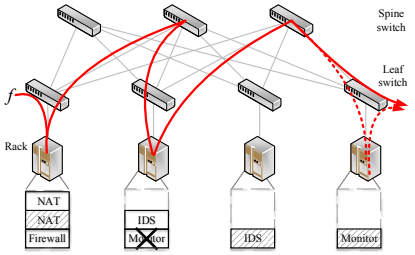


Fig. 2: A motivation example.

backup instance, the service chain becomes unavailable with one function not working. If the primary instance of Monitor fails, the flow will reroute to its backup instance before leaving the right-most leaf switch, shown in the red dashed line. This will add extra transmission delay because of a longer path.

In this paper, we study the service chain resilient deployment with backup instances. We aim at minimizing the total expected transmission delay among all the admitted flows. First, we formulate the problem mathematically in the form of Integer Programming and propose the corresponding solution for the general case. Then we simplify several settings, including specific network topologies, the settings of link transmission delay, and server capacity. We design several simple but efficient algorithms: For the case of a single flow without backup instances, we propose four optimal solutions corresponding to four combinations of link transmission delay and server capacity. For the case of a single flow with backup instances, we extend the algorithms. For the case of two flows without backup instances, we propose efficient algorithms with approximation ratios inspired by the solutions for a single flow.

II. Related Work

NFV frameworks have drawn a lot of attention, especially in the areas of VNF deployment and service chain resilience. The problems of efficient resource management for NFV, including VNF placement, resource allocation, and flow routing, have been extensively studied in the literature [11]. Several studies consider the placement of a minimum number of VNF instances to cover all of the flows. While the case of a single type of network function is considered in [12], the case of multiple types of network functions is addressed in [13]. The work of [14] considers the placement of middleboxes to keep the shortest path between communicating pairs below threshold, but this work does not consider multiple network functions.

In other related domains, similar problems have also been studied. For example, the work in [15] considers the placement of SDN-enabled routers to maximize the total processed traffic. They consider a total resource constraint but neglect the limited resource constraint. Similarly, in the work on link cloud computing [16], although the resource constraints are considered, their proposed solution is only for a special case, and the overall problem does not consider the multi-dimensional setting. To the best of our knowledge, the multi-dimensional setting has rarely been considered except in a limited number of studies. In [17], the authors consider multi-resource VNFs, with a focus on the analysis of the vertical scaling (i.e., scaling up/down of various resources) and horizontal scaling (i.e., varying number of VNFs instances). The work of [18] focuses only on request admission and routing. In [19], although the multi-resource setting is considered, the focus is on how to balance the traffic load across servers, taking into account different resource requirements by different network functions.

Most of the existing work on resilience design for NFV is focused on the standby deployment model [7–9]. This model requires at least one standby VNF instance for each primary VNF instance so as to ensure certain availability when failure occurs. Instead, [20] considers the hot-standby resilience design, where each standby instance is also active and is consistently synchronized with the primary instance. In contrast to backing up instances of the same type of network functions, [21] takes into account the inherent resource-sharing property of the virtualized resources and considers models where each backup server can be provisioned for multiple types of VNFs; the backup server can be up and running as a certain type of VNF when failure occurs at some instances of the corresponding network function. Taking a different path, [22] studies the problem of VNF recovery by dynamically reallocating the processing resource of VNF instances. Instead of having one primary instance and one backup instance, both instances are actively functioning and have their states synchronized with each other. When one instance fails, the other instance will dynamically adjust its processing resource to accommodate the traffic that is supposed to be processed by the faulty instance. Taking advantage of the cloud networks, [23] minimizes the backup cost with edge resource constraints when considering the availability requirements. It also studies both the static and the dynamic backup situation.

III. Network Model and Problem Formulation

A. Resilience

Here we introduce the definition of resilience and the setting of resilience in this paper. With the advent of Network Function Virtualization (NFV), network functions that traditionally run on proprietary dedicated hardware can now be realized using Virtual Network Functions (VNFs) that are hosted on off-the-shelf general-purpose commodity servers. While this new network paradigm offers great flexibility to network operators for efficient management of their networks, VNF instances based on software implementation running on general-purpose servers are typically more prone to error and more vulnerable to security threats compared to dedicated hardware devices [24, 25]. They can suffer from temporary unavailability due to mis-configuration or software and hardware malfunction. The VNF resilience can be achieved by an active-standby deployment, where each network function has a backup instance, which is activated in case of failure. Therefore, the NFV paradigm poses new challenges.

***Theorem** 1:* The active and backup instances for one VNF should be deployed in different servers in order to have a higher availability of the VNF.

Proof: We prove the theorem by probability analysis. We need there to be at least one instance between the active

Symbols	Definitions
$\mathcal{V}, \mathcal{E}$	the set of vertices and edges
γ	availability of server
p_ϕ, p'_ϕ	availability of middlebox ϕ and its backup
$f, \lambda_f, P_f, \Phi_f$	a flow, its traffic rate, path set, its required service chain
src_f, dst_f	the source and the destination of the flow
γ_f	service chain availability requirement of flow f
d_e, c_e	transmission delay and remaining capacity of link e
x_ϕ^v	indicator function of middlebox ϕ having a backup on v
$\alpha_\phi, \alpha'_\phi$	availability of VNF ϕ primary and backup instances

TABLE I: Symbols and Definitions.

