Availability-driven NFV Orchestration

Marco Casazza[a,*], Mathieu Bouet[c], Stefano Secci[b]

[a] *Università degli Studi di Milano, Dipartimento di Informatica, Crema, Italy*
[b] *Cnam, Paris, France*
[c] *Thales, France*

Abstract

Virtual Network Functions as a Service (VNFaaS) is a promising business whose technical directions consist of providing network functions as a Service instead of delivering standalone network appliances, leveraging a virtualized environment named NFV Infrastructure (NFVI) to provide higher scalability and reduce maintenance costs. Operating the NFVI under stringent availability guarantees is fundamental to ensure the proper functioning of the VNFaaS against software attacks and failures, as well as common physical device failures. Indeed the availability of a VNFaaS relies on the failure rate of its single components, namely the physical servers, the hypervisor, the VNF software, and the communication network.

In this paper, we propose a versatile orchestration model able to integrate an elastic VNF protection strategy with the goal to maximize the availability of an NFVI system serving multiple VNF demands. The elasticity derives from (i) the ability to use VNF protection only if needed, or (ii) to pass from dedicated protection scheme to shared VNF protection scheme when needed for a subset of the VNFs, (iii) to integrate traffic split and load-balancing as well as mastership role election in the orchestration decision, (iv) to adjust the placement of VNF masters and slaves based on the availability of the different system and network components involved. We propose a VNF orchestration algorithm based on Variable Neighboring Search, able to integrate both protection schemes in a scalable way and capable to scale, while outperforming standard online policies.

Keywords: High Availability NFV, Virtual Network Functions, NFV Orchestration, Variable Neighborhood Search, Greedy Heuristic.

1. Background

A recent trend in computer networks and cloud computing is to virtualize network functions, in order to provide higher scalability, to reduce maintenance costs, and to increase the reliability of network services. Virtual Network Functions as a Service (VNFaaS) is currently under attentive study by telecommunication and cloud stakeholders, as a promising business and technical direction consisting of providing network functions (i.e., firewall, intrusion detection, caching, gateways...) as a Service instead of delivering standalone network appliances. While legacy network services are usually implemented by means of highly reliable hardware specifically built for a single purpose middlebox, VNFaaS moves such services to a virtualized environment [1], named *NFV Infrastructure (NFVI)*, based on commercial-off-the-shelf hardware [2].

Services implementing network functions are called *Virtual Network Functions (VNFs)*. One of the key points differentiating NFVI design from classical Cloud infrastructure design is the requirement to guarantee high levels of availability to the service [3], i.e., the probability that the network function is working at a given time. In other words, a higher availability corresponds to a smaller downtime of the system, and it is required to satisfy stringent *Service Level Agreements (SLA)*. While in common IT/cloud systems the expectations in terms of high availability are in the order of 99,9% availability, telco/NFV carrier grade systems expectations are in the order of 99,999% availability, and even beyond for mission/business critical services [4].

While failures are tolerated in other contexts, the interruption of NFVI services is not acceptable, since the failure of a single VNF has a cascade effect on all the overlying services [5]. To achieve high availability, backup VNFs can be placed into the NFVI, acting as replicas of the running VNFs, so that when one of the latter goes down, the load is rerouted to the former. However, not all VNFs are equal ones: the software implementing a network function of the server where a VNF is running may be more prone to errors than others, influencing the availability of the overall infrastructure. Also, client requests may be routed via different network paths, with different availability performance. Therefore to guarantee high levels of availability it is important not only to increase the number of VNF replicas placed on an NFVI, but it is also crucial to select where they are placed and which requests they serve. A level of flexibility in the VNF placement and protection strategies is needed, such that VNF protection is adopted only when needed, that a stronger protection scheme can be adopted to compensate low component/network availability for some VNFs, and that no protection may be acceptable for some other VNFs, for a set of VNFs demanded by different users beyond different network

*Corresponding author
Email addresses: marco.casazza@unimi.it (Marco Casazza), mathieu.bouet@thalesgroup.com (Mathieu Bouet), stefano.secci@cnam.fr (Stefano Secci)

gateways, concurrently.

In this context, we propose and evaluate an NFVI orchestration algorithmic framework that is able to offer such a flexibility. More precisely, our contribution consists of:

(i) a quantitative probabilistic model to measure the expected availability of VNF placement;

(ii) a formalization of the orchestration problem, with a proof of its $\mathcal{NP}$-hardness, along with a linear mathematical programming formulation that can be solved to optimality for instances of limited size;

(iii) a *Variable Neighborhood Search (VNS)* heuristic, with controllable complexity and hence suitable for both online and offline orchestration;

(iv) an extensive multi-facet evaluation, including a simulation campaign, algorithm integration in a Decision Support System tool and in an OpenStack testbed [6].

The paper is organized as follows: in Section 2 we start by quickly defining the optimization framework, the VNF protection strategies and their impact on NFVI operations. In Section 3 we briefly describe previous works on VM/VNF placement for cloud/NFVI systems. In Section 4 we formally describe the optimization problem and propose a linearization of the problem and a mathematical programming formulation that can solve it to optimality. In Section 5 we describe our heuristic methodologies, which are then tested in an extensive simulation campaign in Section 7. Section 8 describes how we could integrate our algorithm in an OpenStack testbed. We briefly conclude the paper in Section 9.

A preliminary version of the content of this paper was presented at the 2017 IFIP Networking conference [7].

2. High Availability NFV Orchestration

In this section we describe the overall VNF orchestration problem that we address. We also propose a taxonomy of the VNF protection strategies which, to the best of our knowledge, is a unique contribution to the state of the art on NFV orchestration strategies.

2.1. Networking context

We consider an NFVI with several geo-distributed datacenters or *clusters* (see Fig. 1). Each cluster consists of a set of heterogeneous servers with limited available computing resources. Several instances of the same VNF type can be placed on the NFVI but on different servers. Each VNF instance can be assigned to a single server allocating some of its computing resources. Indeed each server has a limited amount of computing resources that cannot be exceeded.

