On Improving Service Chains Survivability Through Efficient Backup Provisioning

Saifeddine Aidi*, Mohamed Faten Zhani*, Yehia Elkhatib*‡

*École de Technologie Supérieure (ÉTS Montreal), Montreal, Quebec, Canada
‡MetaLab, School of Computing and Communications, Lancaster University, UK
E-mail: saifeddine.aidi.1@ens.etsmtl.ca, mfzhani@etsmtl.ca, {i.lastname}@lancaster.ac.uk

Abstract—With the growing adoption of Software Defined Networking (SDN) and Network Function Virtualization (NFV), large-scale NFV infrastructure deployments are gaining momentum. Such infrastructures are home to thousands of network Service Function Chains (SFCs), each composed of a chain of virtual network functions (VNFs) that are processing incoming traffic flows. Unfortunately, in such environments, the failure of a single node may break down several VNFs and thereby breaking many service chains at the same time.

In this paper, we address this particular problem and investigate possible solutions to ensure the survivability of the affected service chains by provisioning backup VNFs that can take over in case of failure. Specifically, we propose a survivability management framework to efficiently manage SFCs and the backup VNFs. We formulate the SFC survivability problem as an integer linear program that determines the minimum number of required backups to protect all the SFCs in the system and identifies their optimal placement in the infrastructure. We also propose two heuristic algorithms to cope with the large-scale instances of the problem. Through extensive simulations of different deployment scenarios, we show that these algorithms provide near-optimal solutions with minimal computation time.

I. INTRODUCTION

The emergence of Network Function Virtualization (NFV) and Software-Defined Networking (SDN) technologies is currently transforming the way networks are designed and managed as they provide operators much more flexibility to dynamically provision and configure network services. In particular, it is now possible to dynamically create chains of network services (Service Function Chains - SFCs) that can process incoming traffic and steer it across a chain of Virtual Network Functions (VNFs) like routers, IDSs and NATs that are running on virtual machines.

In the last few years, a large body of work has been dedicated to address resource provisioning and management of such SFCs [1]–[6]. Most of existing studies assume the complete availability of the physical infrastructure which is not realistic as failures are common in cloud network infrastructures [7]–[10]. Due to the dependency between virtual network functions in the chain, a single physical node failure in the network could easily bring down many VNFs and hence break several SFCs and make these services unavailable. Such downtime, even for few seconds, not only hurts the reputation of service providers but also incurs high revenue losses depending on the type of the offered service (e.g., $5,600 per minute according to [11]).

Existing proposals to manage failures and mitigate them can be broadly categorized into reactive and proactive techniques [12]. In proactive techniques, backup VNFs are provisioned whenever an SFC is received and embedded. These backups remain idle but are activated only when a failure occurs to take over the service and replace the failed VNFs [13]–[16]. The second category of existing solutions are reactive techniques. These techniques do not pre-allocate backup resources and deal with failures after they occur [15], [17], [18]. Consequently, they need additional time to allocate resources and provision new VNF instances to take over the service. This definitely results in a longer service disruption, which is very costly for service providers [12]. This makes proactive techniques more appealing even though they waste some resources for backup VNFs.

To remediate to this problem and minimize wastage of resources, several research efforts advocate to use shared backup VNFs [14] where the same backup resource can be used to mitigate the failure of a set of VNFs assuming that they do not fail at the same time (i.e., only a single VNF from this set can fail at a time). In this context, this paper investigates possible solutions to ensure the survivability of service chains against single physical node failures by using shared backup resources. Unlike previous work addressing the same problem where backups are shared only between the VNFs of the same chain [13] [14] , our solution assumes that backup VNFs are shared among all the chains embedded in the infrastructure. This significantly reduces the amount of resources used for the backup VNFs while still ensuring all SFCs are protected against single failures.

Our main goal is to ensure the survivability of all embedded SFCs against any single-node failure in the physical infrastructure. We reach this objective by proactively provisioning the minimal number backup VNFs to minimize resource wastage and by carefully placing them in the infrastructure. We also take into consideration the synchronization cost in terms of bandwidth and delay needed to keep the backup nodes up-to-date.