instance and the backup one for each VNF ϕ that is available. If both the active instance and the backup instance are deployed in the same server, the availability of the VNF ϕ is equal to $a_1 = \gamma \cdot [1 - (1 - \alpha_\phi)^2]$. If they are distributed to different servers, the availability of the VNF ϕ is equal to $a_2 = 1 - (1 - \gamma \cdot \alpha_\phi)^2$. Then we calculate the difference between these two values:

$$a_2 - a_1 = 1 - (1 - \gamma \cdot \alpha_\phi)^2 - \gamma \cdot [1 - (1 - \alpha_\phi)^2]$$
$$= (2\gamma \cdot \alpha_\phi - (\gamma \cdot \alpha_\phi)^2) - (2\gamma \cdot \alpha_\phi - \gamma \cdot \alpha_\phi^2) = \gamma \cdot \alpha_\phi^2(1 - \gamma)$$

We know that the availability of a server γ is always less than 1, even though it is close to 1. Then we have $a_2 - a_1 \geq 0$, illustrating that the distributed deployment has a higher availability. As a result, the active and the backup instances need to be deployed in different servers. ■

B. Network model

When we have the probabilistic prior failure information, we consider the problems of VNF deployment, resource allocation, and flow routing with resilience guarantees against both hardware and software failures. In certain scenarios, such information can be learned from failure analytics using historical data. We first consider software failure only. That is, each VNF instance may fail independently due to software errors, but other VNF instances at the same VNF-node could still be functioning. We also discuss how to provide resilience guarantees against both software and hardware failures.

Let $\Phi_f = (\phi_f^1, \ldots, \phi_f^{|\Phi_f|})$ be the required SFC for flow f, where $|\Phi_f|$ is the chain length of the SFC for flow f. In order to ensure resilience against failure and improve the availability of the required SFC for a flow, we may allocate backup instances of the same type of network function in addition to its active instance. We assume that each deployed VNF instance (active or backup) is for an individual flow and not shared by other flows. By slightly abusing the notation, we also use Φ_f to denote the set of network functions in the SFC for flow f when there is no ambiguity. Let $\alpha^\phi \in [0, 1]$ be the availability probability of a VNF instance of network function ϕ, let $n_f^\phi(v)$ be the total number of VNF instances of network function $\phi \in \Phi_f$ (including active and backup) for flow f at node v, and let $\alpha_f^\phi \in [0, 1]$ be the availability probability of network function $\phi \in \Phi_f$ for flow f (taking into account both the active and backup instances). Hence, we have $\alpha_f^\phi = 1 - (1 - \alpha^\phi)^{\sum_{v \in \mathcal{V}} n_f^\phi(v)}$.

C. Problem formulation

Our objective is to minimize the maximum expected transmission delay of all flows based on the VNF failure probability distribution. The expected transmission delay of a flow f is denoted as D_f, which is the probability-weighted average transmission delay based on the probabilistic distribution of all failure cases when its provided service chain is still available. Note that when an instance fails, we still use the other active VNF instances in the service chain. Then D_f can be formulated as $D_f = \sum_i d_f^i \cdot \rho_f^i$ if the probability of a failure case i is ρ_f^i and the transmission delay of f under the case is d_f^i. (Case 0 is when none of the active instances fail.) We have $\sum_i \rho_f^i = 1$ and $d_f^i = \sum_{e \in p_f^i} d_e$ if p_f^i is the routing path of f under case i. p_f^i includes the same routing path of p_f^0, excluding the partial path from the failed active instance to its backup instance. This can make sure that the new routing path can still satisfy the constraints of the functioning and processing sequence.

For the service chain processing, we first need to make sure each flow is processed by all VNFs in its required chain of active instances along its routing path $p_f \in \mathcal{P}_f$, where $\mathcal{P}_f$ is the set of feasible paths of flow f. Let $\Phi(v)$ be the set of VNF instances (including both active and backup ones) deployed at node v, and let $\Phi(\mathcal{V}) = [\Phi(v)]_{v \in \mathcal{V}}$ be its vector form. We formulate this functioning constraint as:

$$\sum_{v \in p_f} \mathbf{1}_{\{\phi \in \Phi(v)\}} = 1, \forall \phi \in \Phi_f, \forall f \in \mathcal{F} \tag{1}$$

Secondly, each VNF $\phi \in \Phi_f$ needs to process f in the sequence of the chain along its routing path p_f. This sequence constraint can be formulated as: $\forall v_i,\ v_j \in p_f$ and $\forall \phi_m,\ \phi_n \in \Phi_f$, and we require that it satisfies the following constraint:

$$i \leq j, \quad \text{if } \phi_m \in \Phi(v_i),\ \phi_n \in \Phi(v_j) \text{ and } m < n \tag{2}$$

Then, to satisfy the SFC processing sequence, we always deploy the active instances according to the required order. Finally, each flow f has a service chain availability requirement, denoted as α_f. Its accumulated availability of the provided service chain with deployed active and backup instances, which is denoted as a_f, can be expressed as $a_f = \prod_{\phi \in \Phi_f} a_\phi$. We need to make sure that each flow $f \in \mathcal{F}$ meets its service chain availability requirement, which can be expressed as:

$$a_f \geq \alpha_f, \forall f \in \mathcal{F} \tag{3}$$

Note that when the availability of the provided service chain can be satisfied, we should deploy as few backup instances as possible in order to save resources for future use. This problem has been proved NP-hard in [26]. In this paper, we directly apply the proposed solutions in [26] to decide which functions should have backup instances.