A network connects together all servers of the NFVI: we suppose that the communication links inside a cluster are significantly more reliable than those between servers in different clusters. An access network with multiple access/gateway points connects VNFs to users. Links connecting access points

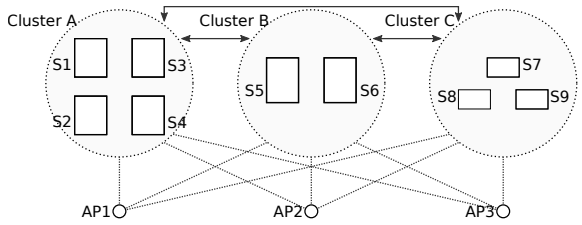


Figure 1: Abstract representation of an NFVI: each server is depicted as a white box whose height represents the amount of available resources. Clusters are connected together to allow synchronization operations. Access network is represented by access points connecting clusters to the external network.

to clusters can differ in the availability level, depending on the type of the connection or the distance from the cluster.

In this article, we assume that the total amount of resources and network capacities are sufficient to manage the expected client requests at any time. However assignment decisions may artificially produce congestion over the servers. We analyze how to find assignments providing a trade-off between NFVI availability and system congestion.

We are given an estimation of the expected client VNF requests, each characterized by a computing resource demand. An assigned request consumes part of the resources reserved by a VNF instance. Indeed the consumed resources must not exceed the reserved ones.

Requests can be assigned using two different policies:

- *demand load balancing*: a client request is always fully assigned to a single server, and multiple demands of the same VNF can be balanced to different servers;
- *split demand load balancing*: a client request may be split among different servers. Splitting a request also splits proportionally its demand of computing resources. Indeed, when a demand is split it relies on the availabilities of many VNF instances, decreasing the expected availability of the service, but increasing the chance of finding a feasible assignment in case of congestion.

In the orchestration model, we do not address the VNF chaining dimension, and hence the related network embedding problem. Instead, we focus on the effective placement of multiple VNFs that compete for the highest availability through replication, in an infrastructure with limited resources.

2.2. VNF protection strategies

We suppose a multi-failure environment in which VNFs, servers, clusters, and networks may fail together. To improve the VNF availability, VNF instances are replicated on the NFVI. We distinguish between *master* and *slave* VNFs: the former are active VNF instances, while the latter are idle until masters fail. In case of multiple active master VNFs, one of the load-balancing policies can be used. Masters entertain with slaves a synchronization link, to allow state updates, which is particularly appropriate for stateful VNF configurations[1]. An example

[1] A stateful VNF is a VNF that embeds the states needed to its operation within the VNF system. A stateless VNF, instead, externalizes the states so that

of VNF placement with demand load-balancing is depicted in Fig. 2.

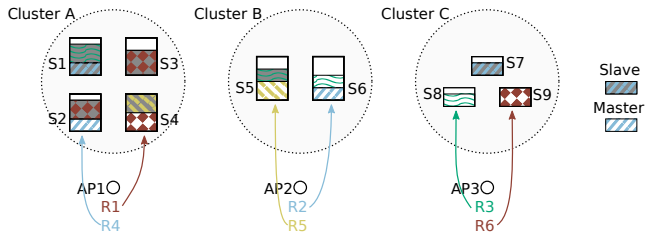
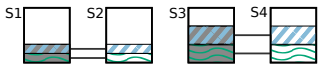


Figure 2: Abstract representation of VNFs placement on a 3-cluster NFVI. Each patterned box is a VNF instance. Instances running the same VNF type have the same pattern. VNFs with a gray background are slaves placed as protection for the masters. Sets of requests are routed from access points and assigned to VNFs running the requested function.

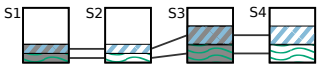
Each master may be *protected* by many slaves, but each slave must be placed on a different server than its master and must be allocated at least the same amount of computing resources. Each master periodically saves and sends its state to its slaves, e.g. using technologies such as the one presented in [9], in such a way that the latter have always an updated state and can consistently restore the computation.

In this paper, we present and compare two different VNF protection schemes (represented with an example in Fig. 3):

- *dedicated VNF protection*: a slave protects only one master (*dedicated* slave): in case such a master fails, if the slave is available it can always restore the master state;
- *shared VNF protection*: a slave protects at least one master (*shared* slave): when a master fails, a shared slave can restore its state only if it is available and it is has not restored another master state yet.



(a) In dedicated protection each slave protects at most one master. In this example each master has a single slave.



(b) In shared protection a slave can protect many masters. In this example masters on server $S2$ have 2 slaves. Slaves on server $S1$ cannot protect masters on server $S4$ because they lack of allocated computing resources.

Figure 3: Example of protection schemes: master VNFs are running on servers $S2$ and $S4$, while slave ones are running on $S1$ and $S3$. Each link between VNF instances represents the connection between a master and its slave.

Table 1 gives a summary of the different possible scenarios we aim to cover with our orchestration framework, depending on the load-balancing strategy and the VNF protection strategy. Given a set of demands, potentially any of the six possible configurations could manifest for a given demand, and different demands could fall under a different scenario.

they can be retrieved from an external database system, ideally a distributed one [8].

We suppose that if a master is unreachable, a recovery process either has already cached the location information of the slave at network switches, or retrieves it from the control-plane. In case of multiple slaves, the dedicated slaves are selected first. If all slaves are unreachable or already restoring another master, then the service is considered unavailable.

About the shared protection option, we can further differentiate between:

- *homogeneous shared protection*: a slave VNF can be shared only by master VNFs of the same VNF type. This implies that both stateful VNFs and stateless VFNs could be used.
- *heterogeneous shared protection*: a slave VNF can be shared by master VNFs of different VNF types. This necessarily implies that a context change process is able to select which protection VNF from a local file system to execute, and that to minimize the impact of the recovery time on VNF operations, stateless VNFs would be highly desirable.