We can summarize the main contributions of this paper as follows:

- We propose a resource management framework with a survivability module. This module allows to provision and manage backup VNFs and it could be
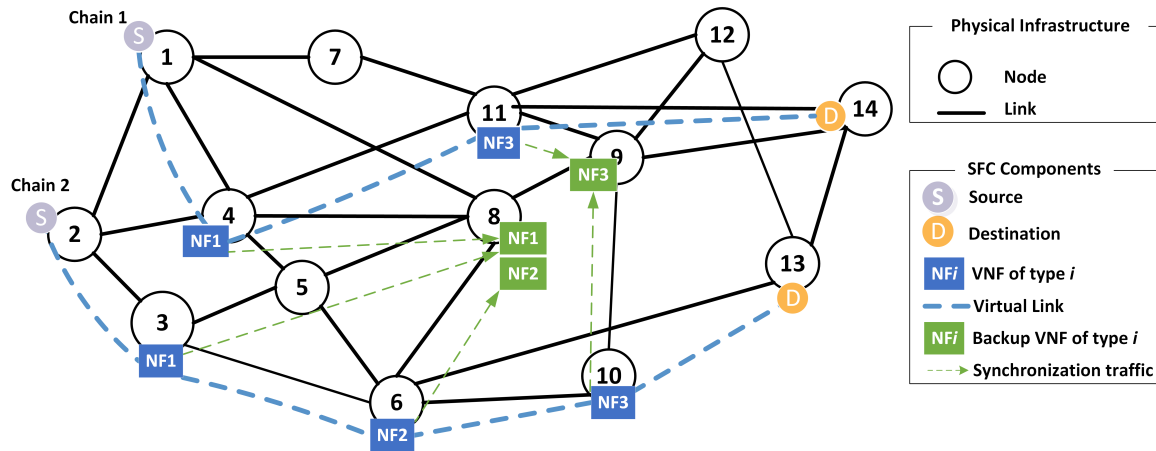
Figure 1: An example of various embedded SFCs sharing backups.

easily integrated into existing SFC resource management frameworks.

- We formulate the backup provisioning and placement problem as an Integer Linear Program (ILP) that finds the optimal number of shared backups for each type of VNF and optimally places them in the physical infrastructure.
- We devise two heuristic algorithms, called BS-Push and BS-Pull, respectively, that aim at solving the problem for large-scale scenarios within a reasonable timescale.
- We evaluate the performance of the proposed heuristics and compare it to the optimal solution found with the proposed ILP solved by the CPLEX optimizer.

The remainder of this paper is organized as follows. Section II provides a detailed description of service chain survivability problem. We discuss relevant related work in Section III. In Section IV and V, we mathematically formulate the addressed problem and then describe the proposed heuristic solutions. We present the experimental results in Section VI and follow up with some conclusions and future work in Section VII.

II. PROBLEM DESCRIPTION

A Service Function Chain is made out from a set of different types of virtual network functions connected in a specific order to form a chain that steers the traffic from and to predefined source and destination [3]. A Virtual Network Function (VNF) is simply a virtual resource (i.e., virtual machine or container) that is running a specific network function (e.g., router, load balancer, NAT, IDS). To build the chain, the VNFs are connected through a set of virtual links having a sufficient amount of bandwidth to handle the traffic. Typically, service function chains are embedded into a physical infrastructure (referred to as NFV Infrastructure - NFVI [9]).

Figure 1 shows an example of two service chains mapped onto a wide area NFV infrastructure. The figure shows for each chain how the VNFs are embedded from the source

to the destination. For instance, chain 2 has traffic coming from physical node 2 towards physical node 13 and it is composed of three VNFs of type 1, 2 and 3 that are embedded in physical nodes 3, 6 and 10, respectively. Chain 1 has only two VNFs of type 1 and 3 that are embedded in physical nodes 4 and 11, respectively.

Once SFCs are embedded into the NFVI, the operator faces the challenging task of ensuring the high survivability of these SFCs. In other words, they need to survive potential network failures in order to minimize service interruptions. However, as mentioned earlier, physical nodes are prone to failures and a single node failure may result in bringing down several VNFs and hence breaking multiple service chains. In this paper, we propose to ensure the survivability of the SFCs affected by a single failure by leveraging shared backups that could be used when the failure occurs. We also assume that a backup VNF should be of the same type of the set of VNFs it is backing up. In other words, a VNF of type i can only back up VNFs of type i. This assumption is reasonable as in practice the backup is a virtual machine that should implement a specific software and hence a backup VNF has to contain exactly the same software stack as the original VNF.

As an example of how shared backups could be placed, we can see in Figure 1 that physical node 9 hosts a backup of VNF type 3 (i.e., NF3) that is shared between the VNF type 3 of chain 1 and that of chain 2. If physical node 11 fails, and hence NF3 of chain 1 becomes out of service, the backup NF3 hosted in 9 takes over and replaces the failed function. Similarly, it can take over the service of NF3 belonging to chain 2 (hosted in node 10) if it fails. The figure also shows other examples of shared VNF backups (e.g., NF1 and NF2 hosted in node 8). It is easy to check that the two service chains shown in this example are perfectly survivable to any single node failure.

Furthermore, backup VNFs are continuously synchronized with the active VNFs to be ready to take over the service in case of failure (see green arrows in Figure 1). For instance,

the backup NF3 hosted in 9 has the state of NF3 hosted in physical node 11 and that of NF3 hosted in 10. Whenever a failure happens, the last state of the failed function will be used when the backup is activated. State synchronization can be done at a different level. For example, at the level of the virtual machine running the function (e.g., memory synchronization [19]) or using customized synchronization scripts depending on the type of the network function (e.g., synchronizing rules in firewalls).