Our problem has been proved NP-hard even when there is no resilience requirement [2]. In order to give some inspiration for the problem, we simplify some of the settings of our problem and propose efficient algorithms with approximation

Algorithm 1 Solution of case 1

In: Flows f, its required service chain ϕ_f and sets of vertices V, links E;

Out: The placement plan and its routing path;

1: Find the number of servers with enough total capacity to deploy all VNFs as l;
2: Select any path with exactly $l-1$ hops from source to destination;
3: Return the deployment plan and its routing path.

Algorithm 2 Solution of case 2

In: Flows f, its required service chain ϕ_f and sets of vertices V, links E;

Out: The placement plan and its routing path;

1: Deploy VNFs from the start of the service chain to use up the server capacity in flow source;
2: Deploy VNFs from the end of the rest of service chain to use up the server capacity in flow destination;
3: Sort the server capacity;
4: Select the servers with the largest capacity to deploy the residual VNFs in sequence until fully deployed;
5: Return the deployment plan and its routing path.

ratios in four special cases. In the next section, we focus on the leaf-spine network topology, which is one of the most popular topologies used in today's datacenters [10]. For the settings of link transmission delay and server capacity, we have two kinds: uniform and non-uniform (various). Based on their different settings, we have four combinations. As the required resource of each VNF is the same as a unit, we can check the feasibility of the service chain deployment by comparing the sum of residual capacity of each server in the network with the total number of VNFs in the required service chain. With the objective of minimizing the total expected transmission delay and the three constraints, our problem can be formulated as:

$$\underset{\Phi(\mathcal{V}),\boldsymbol{p}}{\text{minimize}} \quad \sum_{f\in\mathcal{F}} D_f \tag{4}$$

$$\text{subject to } (1),(2),(3) \tag{5}$$

IV. Solution for special cases

Our problem has been proved NP-hard even when there is no resilience requirement [2]. In order to give some inspiration for the problem, we simplify some settings of our problem and propose efficient algorithms with approximation ratios in four special cases. In this section, we focus on the leaf-spine network topology, which is one of the most popular topologies used in today's data centers [10]. For the settings of link transmission delay and server capacity, we have two kinds: uniform and non-uniform (various). Based on their different settings, we have four combinations. In the following, we discuss different solutions corresponding to each combination.

A. Single flow without backups

We start with a simple case that there is only a single flow request, making our objective simplified as $\min D_f$. If the availability of active VNF instances in Φ_f is high and we have $\prod \alpha_\phi \geq \alpha_f$, then no backup instance for the requested service chain is needed. When there is only one flow f in the flow set $\mathcal{F}$, we can eliminate all links, whose capacities are less than λ_f. This makes our formulation have one less constraint. Our problem becomes deploying the requested service chain of f and finding a path for f, in order to minimize its expected total transmission delay.

First, suppose that the availability of the active VNF instances is high enough and no backup instance is needed. There are two constraints: link transmission delay d_e and server capacity c_v. Based on their uniform or non-uniform settings, we can divide the problem into four cases and propose one corresponding solution: (1) case 1 (Alg. 1): uniform d_e and uniform c_v, (2) case 2 (Alg. 2): uniform d_e and non-uniform c_v, (3) case 3 (Alg. 3): non-uniform d_e and uniform c_v, and (4) case four (Alg. 4): non-uniform d_e and non-uniform c_v. Next, we discuss each solution in details as follows:

In Alg. 1, line 1 calculates the length of f's routing path because the server capacity is uniform. Since the link transmission delay is also uniform, no matter which link we select to route the flow, the total transmission delay is identical for any path with a fixed length. Line 2 selects the routing path with $l-1$ hops. The result of the deployment plan and the routing path of the flow f are returned in line 3. The insight of the solution is to find a path with enough total server capacity for deploying all instances of the service chain.

When the server capacity is non-uniform, our proposed Alg. 2 is a little more complex than Alg. 1. The insight of Alg. 2 is to select a path that consists of the servers with the largest capacity in order to shorten the flow routing path. Lines 1-2 use up the capacity of the servers hosted at the source and the destination of the flow. We sort the server capacity in line 3. Line 4 routes the flow f to the path along servers with the largest capacity until all instances of its requested service chain are deployed. The deployment plan and the routing path are returned in line 5.

In case 3, a Dynamic Programming (DP) based algorithm is included in Alg. 3. In order to ensure the deployment feasibility of the service chain, we need to check the total capacity of the bypass servers larger than the length of the requested service chain. Then we define DP(v_i,k) as the minimum delay from the source of the flow src to the current node v_i with k hops. Here we list the formulation of our dynamic programming method as:

$$\min_{P_f} \text{DP}(dst, |P_f|) \tag{6}$$

$$\text{st.} \sum_{v_i\in P_f} c_i \geq |\Phi_f| \tag{7}$$

$$\text{DP}(src,k) = 0 \qquad \forall k \leq |P_f| \tag{8}$$

$$\text{DP}(v_i,k) = \min_{i\neq j\neq l}\{\text{DP}(v_j,k-1) + d_{jl} + d_{li}\} \tag{9}$$

Algorithm 3 Solution of case 3:
One Flow without backup Solution (ONS)

In: Flows f, its required service chain ϕ_f and sets of vertices V, links E;

Out: The placement plan and its routing path;

1: Sort link transmission delay of links in an increasing order.
2: Calculate the number of servers with a total capacity no less than $|\phi_f|$, which is $\lceil \frac{|\phi_f|}{c_v} \rceil$.
3: Use the dynamic programming solution to calculate the minimum transmission delay from v_s to v_d with a hop length as $\lceil \frac{|\phi_f|}{c_v} \rceil$.
4: Return the deployment plan and its routing path.