Fig. 4 is a Venn diagram representation of the solution space. Feasible solutions with dedicated protection can be reached by enriching a feasible placement with no protection. Similarly does for homogeneous shared protection with respect to dedicated protection, and heterogeneous shared protection with respect to homogeneous protection.

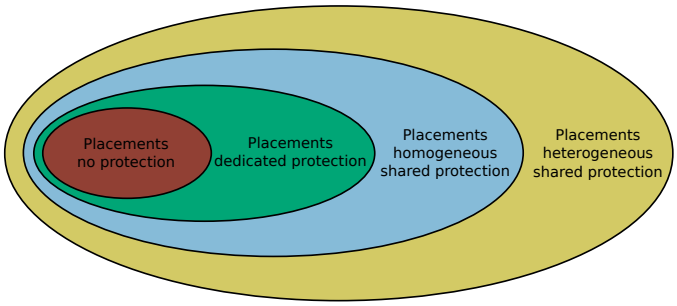
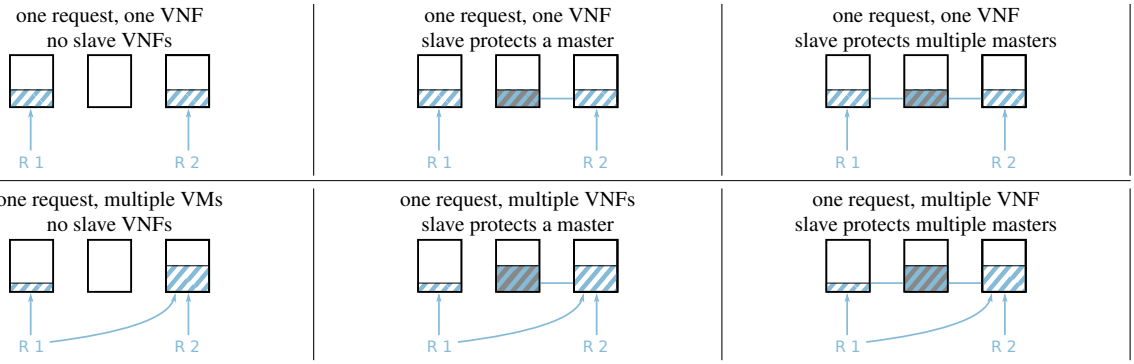


Figure 4: Venn diagram representing the relations between feasible placements of VNFs for each protection scheme.

It is important to note that, if one seeks high availability, with the VM booting performance of current hypervisors, homogeneous shared protection is certainly more appropriate than the heterogeneous alternative: indeed, running heterogeneous shared protection implies that the slave VNF is booted on demand, which implies a non negligible switchover time (or mean-time-to-repair) in the order of seconds, which would drastically decrease the availability performance. In case of homogeneous protection, the slave VNF can be kept running so that the handover time can be kept at the ms level (i.e., essentially given by the re-mapping time), with therefore negligible impact on the recovery time. Therefore we consider in the following only homogeneous shared protection.

Table 1: Summary of VNF placement scenarios.

request assignment	protection scheme: no protection	protection scheme: dedicated protection	protection scheme: shared protection
demand load balancing	one request, one VNF no slave VNFs	one request, one VNF slave protects a master	one request, one VNF slave protects multiple masters
split demand load balancing	one request, multiple VMs no slave VNFs	one request, multiple VNFs slave protects a master	one request, multiple VNF slave protects multiple masters



3. Related works

Although there already exist orchestrators that are driven by optimization algorithms for the placement, such as [10], the management of cloud computing resources for VM and VNF placement is still a recent area of research (see [11] for a high-level comprehensive study). We now present few works in the literature studying the optimization problems that arise in this context.

Placement of Virtual Machines. [12] studies the problem of placing VMs in datacenters minimizing the average latency of VM-to-VM communications. Such a problem is $\mathcal{NP}$-hard and falls into the category of *Quadratic Assignment Problems*. The authors provide a polynomial time heuristic algorithm solving the problem in a *"divide et impera"* fashion. In [13] the authors deal with the problem of placing VMs in geo-distributed clouds minimizing the inter-VM communication delays. They decompose the problem in subproblems that they solve heuristically. They also prove that, under certain conditions, one of the subproblems can be solved to optimality in polynomial time. [14] studies the VM placement problem minimizing the maximum ratio of the demand and the capacity across all cuts in the network, in order to absorb unpredictable traffic bursts. The authors provide two different heuristics to solve the problem in reasonable computing time.

Placement of Virtual Network Functions. [15] applies NFV to LTE mobile core gateways proposing the problem of placing VNFs in datacenters satisfying all client requests and latency constraints while minimizing the overall network load. Instead, in [16] the objective function requires to minimize the total system cost, comprising the setup and link costs. [17] introduces the VNF orchestration problem of placing VNFs and routing client requests through a chain of VNFs. The authors minimize the setup costs while satisfying all client demands. They propose both an Integer Linear Programming model (ILP) and a heuristic to solve such problem. Also [18] considers the VNF orchestration problem with VNF switching piece-wise linear latency function and bit-rate compression and decompression operations. Two different objective functions are studied: one minimizing costs and one balancing the network usage. In [19] replicas of VNFs are placed in the networks to increase load balancing and minimize the usage of the network. The authors propose an ILP to solve the optimization problem with optimality guarantees, a genetic algorithm, and a fast greedy algorithm. In [20] the authors consider the problem of allocating the minimum quantity of resources to perform the VNF placement in a two-level hierarchical geo-distributed infrastructure, proposing both an ILP and a Tabu Search heuristic algorithm. In [21] the usage of both computing and communication resources are balanced into a bi-objective optimization problem, which is proved to be $\mathcal{NP}$-hard. The authors, propose an ILP together with a greedy heuristic: each service chain is sequentially deployed over network nodes, in such a way that the nodes causing less resource usage increment are the ones having higher probability of being chosen. Such a probability is computed through a Hidden Markov Model. In [22] the authors take into account the time-variance of the workload of each function in the placement problem. In such a way the utilization efficiency of resources is improved and costs are minimized. The authors propose an ILP working for small number of service chains, and a two-stage heuristic algorithm that first assigns service chains to physical servers in a greedy fashion starting from the least correlated chains, and then tries to reduce the number of physical servers by collecting together requests of the same type and freeing computing resources from servers.