To make sure that state synchronization is efficient, the latency between a VNF and its backup should not exceed a certain bound. Furthermore, synchronization of the VNF state may consume bandwidth that should be minimized. In our work, we ensure to minimize the synchronization delay and consumed bandwidth by limiting the number of hops between each VNF and its backup (for example, the number of hops is limited to 2 in Figure 1).

The main challenge that we are addressing in this paper is how to find the minimal number of backup nodes and to determine their optimal placement in the physical infrastructure for each type of VNF taking into account the synchronization delay and the cost in terms of bandwidth consumption.

III. RELATED WORK

In this section, we provide an overview of representative work on the survivability problem. We note that most existing work focuses on virtual networks survivability and not SFCs. However, they are still valid for our case as a service function chain can be seen as a particular case of a virtual network with a specific topology. In the following, we summarize existing techniques to ensure the survivability of SFCs and virtual networks as either *reactive* or *proactive* [12].

Reactive techniques do not pre-allocate resources for backup but simply deal with a failure when it occurs. This leads to a long convergence time after the failure, resulting in a higher service downtime. On the other hand, proactive solutions anticipate failures and pre-allocate backup resources to ensure fast recovery of the service in case of failures.

Yu et al. [13] considered the case of a single-node failure and introduced two approaches to provision backup nodes. The first approach, called 1-redundant, redesigns the virtual network request into a survivable request by adding a single backup node. The second approach is called k-redundant where k is a constant that represents the number of backup nodes to be provisioned. The problem with these approaches is that a single redundant node may not be enough whereas k redundant nodes might be too much, and hence could lead to a wastage of resources. To address this limitation, the solutions presented in this current work aim at finding the optimal number of backup nodes when the proposed ILP is used or at least minimize it when the proposed heuristics are used.

In the same direction, Ayoubi et al. [14] explored the space between 1 and k to find the optimal number of backup nodes to be incorporated into the requested virtual

network. However, in this solution, the backup virtual nodes are provisioned for each request and hence they are not shared with other virtual networks. Our work is different in that it provisions backups that are shared among all virtual nodes belonging to all virtual infrastructures (i.e., SFCs) embedded in the physical infrastructure. As a result, our solutions further reduce the total number of backups provisioned in the system.

Xiao et al. [17] proposed a topology-aware solution that ensures a rational resource allocation for the virtual network and a fail-over remapping based on a set of pre-computed detour-paths. Rahman et al. [15] also proposed a hybrid approach that benefits from a set of a possible backup detours for each link. These detours are proactively precomputed before the arrival of virtual network requests to allow fast re-routing in case of link failure.

Finally, Bo et al. [18] proposed a greedy algorithm that, in case of a link failure, searches for alternative resources to re-allocate the end-to-end path or re-embed the entire virtual network if resources are not sufficient. This may result in long convergence time and higher service downtime. In [21], Ayoubi et al. have demonstrated the NP-hard nature of the survivability-aware embedding and proposed a polynomial time heuristic algorithm to restore failed services while maintaining the QoS requirements in terms of delays in case of single-node failures. Multiple failures were addressed in [22] where a heuristic was introduced in order to find a backup node. The algorithm is based on filtering techniques to parse the solution space and to speed up the search process for backups.

We summarize in Table I the aforementioned solutions. The table presents the type of solution (i.e., reactive vs. proactive) and it indicates whether it is addressing a single or multiple failures, node or link failures and whether the backup are shared between the virtual nodes of all virtual networks or among the virtual nodes of a single virtual network. As shown in the Table, the novelty of our work lies in the idea of sharing backups between VNFs of the same type that belong to different or virtual networks (or service chains) rather than the same virtual network (or service chain), which further reduces the amount of backup resources while still ensuring the survivability of the virtual networks to any single failure.

IV. SURVIVABILITY MANAGEMENT FRAMEWORK

In this section, we propose a management framework that incorporates a survivability module. Figure 2 shows the main components of this framework. It is made out from the following modules:

• **Service Chain Provisioning Module:** This module allocates the resources for the service chains and instantiates the required virtual machines running the network functions. It also makes use of the SDN controller to provision the required amount of bandwidth and to set the required forwarding rules into the switches to steer the traffic across the VNFs composing each service chain.