The initial value is DP(src, k), which equals 0 for all $k \leq |P_f|$. The value of DP(v_i, k) is the minimum value among all the values, which are the summation of the delay of its previous hop v_j with $k-1$ hops, DP($v_j, k-1$), plus the delay between v_i and v_j through v_l, which is $d_{jl} + d_{li}$. We also have $i \neq j \neq k$. Our objective is to find the minimum delay between the source src and the destination dst with $|P_f|$ hops, which is DP($dst, |P_f|$). We propose the complete algorithm in Alg. 3 for the case with settings of various link transmission delay, and uniform server capacities. Line 1 sorts all link transmission delays in an increasing order. Line 2 calculates the number of servers to deploy all the active instances, which is the value of $|P_f|$. We apply the dynamic programming method to find the path with the minimum transmission delay DP(dst, $|P_f|$) in line 3. Line 4 returns the result.

When both the link transmission delay and the server capacity are non-uniform, we cannot figure out the path length of the flow f. As a result, we need to try all possible values of the length, starting at 1. The maximum path length is the total number of leaf switches, meaning the instances of the requested service chain are deployed at all leaf-switch-connected servers. We propose the solution in Alg. 4. Line 1 specifies the path length. When we fix the path length, Alg. 3 is applied in line 2 because the remaining problem is the same with case 3. Line 3 selects the optimal one among all the obtained paths. The deployment plan and the routing path are returned in line 5.

***Theorem** 2:* Algs. 1, 2, 3, and 4 are optimal. The time complexities of the proposed solutions are listed in Tab. II.

Proof: Simply speaking, for the cases with uniform link transmission delay, our solutions are based on the shortest path algorithm. For the cases with various link transmission delay, our solutions are based on the dynamic programming method. Because of the optimality of the original algorithms, our solutions are also optimal. Explanations are as follows:

(1) For Alg. 1, when we have uniform server capacity, we can calculate the number of servers that can hold all VNF instances as $l = \lceil \frac{|\phi_f|}{c_v} \rceil$. If the link transmission delay is also uniform, the minimum total delay is $l \cdot d_e$ when we select any path from src_f to dst_f along l servers, which has $l-1$ hops. The time complexity is $O(N)$ for finding a path with l links.

Algorithm 4 Solution of case 4

In: Flows f, its required service chain ϕ_f and sets of vertices V, links E;

Out: The placement plan and its routing path;

1: Increase the hop length of the path l from 1 to N;
2: Find the minimum delay l-hop path with enough server capacity by Alg. 3;
3: Select the path with the minimum delay among all various-hop paths;
4: Return the deployment plan and its routing path.

Time Complexity	Uniform d_e	Non-uniform d_e
Uniform c_v	$O(N)$	$O(N \log N)$
Non-uniform c_v	$O(N^3)$	$O(N^4)$

TABLE II: Time complexity.

(2) For Alg. 2, when the server capacity is non-uniform but the link transmission delay is still uniform, in order to minimize the total transmission delay, we need to find the minimum number of servers to hold all VNF instances. Because of the full connectivity property of the leaf-spine topology, we select the servers in decreasing order of their capacity until the total capacity can hold all the VNF instances. Then the path is optimal with the minimum transmission delay. As we need to sort the capacity of all servers, the time complexity is $O(N \log N)$.

(3) For Alg. 3, when the link transmission delay is non-uniform, the solutions are not trivial. We can treat the transmission delay of each link as its weight, then our problem changes into finding a minimum total weight path with $l-1$ links between two fixed nodes. We solve this problem by using the dynamic programming method, which generates an optimal solution. According to the formulation of our DP solution, the time complexity is $O(N^3)$ for finding a path with l links.

(4) For Alg. 4, when both link delay and server capacity are non-uniform, our problem becomes more complicated. We cannot simply know the length of the optimal path, so we have to try it as a priori ranging from 1 to N. When the length of the path is k, we apply the same dynamic programming method as in case 3. If there is a feasible solution, then it is the optimal path with the minimum transmission delay. As the length of the optimal path is unknown, we need to try N times at most. For each fixed length, the time complexity is the same as case 3, which is $O(N^3)$. Then, the total time complexity is $O(N^3 \cdot N) = O(N^4)$. Since our solutions for the cases are optimal, our theorem holds. ■

B. Single flow with backups

When the availability requirement α_f is higher than the availability of the service chain with only active instances $\prod \alpha_\phi$, we need to add backup instances of selected VNFs in order to increase the availability of the provided service chain. Then when a failure happens, we can use its corresponding backup instance of the same network function and the service chain would still be available. When we need to add backup instances, our problem needs to be decomposed into three

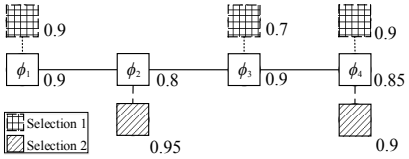


Fig. 3: Two backup selection strategies.

sub-problems: 1) backup selection: which VNFs need to have backups, 2) deployment: where to deploy the service chain with backup instances at VNF nodes, and 3) routing: how to route the flow f to get processed by its requested service chain under all failure cases.