Placement with availability constraints. In [23] VMs are placed with a protection guaranteeing k-resiliency, that is at least k slaves for each VM. The authors propose an integer formulation that they solve by means of constraint programming. In [24] the recovery problem of a cloud system is considered where slaves are usually turned off to reduce energy consumption but can be turned on in advance to reduce the recovery time. The authors propose a bicriteria approximation algorithm and a greedy heuristic. In [25] the authors solve a problem where links connecting datacenters may fail, and a star connection between VMs must be found minimizing the probability of failure. The authors propose an exact and a greedy algorithm to solve both

small and large instances, respectively. Within disaster-resilient VM placement, [26] proposes a protection scheme in which for each master a slave is selected on a different datacenter, enforcing also path protection. In [27] the authors solve the problem of placing slaves for a given set of master VMs without exceeding neither servers nor link capacities. Their heuristic approaches decompose the problems in two parts: the first allocating slaves, and the second defining protection relationships.

In a recent work, VNF chaining under path-level and link-level protection strategies is studied in [28], with a particular emphasis on the network embedding, excluding the modeling of different VNF node protection strategies and of the VNF state synchronization link. In another recent work [29], the authors model the VM availability by means of a probabilistic approach and solve the placement problem over a set of servers by means of a nonlinear mathematical formulation and greedy heuristics. This is the only work offering a comprehensive estimation of the availability of the system. However, it considers only the availability of the servers, while in our problem we address a more generic scenario: when datacenters are geo-distributed, a client request shall be assigned to the closest datacenter, since longer connections may have a higher failure rate. Therefore, the source of the client requests may affect the placement of the VNFs on the NFVI, and must be taken into account in the optimization process and in the estimation of the availability.

In [30, 31] the authors consider the placement of VNF service chains guaranteeing resiliency against both single-node and single-link failure. Three different protection schemes are proposed and compared by solving the corresponding ILPs. In [32] instead, VNFs are placed in such a way that, for each service chain, availability and delay SLA constraints are met. Resource protection is not addressed and availability constraints are met by assigning more reliable links and network nodes to service chains requiring higher level of availability. The authors propose both an ILP model to solve the problem with optimality guarantees and an efficient greedy heuristic. In [33] a bi-objective VM placement problem is considered where cost and availability are balanced. The authors provide a two-phase heuristic that first distributes resources to VMs and then places VMs to physical servers.

4. Network Orchestration Model

In the following we propose a formal definition to the High Availability Virtual Network Function Placement Problem (HA-VNFP), and a mathematical programming formulation.

Clusters and servers. We are given the set of clusters C and the set of servers S. Each server s belongs to a cluster c_s, and we define as $S_c \subseteq S$ the set of servers of cluster c. We represent the usual distinct types of computing resources (CPU, RAM, ...) of server $s \in S$ by the same global amount $q_s \in \mathbb{R}_+$ of available resources.

Virtual Network Functions. A set F of VNF types is given. Each VNF instance runs on a single server. Each server can host multiple VNF instances, but at most one master for each type.

Networks. An inter-cluster network allows synchronization between clusters, while an access network connects clusters to a set of access points P. We are given sets L_C and L_P of logical links $(c', c'') \in L_C$ connecting clusters $c', c'' \in C$, and logical links $(c, p) \in L_P$ connecting cluster $c \in C$ to access point $p \in P$, respectively.

Client requests. A set of client requests R is given. Each request $r \in R$ is a tuple (f_r, P_r, d_r) of the requested VNF type $f_r \in F$, a subset of available access points $P_r \subseteq P$, and the resources demand $d_r \in \mathbb{R}_+$.

Availability. Taking into account explicit availability in NFVI design becomes necessary to ensure SLAs [5, 2]. We suppose that the availabilities of each component (server, cluster, VNF, link) are given (see Table 2), each corresponding to the probability that a component is working.

Objective function. All client requests must be assigned to servers maximizing the availability of the system, which we measure as the minimum availability among all requests. We consider the availability as the most important and sole objective: in mission critical systems other criteria are overshadowed and can be considered as constraints, such as the available budget or the amount of computing resources.

4.1. Computational complexity

We now investigate the computational complexity of our optimization problem starting from the computational effort required to find a feasible solution. We prove that:

Observation 1. *When demand split is allowed and $\sum_{r \in R} d_r \leq \sum_{s \in S} q_s$, HA-VNFP has always a feasible solution that can be found in polynomial time.*

In fact since the requests can be split among servers, the feasibility of an instance can be found applying a Next-Fit greedy algorithm for the Bin Packing Problem with Item Fragmentation (BPPIF) [34]: servers can be seen as bins, while requests as items that must be packed into bins. The algorithm iteratively packs items to an open bin. When there is not enough residual capacity, the item is split, the bin is filled and closed, and a new bin is open packing the rest of the item. When requests can be split, such algorithm produces a feasible solution for the HA-VNFP: if a request is assigned to a server, then a master VNF serving such a request is allocated on that server too. The Next-Fit algorithm runs in $O(|R|)$ and therefore a feasible solution can be found in polynomial time.

Observation 2. *The feasibility of a HA-VNFP instance without demand split is a $\mathcal{NP}$-hard problem.*

Indeed we can see again the feasibility problem as a Bin Packing Problem (BPP). However, without split each item must be fully packed into a single bin. Therefore, finding a feasible solution is equivalent to the feasibility of a BPP, which is $\mathcal{NP}$-hard. It follows that for what concerns the computational complexity of the HA-VNFP, that is the computational effort required to find the placement having maximum availability, we have that:

Table 2: Mathematical notation.