Table I: Existing solutions vs. the proposed ones

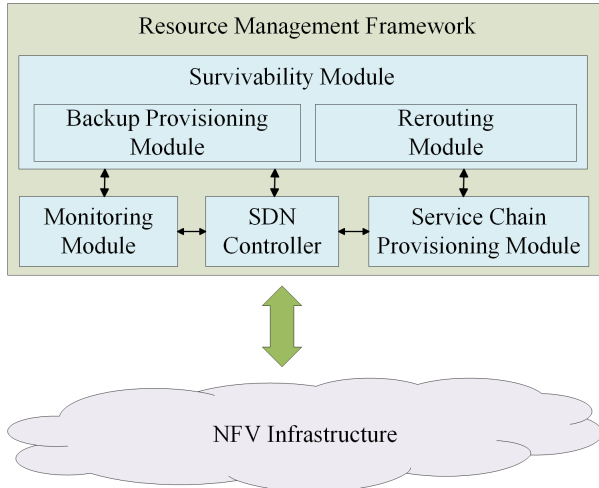| Solutions	Type of solution		Single/Multiple failures		Node/Link failures		Support of Shared backups		
	Proactive	Reactive	Single	Multiple	Node	Link	Supported	Shared among all VNs	Shared among a single VN
Yu et al. [13]	x		x		x		x		x
Ayoubi et al. [14]	x		x		x		x		x
Guo et al. [20]	x		x		x		-		-
Xiao et al. [17]	x			x	x		-		-
Rahman et al. [15]	x		x			x	-		-
Bo et al. [18]		x	x		x		-		-
Ayoubi et al. [21]		x	x		x		-		-
Ghaleb et al. [22]		x		x	x	x	-		-
BS-Pull/BS-Push	x		x		x		x	x	

Figure 2: Architecture of the Proposed Resource Management Framework with the Survivability Module.

It is worth noting that the design of this module is out of the scope of this work. There is a large body of work addressing this module and any of the existing solutions could be used (e.g., [2], [4], [6]).

• **Monitoring Module:** This module is in charge of continuously monitoring the infrastructure physical nodes and links and of feeding other modules at real time with the state of the resources. When a failure is detected, the monitoring module reports to the survivability module, which, in turn, reacts to mitigate the failure and ensure service continuity.

• **Rerouting Module:** In case of failure, the rerouting module redirects the traffic, that is originally destined to the failed VNFs, to the backup VNFs.

• **Backup Provisioning Module:** This module is responsible for finding the minimal number of backups needed to ensure the survivability of the embedded chains and to determine their locations. This module also instantiates the backup VNFs and the synchronization links required to keep them up-to-date. In the following section, we describe in

details the proposed solutions to provision backup VNFs while achieving the sought-after objectives.

V. BACKUP PROVISIONING SOLUTIONS

A. Integer Linear Program

In this section, we formulate the service chain survivability problem as an ILP aiming at minimizing the amount of resources allocated for the backup instances while ensuring a minimal synchronization cost and delay.

• **Infrastructure and chain modeling:** We model the physical infrastructure as a graph denoted by $G = (N, L)$ where N is the set of physical nodes and L is the set of physical links connecting them. Each physical node $n \in N$ has a computing capacity c_n. The capacity c_n is expressed as the maximal number of VNFs that can be hosted by physical node $n \in N$. For sake of simplicity, we assume that each VNF is running on a single virtual machine with a standard size. We hence can see c_n as the maximal number of virtual machines that could be provisioned in the physical node n. We also define d_{in} as the minimum number of hops separating the physical nodes i and n.

We model the service chain as a graph denoted by $S = (V, E)$ where V is the set of its composing VNFs and E is the set of virtual links connecting them. We assume there are different types of VNFs (e.g., Firewall, IDS, NAT). We denote by J the set of VNF types and we define m_{ij} as the number of VNFs of type $j \in J$ embedded in physical node i.

• **Decision Variables:** We define two decision variables. We denote by x_{ij} the number of backup VNFs of type j embedded in physical node i. We also define $y_{ijn} \in \{0,1\}$ to indicate whether the backups provisioned for the VNFs of type j embedded in the physical node i are hosted in the physical node n.

• **Problem constraints:** In order to find a feasible solution, several constraints must be satisfied. For instance, to ensure that the primary and backup VNFs are not

embedded in the same physical node, the following constraint must be satisfied:

$$y_{iji} = 0 \quad \forall i \in N, \forall j \in J \qquad (1)$$

We also need to ensure that all VNFs of type j embedded in the physical node i necessarily have backups in another physical node:

$$\sum_{n \in N} y_{ijn} = 1 \quad \forall i \in N, \forall j \in J \qquad (2)$$

Furthermore, if the backups for the type j VNFs embedded in the physical node i are hosted in physical node n then the total number VNF backups of type j provisioned in n should be higher or equal than the number of VNFs of type j embedded in node i. In other words, we have:

$$if \quad y_{ijn} = 1 \quad then \quad x_{nj} \geq m_{ij} \quad \forall i, n \in N, \forall j \in J \quad (3)$$

This if statement can be translated as the following constraint:

$$m_{ij} \leq x_{nj} + M(1 - y_{ijn}) \quad \forall i, n \in N, \forall j \in J \qquad (4)$$

where M is a constant with a large value (in the vicinity of $10,000$).