For the backup instance selection problem, we apply the solution in [26], which has already studied the same problem: finding the minimum number of backup VNFs in order to guarantee a certain degree of availability of a service chain. They also prove that the backup selection problem is NP-hard. Here we use a motivating example, shown in Fig. 3, to illustrate our backup selection problem. We are given a service chain that consists of 4 primary VNFs. Their availability probabilities are 0.9, 0.8, 0.9, and 0.85. Thus, the availability of the service chain is $0.9 \cdot 0.8 \cdot 0.9 \cdot 0.85 = 0.5508$, which cannot meet most flows' requirements. As a result, we need to induct redundancy in order to mask failures. Here we use the traditional active/standby (i.e., 1+1) redundancy model [27] such that the primary VNF can be failed over to the standby entity in case it fails. We suppose the availability requirement is 0.75. The solid line and the dashed line represent two redundancy deployment strategies. Their availability probabilities are calculated as $(1-0.1\cdot0.1)\cdot0.8\cdot(1-0.1\cdot0.3)\cdot(1-0.1\cdot0.15) = 0.7567$ and $0.9\cdot(1-0.2\cdot0.05)\cdot0.9\cdot(1-0.1\cdot0.15) = 0.7898$, respectively. While both strategies can achieve the availability requirement, it is clear that the solid one uses fewer backup VNFs and may save resources for other chain requests.

Alg. 5 is the complete algorithm to deploy the active and backup instances for the case with the settings of various link transmission delays and uniform server capacity. Line 1 sorts all link transmission delays in an increasing order. Line 2 calculates the number of servers needed to deploy all the active instances. We apply the dynamic programming method to find the path with the minimum transmission delay in line 3. Line 4 sorts the availability of all backup instances in the service chain. Line 5 decides the functions with backup instances and deploys the backup instances primarily. For the other instances, we select the server to deploy them in line 6. Line 7 returns the final deployment plan.

Here we provide a motivating example in Fig. 4. We use the result of the above example as the service chain with backup instances that need to be deployed in a toy leaf-spine topology, shown in Fig. 4. The service chain is required by one flow f, whose source and destination are as shown in the figure. We assume all links' transmission delays are identical, i.e., uniform. The circles are switches and the rectangles are switch-connected servers. The grey parts illustrate the unavailable resources, which makes the server capacities non-uniform. Fig. 4 shows one optimal deployment strategy with the minimum total transmission delay. The active instances

Algorithm 5 One Flow with Backup Solution (ONBS)

In: Flow f, its required service chain ϕ_f and sets of vertices V, links E;

Out: The placement plan;

1: Sort the hop transmission delay of links.
2: Calculate the number of servers with a total capacity no less than $|\phi_f|$, which is $\lceil \frac{|\phi_f|}{c_v} \rceil$.
3: Use the dynamic programming algorithm to calculate the minimum total transmission delay from v_s to v_d with a path length as $\lceil \frac{|\phi_f|}{c_v} \rceil$.
4: Sort the availability of all backup instances.
5: Deploy the active service chain as well as the top $\lceil \frac{|\phi_f|}{c_v} \rceil \cdot c_v - |\phi_f|$ instances with the highest availability at the selected servers. The sequence of all these active and backup instances follow the order of the service chain.
6: For rest backup instances, deploy them in the rest $\lfloor \frac{|\phi_f|}{c_v} \rfloor$ servers in the increasing order of link transmission delay.
7: Return the deployment plan.

are white rectangles while the backup ones are the slashed rectangles. The routing path of f is shown by the red solid directed line. As the link transmission delay is uniform, the red solid path has the minimum total delay. Additionally, we use the dotted red line to illustrate its rerouted path when the active instance of network function ϕ_2 fails. It is also the minimum rerouted path because it only adds one extra hop to its original path. However, if the link transmission delay is non-uniform, the deployment problem with the objective of minimizing the expected total delay becomes much more complicated, especially when the failure probabilities of each network function are variable.

C. Two flows without backups

In this subsection, we handle the case with two flows, whose required service chains are the same. We also start with the case that both of them do not need to deploy backup instances. Specifically, suppose we have two flows, f and f', that have the same source and destination, as well as their requested service chain. We can also assume they are two sub-flows of one flow, whose traffic rate is divided into different paths. We require their paths to be edge-disjoint for the purpose of load balancing. We propose the solution in Alg. 6. In Alg. 6, line 1 divides the resource at all nodes proportional to the traffic rate ratio of these two flows, which is λ_f/λ'_f. We apply the corresponding solution for a single flow with its allocated server resource in line 2. Then we combine the residual node resource of f with the allocated resource for f' in line 3, and similarly apply the solution for f' in line 4. Line 5 returns the deployment plan. The time complexity is the same as Alg. 3. We illustrate the insight of this heuristic solution in Fig. 5. This solution is performance guaranteed as shown in the following Theorem 3. The extra challenge here is to generate an optimal solution or a performance-guaranteed solution with a better approximation ratio for two such flows without backups.

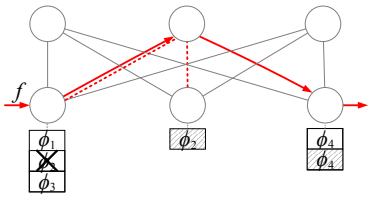


Fig. 4: Deployment of an SFC with backup instances.

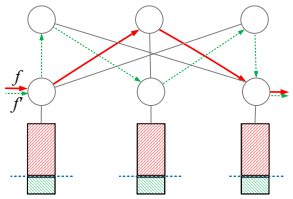


Fig. 5: Illustration of greedy solution.

***Theorem** 3:* The proposed solution has a total expected transmission delay of at most $(1 + \lceil \lambda'_f/\lambda_f \rceil)$ times that of the optimal solution.