C	Set of clusters	S	Set of servers
S_c	Set of servers in cluster $c \in C$	F	Set of VNF types
R	Set of requests	P	Set of access points
L_C	Set of synchronization links	L_P	Set of access links
P_r	Set of access points for request $r \in R$		
q_s	Capacity of server $s \in S$	d_r	Demand of request $r \in R$
c_s	Cluster containing server $s \in S$	f_r	VNF type of request $r \in R$
a_f^F	Availability of VNF type $f \in F$	a_s^S	Availability of server $s \in S$
$a_{cc'}^{L_C}$	Availability of synchronization link (c, c'), with $c, c' \in C$	$a_{cp}^{L_P}$	Availability of access link (c, p), with $c \in C$ and $p \in P$
a_c^C	Availability of cluster $c \in C$		

Theorem 1. *The HA-VNFP without demand split is $\mathcal{NP}$-hard.*

That is, for unsplittable demands, it is $\mathcal{NP}$-hard finding both a feasible solution and the optimum solution. It is less straightforward to also prove that:

Theorem 2. *The HA-VNFP with demand split is $\mathcal{NP}$-hard.*

Proof. In fact, let us suppose a simple instance where all components (servers, clusters, links, ...) are equal ones and where $\sum_{r \in R} d_r = \sum_{s \in S} q_s$, which means that there will be no slaves in our placement. The problem can be seen again as a BPPIF in which the objective is to minimize the number of splits of the item that is split the most: in fact, every time a request is split, the availability of the system decreases. In such scenarios the best solution is the one in which no request is split at all - however, if we could solve such a problem in polynomial time, then we could solve also the feasibility problem of a BPP in polynomial time, which instead is $\mathcal{NP}$-hard. Therefore, since we can reduce a feasibility problem of BPP to an instance of BPPIF, and the latter to an instance of HA-VNFP, the HA-VNFP with split is $\mathcal{NP}$-hard. □

4.2. Mathematical formulation

In the following we provide a mathematical programming formulation of HA-VNFP with dedicated protection, starting from the definition of the set of the solutions: a *request assignment* ω is a pair (s, S_p) indicating the subset of servers $S_p \subseteq S$ running either the master or the slaves of a VNF instance, and the server $s \in S_p$ where the master is placed. We also define

$$\Omega = \{(s, S_p) \mid S_p \subseteq S, s \in S_p\}$$

as the set of all request assignments. An *assignment configuration* γ (see Fig. 5) is a set of all request assignments ω for all the fragments of a request. We define as Γ the set of all assignment configurations γ, that is

$$\Gamma = \{\gamma \in 2^{\Omega} \mid s' \neq s'', \forall (s', \omega'), (s'', \omega'') \in \gamma\}.$$

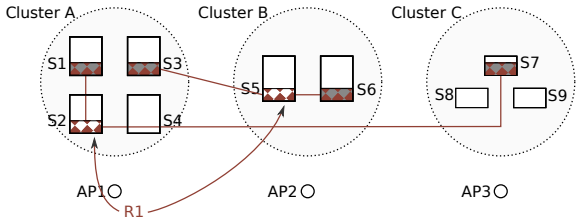


Figure 5: Example of assignment configuration $\gamma = \{(S2, \{S1, S2, S7\}), (S5, \{S3, S5, S6\})\}$, where request $R1$ is split and assigned to two different master VNFs on servers $S2$ and $S5$. Both masters have slaves: the master VNF on server $S2$ has slaves on servers $S1$ and $S7$, while the one on server $S5$ has slaves on servers $S3$ and $S6$.

Availability computation. We compute the NFVI availability for a request r by means of a probabilistic approach [35, 36]. Given a cluster and a set of access points, $a^{L_P}(c, P)$ is the function computing the probability that at least one of the access links is working: $a^{L_P}(c, P) = 1 - \prod_{p \in P}(1 - a_{cp}^{L_P})$. Given a VNF and a set of servers, $a^S(f, S)$ is the function computing the probability that at least one instance of VNF is working: $a^S(f, S) = 1 - \prod_{s \in S}(1 - a_f^F \cdot a_s^S)$. Given a request r and a request assignment $\omega = (s, S_p)$, $a(r, \omega)$ is the function computing the probability that at least one of the instances of ω is working:

$$a(r, \omega) = 1 - [\ \underbrace{(1 - a^{L_P}(c_s, P_r) \cdot a_{c_s}^C \cdot a^S(f_r, S_p \cap S_{c_s}))}_{\text{availability of the cluster containing master}} \cdot$$

$$\cdot \prod_{c \in C \setminus \{c_s\}} \underbrace{(1 - a^{L_P}(c, P_r) \cdot a_c^C \cdot a_{c_s c}^{L_C} \cdot a^S(f_r, S_p \cap S_c))}_{\text{availability of cluster containing only slaves}}\ ]$$

When a request r is split, we compute its availability $a(r, \gamma)$ as the probability that all of its parts succeed:

$$a(r, \gamma) = \prod_{(s,\omega) \in \gamma} a(r, \omega). \tag{1}$$

We remark that such formula is nonlinear and produces an Integer Nonlinear Programming formulation which cannot be solved by common integer solvers like CPLEX. Therefore we propose a Mixed Integer Programming (MIP) linearization of such nonlinear formulation in which for each assignment configuration $\gamma \in \Gamma$ we have a binary variable stating if such configuration is selected in the solution.