Furthermore, to ensure that the physical node hosting the backups have sufficient resources, the following capacity constraint must be satisfied for every physical node n:

$$\sum_{j \in J} m_{nj} + \sum_{j \in J} x_{nj} \leq c_n \quad \forall n \in N \qquad (5)$$

where the first term represents the number of VNFs hosted in the physical node n and the second term represents the number of VNF backups hosted in the same physical node.

Finally, as we have to minimize the synchronization cost and delay between the VNFs and their backups, we limit the number of hops between each VNF and its backup to a limited number of hops denoted by d_{max}. Thus, we have:

$$if \quad y_{ijn} = 1 \quad then \quad d_{in} \leq d_{max} \quad \forall i, n \in N, \ \forall j \in J \quad (6)$$

The previous statement can be also written as the following constraint:

$$d_{in} \leq d_{max} + M(1 - y_{ijn}) \quad \forall i, n \in N, \forall j \in J \qquad (7)$$

where M is a constant with a large value.

It is also worth noting that, for sake of simplicity, we assume that the number of hops between two physical nodes reflects the time delay between them. However, this may not be always true. In this case, our model can be easily updated to consider the propagation delay between the nodes by defining d_{in} as the delay of the shortest path between nodes i and n, and d_{max} as the maximum delay required between a VNF and its backup.

• **Objective function:** Our ultimate goal is to minimize the amount of resources used by the backup VNFs while satisfying all the aforementioned constraints. This can be achieved by minimizing the total number of backups in all the physical nodes of the physical infrastructure. The objective function can be then written as:

$$\min \sum_{i \in N} \sum_{j \in J} x_{ij} \qquad (8)$$

B. Heuristic Algorithms

In this section, we will present two heuristic solutions designed to solve the survivability problem. We call the first algorithm *Backup Sharing "Pull"* (BS-Pull) as we are looking at each physical node to find the maximum number of VNFs that it can backup (we refer to this as *pulling*). The second algorithm is called *Backup Sharing "Push"* (BS-Push) as the algorithm tries to push the coverage of the physical node in order to let it host backups of VNFs that are as spread as possible in several physical nodes.

Both algorithms are carried out in two phases. The first phase aims at finding the candidate physical nodes that satisfy the constraints of number of hops and the capacity for each type of VNFs. The second phase aims at selecting among

---

Algorithm 1 BS-Pull

1: **Inputs**
2: N: set of physical nodes
3: u_n: total number of VNFs hosted in physical node n
4: m_{ij}: number of type j VNFs hosted in physical node i
5: **for** $j \in J$ **do** ▷ *Parsing VNF types*
6:     $BNodes(j) \leftarrow \emptyset$ ▷ *set of physical nodes whose type j VNFs have already backups*
7:     **repeat**
8:         ▷ *Parsing all potential hosting nodes*
9:         **for** $n \in N$ **do**
10:             $SNeigh(n) \leftarrow \emptyset$ ▷ *set of source neighbors for physical node n*
11:             $b_n \leftarrow 0$ ▷ *number of type j VNFs able to use shared VNF backups hosted in node n*
12:         ▷ *Finding source neighbors of n*
13:         **for** $i \in N \setminus (BNodes(j) \cup \{n\})$ **do**
14:             **if** $d_{ni} \leq d_{max}$ & $m_{ij} + u_n \leq c_n$ **then**
15:                 $SNeigh(n) \leftarrow SNeigh(n) \cup \{i\}$
16:                 $b_n \leftarrow b_n + m_{ij}$
17:             **end if**
18:         **end for**
19:         **end for**
20:     ▷ *Finding the node n_{host} that maximizes the number of type j VNFs that are backed up*
21:     $n_{host} = \arg\max_n b_n$
22:     ▷ *Compute the number of shared backups s*
23:     $s = \max_{i \in SNeigh(n_{host})}(m_{ij})$
24:     ▷ *Allocate backups and update $u_{n_{host}}$*
25:     Allocate (s backups, VNF type j, host n_{host})
26:     $BNodes(j) \leftarrow BNodes(j) \cup SNeigh(n_{host})$
27:     **until** $SNeigh(n) = \emptyset \ \ \forall n \in N$
28: **end for**

the candidate nodes the ones that should host the backups. The difference between the two algorithms lies in the way the candidates hosting nodes are selected. In the following, we provide the details of the two proposed algorithms.

• **Algorithm BS-Pull:** Algorithm 1 describes the BS-Pull algorithm. It aims to allocate VNF backups for each VNF type one by one. Assuming we consider VNF type $j \in J$ first, all nodes in the physical infrastructure are assumed to be able to host backups for type j VNFs. Our goal in the following steps is to select which node or nodes could really host these backups and how many backups per node.