Proof: When no backup instance is needed and link delays are uniform, the number of links in the path of each flow is at most $(1 + \lceil \lambda'_f/\lambda_f \rceil)$ times that of the optimal solution for a single flow. Because of uniform link delays, this results in a total expected delay with an approximation ratio of $(1 + \lceil \lambda'_f/\lambda_f \rceil)$ to that of the optimal solution. ■

D. Two flows with backups

The above preliminary results do not handle the case of two flows with backups, or more generally, multiple flows with backups. When we need to add backups for their requested service chains, our problem becomes much more complicated because of the resource competition. We can simply extend the solution for the case of two flows without backup in Alg. 6 by applying Alg. 5 in lines 2 and 4. For future work, we would like to try to generate some approximation heuristic algorithms or apply some solutions of integer programming. Even when there is no backup, our problem can be reduced to a maximum latency problem, which is NP-hard. We would like to try to generate some approximation heuristic algorithms or apply some solutions of integer programming in the future.

Additionally, there is a novel approach to provide backup instances by maintaining only a few backup servers, where each can serve one of multiple functions when the corresponding middleboxes are unavailable. We can also study the resilience issue based on this backup provision model. It adds an extra challenge of determining which network functions to implement in each of the backup servers in order to have a higher availability of the provided service chains. We may rely on the graph theoretical model to propose efficient strategies.

V. Simulations

A. Experimental settings

We do our simulations on the leaf-spine network topologies [10]. The number of leaf and spine switches are 200 and 100, respectively. The link transmission delay is a random number ranging from 0.1 ms to 1 ms. The default uniform link transmission delay is 0.5 ms. The server capacity is a random integer ranging from 10 to 100. The default uniform server capacity is 50. There are 20 types of middleboxes. The availability of active and backup instances for each type is a random number ranging from 0.7 to 0.9, respectively. The source and destination of each flow are generated randomly among all leaf switches. Each service chain has a length ranging from 3 to 6, all of whose middleboxes are selected from the 20 types. The availability requirement of each service chain is a random number ranging from 0.6 to 0.8. The variable is the number of incoming flows ranging from 1000 to 9000 with a stride of 1000.

Algorithm 6 Two Flows Without Backup Solution (TFS)

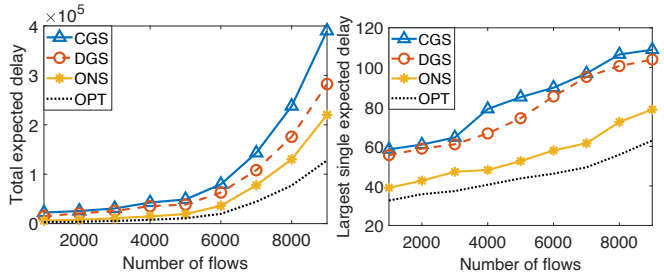
In: Flows f and f', their required service chain ϕ_f and sets of vertices V, links E;
Out: The placement plan;
1: Divide the resource at all nodes proportional to λ_f/λ'_f;
2: Apply Alg. 3 for flow f with its allocated resource;
3: Update the allocated resource for flow f' with f residual resource;
4: Apply Alg. 3 for flow f' with its allocated resource;
5: Return the deployment plan.

B. Comparison algorithms and performance metrics

We include three comparison algorithms to evaluate our proposed solutions from different perspectives. Optimal solution (OPT): We use the integer programming solver to get the optimal solution of our input. Distributed Greedy Solution (DGS): All active and backup instances are deployed in different servers. All instances in the service chain are greedily deployed in sequence on the server with residual capacity, making the increment of the transmission delay minimized. Central Greedy Solution (CGS): All backup instances are deployed in one server with enough residual capacity. We select the server that can make the expected transmission delay minimized. We evaluate three of our proposed algorithms: (1) ONS: algorithm for the single flow without backup instances, (2) ONBS: algorithm for the single flow with backup instances, and (3) TFS: algorithm for the two flows without backup instances. There are four performance metrics that we use to compare the algorithms: (1) the total expected transmission delay, (2) the largest expected transmission delay among all flows, (3) the length of the longest flow path, and (4) the execution time. We evaluate all metrics for three cases: (1) one flow without backups, (2) one flow with backups, and (3) two flows without backups.

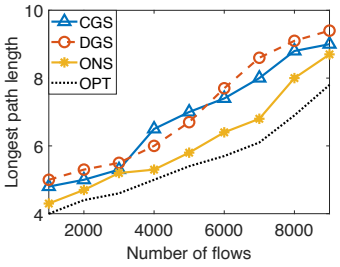
C. Results for one flow without backups

Fig. 6 shows the results of changing the number of flows from 1000 to 9000 when we have only one flow without backup instances for their requested service chain at each time. Our proposed algorithm for this case is called Alg. ONS. Fig. 6(a) shows the result of the total expected delay. OPT has the best performance with the minimum total expected transmission delay while our proposed ONS has the second

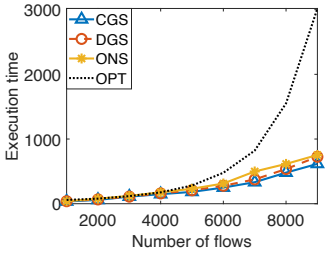


(a) Total expected delay.

(b) Largest single expected delay.

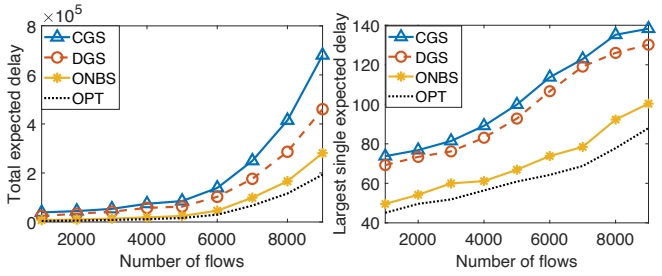


(c) Longest path length.