Variables. The following variables are needed:

$$x_{rs} : \text{fraction of request } r \text{ assigned to server } s$$
$$z_{r\gamma} = \begin{cases} 1, \text{ if } \gamma \text{ is active for request } r \\ 0, \text{ otherwise} \end{cases}$$
$$u_{fs} : \text{resources consumed by VNF } f \text{ on } s$$
$$v_{fss'} : \text{resources consumed by slave on server } s'$$
$$\mathcal{A}_{min} : \text{minimum availability}$$

Model. HA-VNFP can be modeled as follows:

$$\max \mathcal{A}_{min} \tag{2}$$
$$\text{s.t.} \sum_{s \in S} x_{rs} = 1 \qquad \forall r \in R \tag{3}$$
$$x_{rs} - \sum_{\substack{\gamma \in \Gamma \\ \exists (s,\omega) \in \gamma}} z_{r\gamma} \leq 0 \qquad \forall r \in R, s \in S \tag{4}$$
$$\sum_{\substack{r \in R \\ f_r = f}} d_r \cdot x_{rs} - u_{fs} \leq 0 \qquad \forall f \in F, s \in S \tag{5}$$
$$u_{f_r s} + q_s \cdot \sum_{\substack{\gamma \in \Gamma \\ \exists (s,\omega) \in \gamma | s' \in \omega}} z_{r\gamma} - v_{f_r ss'} \leq q_s \qquad \forall r \in R, s, s' \in S \tag{6}$$
$$\sum_{f \in F} u_{fs} + \sum_{s' \in S} v_{fs's} \leq q_s \qquad \forall s \in S \tag{7}$$
$$\sum_{\gamma \in \Gamma} z_{r\gamma} \leq 1 \qquad \forall r \in R \tag{8}$$
$$\sum_{\gamma \in \Gamma} a(r, \gamma) \cdot z_{r\gamma} - \mathcal{A}_{min} \geq 0 \qquad \forall r \in R \tag{9}$$

Constraints (3) and (4) ensure that each request is fully assigned and selects an assignment configuration, respectively. Constraints (5) and (6) set the allocated resources of masters and slaves, respectively. Constraints (7) ensure that servers capacities are not exceeded. Constraints (8) impose that at most one assignment configuration is selected for each request. Constraints (9) compute the minimum availability. It is worth noting that, in practice, an additional constraint can be added to set an upper bound to the minimum availability, hence avoiding near-full usage of the computing capacity. Our formulation can model both the HA-VNFP with and without split: in fact by simply setting $|\gamma| = 1$ for each configuration γ we forbid configurations splitting a request.

4.3. Modeling shared protection

Shared protection cannot be integrated in our mathematical programming formulation: in fact, the computation of the availability of a given placement using shared protection is already challenging and requires advanced methodologies such as Markov Chains or Bayesian Networks, which unfortunately run in exponential time. Therefore we chose to provide a methodology to obtain both lower and upper bounds to the availability of a VNF placement with shared protection.

Let us be given a placement of VNFs, such that for each selected request assignment $\omega = (s, S_p)$, we can split the set $S_p \setminus \{s\}$ in two sets: the set S_p^- of dedicated slaves, and the set S_p^+ of shared slaves. Let us be given also a function $a_{shared}(r, \omega)$ computing the availability of a placement using shared protection.

Observation 3. *Given a request assignment $\omega = (s, S_p)$ and a request $r \in R$, it holds that*

$$a(r, (s, S_p \setminus S_p^+)) \leq a_{shared}(r, \omega) \leq a(r, \omega)$$

In fact, when a master fails, we may have two extreme cases with regard to shared slaves: in the worst case all shared slaves are already restoring the state of others protected masters, while in the best case all shared slaves are available. In the worst case, the only servers available for protection are the ones in S_p^-, which however are dedicated slaves. Therefore, the availability in the worst case is at least the availability of the dedicated protection scenario with dedicated slaves only, and $a(r, (s, S_p \setminus S_p^+)) \leq a_{shared}(r, \omega)$.

Similarly, in the best case each server in S_p can be chosen to restore the master state, as they are all dedicated slaves. Therefore, in this case the availability is at most the availability of the same request assignment with all slaves and dedicated protection, and $a_{shared}(r, \omega) \leq a(r, \omega)$.

Since the availability of an assignment configuration is given by the product of all its request assignments, it follows that:

Proposition 1. *The availability of a placement computed using dedicated slaves only is a lower bound to the availability of the same placement with shared protection.*

Proposition 2. *The availability of a placement computed using all slaves in a dedicated protection fashion is an upper bound to the availability of the same placement with shared protection.*

5. Resolution algorithms

Solving HA-VNFP as a MIP using an integer solver works only for small NFVI, since the number of variables grows exponentially in the number of servers. Therefore we propose two different heuristic approaches for HA-VNFP: the first is an adaptation of well-known greedy policies for the BPP that will serve as comparison, while the second is a *Variable Neighborhood Search* heuristic using different algorithmic operators to explore the neighborhood of a starting point.

5.1. Greedy heuristics

Most of the heuristics for the placement of VMs or VNFs are based on a greedy approach, and BPP heuristics are often exploited to obtain suitable algorithms for the placement, such as in [37]. We also exploit BPP heuristics to obtain three different greedy approaches for the HA-VNFP: *Best Availability*, *Best Fit*, and *First Fit* greedy heuristics. The algorithm, reported in Algorithm 1, starts from an empty initial placement and for each request r it looks for a server having enough residual capacity to satisfy the demand d_r. If such a server is found, then the request is assigned to it, otherwise the algorithm fails without finding a feasible solution. However, we can observe that:

Observation 4. *When $\sum_{r\in R} d_r \leq \sum_{s\in S} q_s$ and split is allowed, our greedy heuristic always finds a feasible solution.*

In fact we can always split a request between two servers, as stated also in Observation 1.

The selection of the server is performed by the procedure SELECTSERVER($\bar{S}, \bar{d}, split$) which discards the servers without sufficient resources to satisfy demand $\bar{d}$, and selects a server depending on the chosen policy:

- *best fit*: the server whose capacity best fits the demand;
- *first fit*: the first server found;
- *best availability*: the server with the highest availability.