We first define *the source neighbors* of a physical node n (i.e., $SNeigh(n)$) as the set of physical nodes that could be reached from n within at most d_{max} hops and such as node n has enough resources to host the backup VNFs required to back up type j VNFs hosted in any of these source neighbors. In other words, if the backup VNFs are provisioned in node n, they can be shared among all the source neighbors of n.

For each physical node $n \in N$, we compute the set of source neighbors $SNeigh(n)$ and we also compute b_n, which is the number of VNFs of type j that could share the VNF backups that could be provisioned in physical node n (Lines 9-19). The higher b_n is, the higher is the number of VNFs sharing the backups. As a result, to maximize backup sharing, we select the node $n_{host} \in N$ that has the highest value of b_n to be the hosting node of the VNF backups. We then allocate the backup VNFs in the node n_{host} (function Allocate in Line 25). We repeat this operation until no source neighbors could be identified for all the physical nodes. Having no source neighbors for all physical nodes means that either there is no enough resources to host the backup VNFs (while satisfying the constraint on the number of hops) or there is no VNFs of type j that are left without backups. Finally, the whole process is repeated for all VNF types.

• **Algorithm BS-Push:** in this algorithm, we are adopting an approach different from the first one. For a particular physical node, our goal is maximize the number of its source neighbors that are using it (i.e., the physical node) to host their VNF backups (unlike BS-Pull that maximizes the number of VNFs that are backed up by the physical node but not the number of source neighbors using it).

As shown in Algorithm 2, similar to BS-Pull, BS-Push computes the set of source neighbors for all the physical nodes (Line 8). However, the algorithm selects the node n_{host} that has the highest number of source neighbors in order to host the VNF backups of all these neighbors (Line 10). The backup resources are then allocated and associated to all neighbors of the selected node (Line 14). The operation is then repeated until there are no more source neighbors for all physical nodes. Finally, the whole process is applied again for each of the VNF types.

Algorithm 2 BS-Push

1: **Inputs**
2: N: set of physical nodes
3: u_n: total number of VNFs hosted in physical node n
4: m_{ij}: number of type j VNFs hosted in physical node i
5: **for** $j \in J$ **do**   ▷ *Parsing VNF types*
6:     $BNodes(j) \leftarrow \emptyset$ ▷ *set of physical nodes whose type j VNFs have already backups*
7:     **repeat**
8:         Compute $SNeigh(n)$ $\forall n \in N$
9:         ▷ *Finding the node n_{host} that is connected to to the maximum number of neighbors*
10:         $n_{host} = \arg\max_n |SNeigh(n)|$
11:         ▷ *Compute the number of shared backups s*
12:         $s = \max_{i \in SNeigh(n_{host})}(m_{ij})$
13:         ▷ *Allocate backups and update $u_{n_{host}}$*
14:         Allocate (s backups, VNF type j, host n_{host})
15:         $BNodes(j) \leftarrow BNodes(j) \cup SNeigh(n_{host})$
16:     **until** $SNeigh(n) = \emptyset$ $\forall n \in N$
17: **end for**

VI. SIMULATION AND RESULTS

In this section, we compare the performance of the proposed algorithms with the optimal solution provided by CPLEX in terms of total number of backups and the execution time. To do so, we implemented the algorithms in C and simulated the physical infrastructure and the service chain embedding. We have considered a network with 24 physical nodes with different computing capacities randomly generated from 20 to 50 virtual machines. For simplicity, we assume that all virtual machines have the same resource capacities and that a single VNF is hoted by a single virtual machine. Furthermore, the physical nodes are connected through 55 physical links that were randomly generated. We assume the embedding of service chains (i.e., VNFs and virtual links) is already carried out by an existing VNF placement algorithm. In our experiments, we used the resource allocation algorithm for service chains that was proposed by Racheg et al. [4].
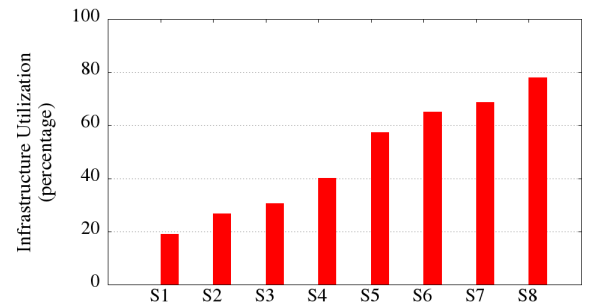

Figure 3: Studied scenarios with different infrastructure utilization.

We considered 8 different embedding scenarios where the utilization of the infrastructure has been gradually increased as shown in Figure 3. We can see in the figure

that we have low-utilization scenarios (i.e., s1 to s4) where utilization is less than 50% and also high-utilization scenarios (e.g., s5 to s8) where utilization is higher than 50%.