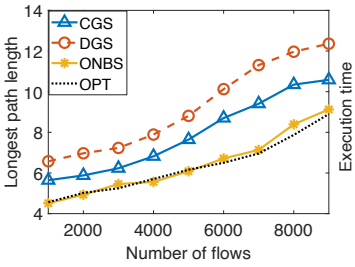
(d) Execution time.

Fig. 6: One flow without backups.

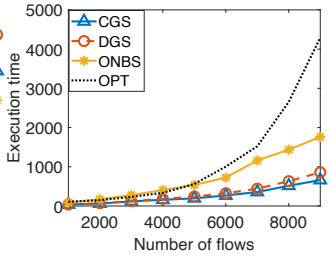


(a) Total expected delay.

(b) Largest single expected delay.



(c) Longest path length.



(d) Execution time.

Fig. 7: One flow with backups.

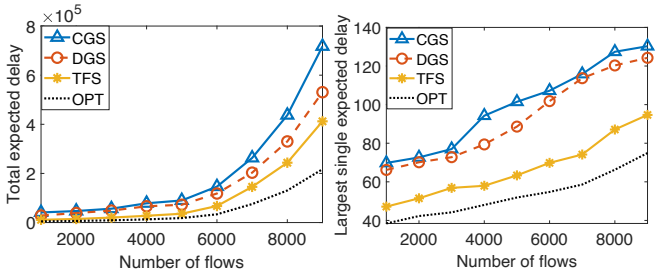
smallest total expected delay. The delay of Alg. ONS is at most 23.1% more than that of the optimal solution. Alg. CGS has the largest total delay because deploying all backup instances in one single server reduces the chance of finding a path with a smaller length. Fig. 6(b) shows the largest expected delay of a single flow when we change the number of flows. The largest expected delay among all flows becomes larger. When the number of flows increases, the largest single delays and the length of the longest path for running all algorithms become larger. This is because it is more difficult to find a shorter path for a flow to place all instances for its required service chain. The result for the longest flows' path is shown in Fig. 8(c). The path length becomes larger, especially for the Algs. CGS and DGS. The increment is more than 80.3% when the number of flows changes from 1000 to 9000. Our proposed ONS algorithm has a close performance with the optimal solution, which indicates its efficiency. Alg. CGS has the worst performance with the largest path length. Fig. 6(d) shows the execution time of running all the algorithms. OPT needs to find the minimum delay value in each time, which makes it much more time-consuming than the others. The changing tendency of the execution time is in a form of exponential increment. The execution times of the other three algorithms are quite close while the performance of our proposed Alg. ONS is much better than Algs. CGS and DGS.

D. Results for one flow with backups

Fig. 7 shows the results of running all algorithms when we have one flow with the need to place backup instances for their requested service chain. Our proposed algorithm for this case is called Alg. ONBS. More VNF instances are deployed in order to meet the availability requirement of the service chains. So the delay and the path length of each flow become a little bit larger. Fig. 7(a) shows the result of the total expected delay. OPT has the best performance with the minimum total expected delay while our proposed ONBS has the second smallest expected delay. Alg. CGS has the worst performance with the largest total transmission delay, which can be more than 5 times than that of Alg. OPT. This indicates that it is not enough to only deploy all backup instances on a single server, which directly results in a larger path length. In Fig. 7(b), the largest expected delay among all flows becomes larger because more flows are inserted, and less server capacity is left. When the number of flows increases, the increment of the total delay of all algorithms becomes larger. This is because it becomes more difficult to find a shorter path for a flow to place all instances for its required service chain. The path length of the incoming flow becomes longer, which has a direct impact on the total transmission delay. Fig. 7(c) shows the result of the largest path length of a single flow. Our proposed ONBS algorithm has a close performance compared with the optimal solution, which indicates its efficiency. Fig. 7(d) shows the execution time of running all the algorithms. OPT needs to find the minimum delay value in each time, which makes it much more time-consuming than the other algorithms. The result demonstrates the trade-off between the performance and the efficiency of the algorithms. Our proposed Alg. ONBS has a much lower execution time than Alg. OPT while their performances on the last three metrics are close.

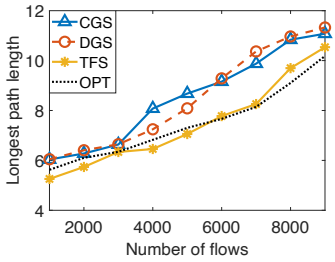
E. Results for two flows without backups

Fig. 8 shows the results of changing the number of flows from 1000 to 9000 when we have two flows without backup instances each time. Our proposed algorithm for this case is called Alg. TFS. Fig. 8(a) shows the result of the total expected delay. When the number of flows increases, the increment of the total delay of all algorithms becomes larger. This is because it is more difficult to find a shorter path for a flow to place all instances for its required service chain. The path length of the newly-coming flow becomes longer, which has a direct impact on the total transmission delay. OPT has the best performance with the minimum total delay while our proposed

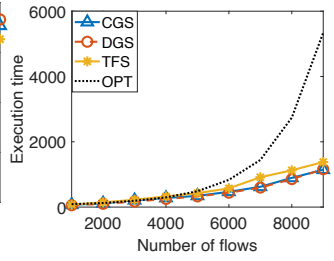


(a) Total expected delay.

(b) Largest single expected delay.



(c) Longest path length.



(d) Execution time.

Fig. 8: Two flows without backups.