While the first two policies are well-know for the BPP, the third one is designed for the HA-VNFP.

Master VNFs are placed during the assignment of the requests. Then, in a similar way, the algorithm places additional slaves: for each master the algorithm looks for a server having enough capacity for a slave still using SELECTSERVER procedure. After a server is found, the slave is placed. Such a procedure is repeated until no additional slave is placed.

Although we described demand load balancing and split demand load balancing as two different policies, it is easy to mix them in our algorithms by simply setting the parameter *split* of procedure SELECTSERVER($\bar{S}, \bar{d}, split$) to *true* or $false$ depending on whether the corresponding VNF can be split or not, respectively.

Algorithm 1 Greedy heuristic procedure

```
1: function GREEDY(R, S, split)
2:   placement ← ∅
3:   for all r ∈ R | d_r > 0 do                ▷ Assignment of requests
4:     if ∃ŝ ←SELECTSERVER(d_r, S, split) then
5:       create VNF f_r if it does not exists in placement
6:       assign request r to server ŝ in placement
7:       demand d_r is decreased
8:     else
9:       return infeasible
10:    end if
11:  end for
12:  do                                          ▷ Add slaves
13:    for all VNFs v ∈ placement do
14:      S̄ ← servers of S without v and its slaves
15:      if ∃ŝ ←SELECTSERVER(d_v, S̄, FALSE) then
16:        create slave of VNF v on server ŝ in placement
17:      end if
18:    end for
19:  while slaves are found
20:  return placement
21: end function
```

5.2. Variable Neighborhood Search

The *Variable Neighborhood Search (VNS)* is a meta-heuristic that systematically changes the neighborhood within the local search algorithm, in order to escape from local optima. In other words, it starts from an initial solution, applies a local search algorithm until it improves, and then changes the type of local search algorithm applied to change the neighborhood. Our VNS algorithm explores 4 different neighborhoods and it is initialized with several starting points, each obtained using a different greedy algorithm.

Algorithm 2 Variable Neighborhood Search

```
1: function VNS(R, S, split)
2:   startingPoints ← {BESTAVAILABILITY(R, S, split),
     BESTFIT(R, S, split), FIRSTFIT(R, S, split) }
3:   operators ← {vnfSwap, slaveSwap, requestSwap,
     requestMove}
4:   bestPlacement ← ∅
5:   for all placement ∈ startingPoints do
6:     do
7:       for all op ∈ operators do
8:         placement ← apply op to placement
9:         if placement improves bestPlacement then
10:          bestPlacement ← placement
11:          break
12:        end if
13:      end for
14:    while improving bestPlacement
15:  end for
16:  return bestPlacement
17: end function
```

The main logic of our VNS algorithm is sketched in Algorithm 2: we generate 3 starting points by using the greedy heuristics of Section 5.1 and we explore their neighborhood for a placement improving the availability. If no improvement can be found, the algorithm switches the neighborhood.

Indeed applying local search is time expensive, but we can observe that a max-min objective function divides the requests in two sets: a set of requests having an availability equal to the objective function and another set having a better availability. We refer to the former as the set of the *worst requests*, since they are the ones whose improvement will also improve the availability of the entire solution. To reduce the computing time and focus our algorithm we found to be profitable to restrict the explored neighborhood to the worst requests only. Also, after applying each operator we look for new slaves, as in the greedy procedure Algorithm 1. Given two feasible placements, there is an improvement if one has a higher availability, or the same availability but fewer worst requests.

In the following we describe the neighborhoods of our VNS.

VNFs swap. The first neighborhood consists of swapping VNFs (see Fig. 6): given a VNF, we swap it with a subset of VNFs deployed on a different server. If the placement is improved, then we store the result as the best local change.

In general our operator is $O(2^{|F|\cdot|S|})$ but we found profitable to set an upper bound of 1 to the cardinality of the set of swapped VNFs, obtaining a $O(|F| \cdot |S|)$ operator.

Slave VNFs swap. We explore the neighborhood where a slave VNF is removed to free resources for an additional slave of a different master VNF (see Fig. 7). The complexity of this operator is $O(|F| \cdot |S|)$.

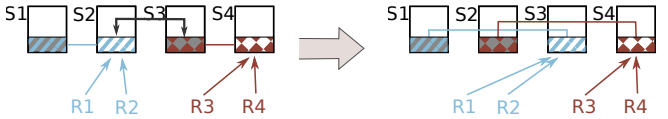


Figure 6: Example of VNFs swap neighborhood: VNFs on server $S2$ and $S3$ are swapped. If a VNF is a master, then all its assigned requests are redirected to the new server.

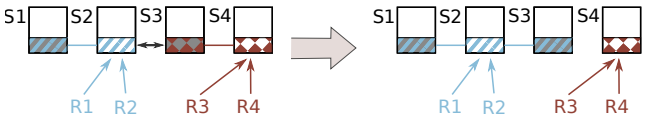


Figure 7: Example of slave VNFs swap neighborhood: a slave is removed from the placement in order to free resources for a slave of a different master VNF.

Requests swap. We also explore the neighborhood where requests are swapped (see Fig. 8): given a request we consider a subset of requests assigned to a different server and then swap the former with the latter. Similarly to the swap of VNFs, the complexity of this operator is $O(2^{|R|})$. However, by setting an upper bound of 1 to the cardinality of the swapped requests set we obtain a $O(|R|)$ operator.

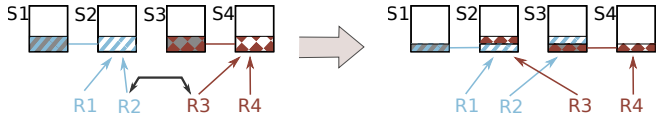


Figure 8: Example of requests swap neighborhood: requests $R2$ and $R3$ are swapped changing the respective servers. When swapping a request, a new VNF instance is created if none existed on the new server.

Request move. In the last exploration we consider the neighbors where a request is simply moved to a different server (see Fig. 9). The complexity of this operator is $O(|S|)$.