The objective of the experiments is to compare the number of backups and the number of VNFs left without backup provided by our algorithms with the one provided by CPLEX.
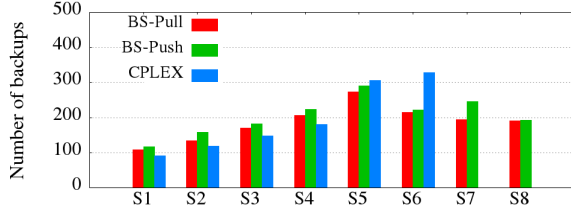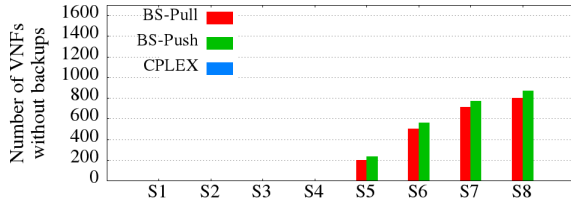

Figure 4: Total number of provisioned backups.

Figure 5: Number of VNFs without backup provisions.

A. Number of backups

Figure 4 compares the total number of backups found with the proposed heuristic algorithms with the optimal solution produced by CPLEX for each scenario. We can see that for low-utilization scenarios (i.e., S1–S4), the two heuristics provide a slightly higher number of backups compared to the optimal solution provided by CPLEX in low utilization scenarios, indicating that their solutions are not far from the optimal ones. In addition, we notice that BS-Pull generally provides lower numbers of backups compared to PS-Push.

For high-utilization scenarios (S5–S8), it becomes harder to find even a feasible solution, i.e., a solution that ensures that all VNFs in the infrastructures have backups. We can see that, for S5 and S6, only CPLEX could find a solution (which is optimal) whereas the two heuristics do not ensure that there are backups for all VNFs as depicted in Figure 5, which shows that many VNFs are left without backups using the heuristics when utilization is high.

Figure 4 also shows that for scenarios S7 and S8 that have high utilization (above 70%), CPLEX does not find an optimal solution that allow to provide backups for all VNFs. This is because there is no enough free resources in the infrastructure to provision all the required backups. The heuristics in this case still provide a solution even though several VNFs are left without backups as shown in Figure 5.

B. Execution Time

Figure 6 depicts the execution time of the two proposed algorithms compared to that of CPLEX for each

of the 8 studied scenarios. The execution time for CPLEX goes from 2s to 7min (S6) as the infrastructure utilization increases. This is because the number of variables in the ILP increase (e.g., the number of nodes and that of VNFs) and makes the problem harder to solve because of the large research space. It is also clear from the figure that the two algorithms' execution times does not significantly change as the utilization of the infrastructure increases. We note, again, that beyond the sixth scenario (i.e., for S7–8), no optimal solution could be found due to the unavailability of free resources in the infrastructure.
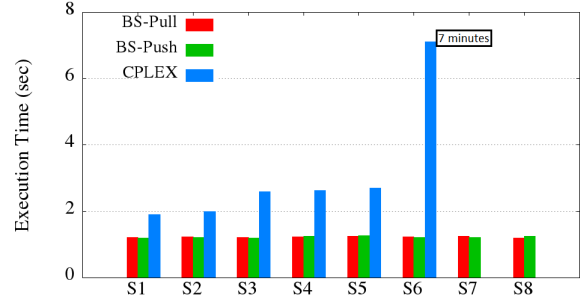

Figure 6: Execution time of the different algorithms.

C. Synchronization Cost

Figure 7 shows the average number of hops between an embedded VNF and its backup for the different solutions and across the studied scenarios. The number of hops provides some insight about the potential synchronization cost in terms of delay and bandwidth. The results show that for the two algorithms as well as CPLEX, the number of hops is below the maximal number of hops d_{max} specified as input to all solutions (i.e., d_{max} is equal to 2 in our experiments).
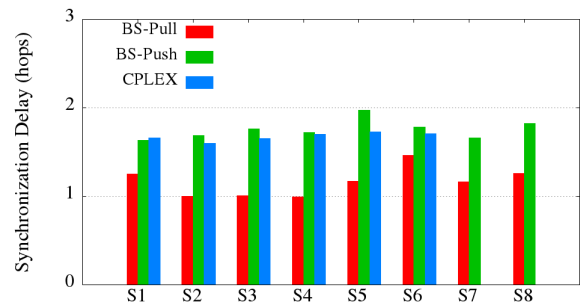

Figure 7: Synchronization distance of the different algorithms.

VII. CONCLUSION

In this paper, we addressed one of the uprising challenges faced by the infrastructure providers: the survivability of the service chains against node failures. We hence proposed a novel solution that provision the minimal number of shared backup VNFs that minimizes the amount of resources allocated for the backup.