TFS has the second smallest total expected delay. In Fig. 8(b), the largest expected delay among all flows becomes larger because more flows are inserted, and less server capacity is left. Our proposed TFS algorithm has a close performance with the optimal solution, which indicates its efficiency. On average, the largest delay of Alg. TFS is just 27.9% more than that of the optimal solution. As for the result for the longest flows' paths, shown in Fig. 8(c), Alg. CGS has the worst performance with the largest path length. This indicates that it is not enough to only deploy all backup instances on a single server, which directly results in a larger path length. Fig. 8(d) shows the execution time of running all the algorithms. OPT needs to find the minimum delay value in each time, which makes it much more time-consuming than the other algorithms.

VI. Conclusion

We not only consider the deployment of network functions on specific topologies, but also study the service chain resilience issue. Our objective is to minimize the maximum expected transmission delay of all flows based on the VNF failure probability distribution. For the general case, our problem is formulated as an Integer Programming problem. Then we specialize several settings and focus on the leaf-spine topology, where we propose performance-guaranteed solutions in various scenarios: For the case of a single flow without backup instances, we propose four optimal solutions corresponding to four combinations of link transmission delay and server capacity. For the case of a single flow with backup instances, we extend the algorithms. For the case of two flows without backup instances, we propose efficient algorithms with approximation ratios inspired by the solutions for a single flow.

References

[1] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, "Nfp: Enabling network function parallelism in nfv," in *SIGCOMM 2017*.

[2] Y. Chen, J. Wu, and B. Ji, "Virtual network function deployment in tree-structured networks," in *ICNP 2018*.

[3] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "Opennf: Enabling innovation in network function control," in *SIGCOMM 2014*.

[4] C. Voudouris, G. Owusu, R. Dorne, and D. Lesaint, *Service chain management: Technology innovation for the service business*. Springer Science & Business Media, 2007.

[5] Z. Zhou, Q. Wu, and X. Chen, "Online orchestration of cross-edge service function chaining for cost-efficient edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 8, pp. 1866–1880, 2019.

[6] J. Sherry, S. Ratnasamy, and J. S. At, "A survey of enterprise middlebox deployments," in *Semantic Scholar*, 2012.

[7] J. Zhang, Z. Wang, C. Peng, L. Zhang, T. Huang, and Y. Liu, "Raba: Resource-aware backup allocation for a chain of virtual network functions," in *INFOCOM 2019*.

[8] 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 Transactions on Network and Service Management*, vol. 14, no. 3, pp. 554–568, 2017.

[9] M. T. Beck, J. F. Botero, and K. Samelin, "Resilient allocation of service function chains," in *NFV-SDN 2016*.

[10] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, F. Matus, R. Pan, N. Yadav, G. Varghese *et al.*, "Conga: Distributed congestion-aware load balancing for datacenters," in *SIGCOMM 2014*.

[11] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1409–1434, 2018.

[12] Y. Sang, B. Ji, G. R. Gupta, X. Du, and L. Ye, "Provably efficient algorithms for joint placement and allocation of virtual network functions," in *INFOCOM 2017*.

[13] G. Sallam, G. R. Gupta, B. Li, and B. Ji, "Shortest path and maximum flow problems under service function chaining constraints," in *INFOCOM 2018*.

[14] T. Lukovszki, M. Rost, and S. Schmid, "Approximate and incremental network function placement," *Journal of Parallel and Distributed Computing*, 2018.

[15] K. Poularakis, G. Iosifidis, G. Smaragdakis, and L. Tassiulas, "One step at a time: Optimizing sdn upgrades in isp networks," in *INFOCOM 2017*.

[16] A. Tomassilli, F. Giroire, N. Huin, and S. Pérennes, "Provably efficient algorithms for placement of service function chains with ordering constraints," Ph.D. dissertation, Université Côte d'Azur, CNRS, I3S, France; Inria Sophia Antipolis, 2018.

[17] H. Yu, J. Yang, C. Fung, R. Boutaba, and Y. Zhuang, "Ensc: Multi-resource hybrid scaling for elastic network service chain in clouds," in *ICPADS 2018*.

[18] G. Even, M. Medina, G. Schaffrath, and S. Schmid, "Competitive and deterministic embeddings of virtual networks," *Theoretical Computer Science*, vol. 496, pp. 184–194, 2013.

[19] T. Wang, H. Xu, and F. Liu, "Multi-resource load balancing for virtual network functions," in *ICDCS 2017*.

[20] S. G. Kulkarni, G. Liu, K. Ramakrishnan, M. Arumaithurai, T. Wood, and X. Fu, "Reinforce: Achieving efficient failure resiliency for network function virtualization based services," in *CoNEXT 2018*.

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

[22] R. Xia, H. Dai, J. Zheng, R. Gu, X. Wang, and G. Chen, "Safe: Service availability via failure elimination through vnf scaling," in *ICPP 2019*.

[23] X. Shang, Y. Huang, Z. Liu, and Y. Yang, "Reducing the service function chain backup cost over the edge and cloud by a self-adapting scheme," in *INFOCOM 2020*.

[24] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," in *SIGCOMM 2011*.

[25] S. Lal, T. Taleb, and A. Dutta, "Nfv: Security threats and best practices," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 211–217, 2017.

[26] J. Fan, C. Guan, Y. Zhao, and C. Qiao, "Availability-aware mapping of service function chains," in *INFOCOM 2017*.

[27] R. Potharaju and N. Jain, "Demystifying the dark side of the middle: A field study of middlebox failures in datacenters," in *IMC 2013*.