We hence formulated the problem as an ILP and then proposed two heuristic algorithms to solve the problem for large-scale scenarios. Through extensive simulations, we demonstrated that our algorithms provide solutions that are close to the optimal one provided by CPLEX while they reduce the execution time considerably.

As a future work, we aim to further optimize both algorithms in order to reduce their complexity. We also plan to extend this work to take into consideration multiple node failures.

REFERENCES

[1] M. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, "Data center network virtualization: A survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 2, pp. 909–928, 2013.

[2] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 13, no. 3, pp. 518–532, 2016.

[3] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[4] W. Racheg, N. Ghrada, and M. F. Zhani, "Profit-driven resource provisioning in NFV-based environments," in *IEEE International Conference on Communications (ICC)*, 2017, pp. 1–7.

[5] L. Qu, C. Assi, and K. Shaban, "Network function virtualization scheduling with transmission delay optimization," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2016, pp. 638–644.

[6] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015.

[7] J. Lee, H. Ko, D. Suh, S. Jang, and S. Pack, "Overload and failure management in service function chaining," in *IEEE Conference on Network Softwarization (NetSoft)*, 2017, pp. 1–5.

[8] "Fail-slow at scale: When the cloud stops working," https://www.zdnet.com/article/how-clouds-fail-slow/, accessed: 2018-07-10.

[9] "ETSI GS NFV-REL 001 V1.1.1 (2015-01), Network Functions Virtualisation (NFV) Resiliency Requirements," https://goo.gl/PbQySQ, accessed: 2018-07-10.

[10] Q. Zhang, M. F. Zhani, M. Jabri, and R. Boutaba, "Venice: Reliable virtual data center embedding in clouds," *IEEE International Conference on Computer Communications (INFOCOM)*, April 27 - Mai 2 2014.

[11] "Downtime, outages and failures - understanding their true costs," https://www.evolven.com/blog/downtime-outages-and-failures-understanding-their-true-costs.html , accessed: 2018-07-13.

[12] M. F. Zhani and R. Boutaba, *Survivability and Fault Tolerance in the Cloud*. John Wiley & Sons, Inc, 2015, pp. 295–308. [Online]. Available: http://dx.doi.org/10.1002/9781119042655.ch12

[13] H. Yu, V. Anand, C. Qiao, and G. Sun, "Cost efficient design of survivable virtual infrastructure to recover from facility node failures," in *IEEE International Conference on Communications (ICC)*, 2011, pp. 1–6.

[14] S. Ayoubi, Y. Chen, and C. Assi, "Towards promoting backup-sharing in survivable virtual network design," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 3218–3231, 2016.

[15] M. R. Rahman and R. Boutaba, "SVNE: Survivable virtual network embedding algorithms for network virtualization," *IEEE Transactions on Network and Service Management*, vol. 10, no. 2, pp. 105–118, 2013.

[16] M. G. Rabbani, M. F. Zhani, and R. Boutaba, "On achieving high survivability in virtualized data centers," *IEICE Transactions on Communications*, vol. E97-B, no. 1, January 2014.

[17] A. Xiao, Y. Wang, L. Meng, X. Qiu, and W. Li, "Topology-aware virtual network embedding to survive multiple node failures," in *IEEE Global Communications Conference (GLOBECOM)*, 2014, pp. 1823–1828.

[18] L. Bo, T. Huang, X.-c. SUN, J.-y. CHEN, and Y.-j. LIU, "Dynamic recovery for survivable virtual network embedding," *The Journal of China Universities of Posts and Telecommunications*, vol. 21, no. 3, pp. 77–84, 2014.

[19] R. Boutaba, Q. Zhang, and M. F. Zhani, "Virtual machine migration: Benefits, challenges and approaches," in *Communication Infrastructures for Cloud Computing: Design and Applications*, H. T. Mouftah and B. Kantarci, Eds. USA: IGI-Global, 2013, pp. 383–408.

[20] B. Guo, C. Qiao, J. Wang, H. Yu, Y. Zuo, J. Li, Z. Chen, and Y. He, "Survivable virtual network design and embedding to survive a facility node failure," *IEEE Journal of Lightwave Technology*, vol. 32, no. 3, pp. 483–493, 2014.

[21] S. Ayoubi, C. Assi, L. Narayanan, and K. Shaban, "Optimal polynomial time algorithm for restoring multicast cloud services," *IEEE Communications Letters*, vol. 20, no. 8, pp. 1543–1546, 2016.

[22] A. M. Ghaleb, T. Khalifa, S. Ayoubi, K. B. Shaban, and C. Assi, "Surviving multiple failures in multicast virtual networks with virtual machines migration," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 899–912, 2016.