In principle, even if all the operators are polynomial time our VNS algorithm is not. However, an upper bound k to the number of iterations can be set, obtaining a $O(k \cdot |R| \cdot |S| \cdot \max\{|R|, |F| \cdot |S|\})$ heuristic. Also, in the following we show that our VNS requires small computing time for NFVI of limited size and it can be parameterized to end within a time limit, making it suitable for both online and offline planning.

5.3. Shared protection extension

We now show how to obtain a placement with shared protection starting from one with dedicated protection. The algorithm, sketched in Algorithm 3, gets a placement with dedicated protection as input. Such a placement can be found with any of the previously presented methodologies. For each master VNF, the algorithm searches for already deployed slaves that could share the protection. If a slave has reserved enough resources, then it shares the protection with the selected master.

This procedure can be applied to any placement with dedicated protection, and it can also be applied at any temporary solution of the VNS algorithms.

In case a Common Cause Failure (CCF) distribution for the network elements is given, i.e., grouping sets of elements in shared failure risk groups associated with a CCF probability, then Algorithm 3 can be easily extended so as to allow protection resource sharing only if a certain CCF threshold is satisfied, or by prioritizing protection sharing to least CCF master-slave associations, or a mix of these two search strategies.

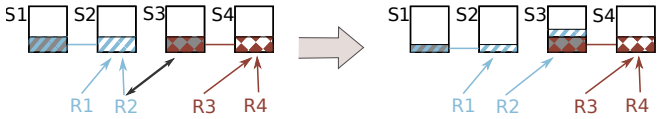


Figure 9: Example of request move neighborhood: request $R2$ is moved and assigned to a different server.

Algorithm 3 Sharing protection algorithm

```
1: function TO_SHARED(placement)
2:   for all master VNF v ∈ placement do
3:     for all slave VNF v' ∈ placement do
4:       if v' ≥ v then
5:         v' is shared and protects v in placement
6:       end if
7:     end for
8:   end for
9:   return placement
10: end function
```

6. Generalization for arbitrary topologies

We now show that when we are given a methodology to measure the availability of a VNF placement, our VNS algorithm can work as a black box having no knowledge of the underlying NFVI. In fact, NFVI orchestrators are expected to operate with an over-the-top approach with respect to the underlay network, moving requests and VNFs between servers, without changing the underlay NFVI network topology. Potentially, our orchestration algorithm could work on any NFVI network topology as the measure of the availability could be considered as an external tool, as depicted in Fig. 10. The combination of

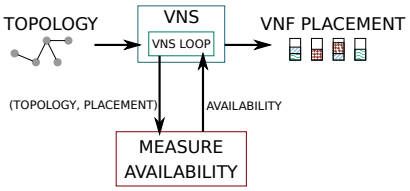


Figure 10: Illustration of our VNS algorithm used as a black box to optimize any NFVI: our VNS receives the description of the NFVI and a methodology to compute the availability of a placement on such an NFVI, producing a VNF placement.

our VNS orchestration algorithm and generic methodologies to provide the availability of a topology, such as [38], could lead to a generic optimization tool for VNF placement on any network.

To prove that our VNS algorithm is flexible enough to provide a high level of availability also on different topologies, we now propose a methodology to transform the hierarchical topology represented in Fig. 11 to a flat topology as the one depicted in Fig. 1 to measure its availability.

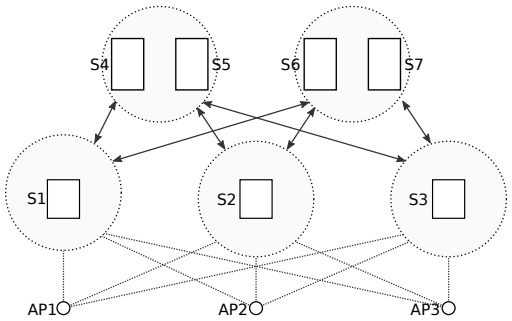


Figure 11: Example of hierarchical topology with 2 layers of clusters.

The transformation process is simple: to obtain a flat topology suitable to apply the VNS algorithm, we need to compute only the availability of the paths connecting the clusters of level 2 to the access points. Instead, the availability of the synchronization links between clusters of the same level is equal to 0, since those clusters have no link in the hierarchical topology. In addition to the notation defined in Section 4, let $C^{(i)} \subseteq C$ be the subset of all clusters at i-th level of the topology. For each pair of cluster $c \in C^{(2)}$ and access point $p \in P$, the availability of the access link (c, p) is the probability that at least one path going from c to p succeeds, that is

$$a_{cp}^{L_p} = 1 - \prod_{c' \in C^{(1)}} (1 - a_{cc'}^{L_C} \cdot a_{c'p}^{L_P}).$$

In the following we will use this methodology to simulate our optimization process also on a hierarchical topology.

7. Simulation

We evaluate empirically the quality of our methodologies: the greedy heuristic using three different policies (best fit, first fit, and best availability), the VNS algorithm, and the mathematical programming formulation as a MIP. However we could run our MIP only on small instances with 3 or 4 servers and 50 requests. For what concerns load balancing policies, because of Eq. (1) splitting demands reduces the availability. Therefore, we first run the algorithms using the demand load balancing policy, and allowing split only if the former fails to assign all the requests. In fact, due to Observation 1, splitting requests always allows a feasible placement. However, in our simulation we never recurred to split demand load balancing policy, because all algorithms were able to assign all requests for any instance of our dataset. All methodologies are implemented in C++, while CPLEX 12.6.3 [39] is used to solve the MIP. The simulations have been conducted on Intel Core i7-6700K CPU at 4.00GHz with 32 GB RAM. We also produced a graphical DSS tool integrating the VNS and the greed algorithms (in python) working on arbitrary 2-hop topologies and made it available in [6].

7.1. Dataset generation

We generated a random dataset consisting of instances that differ for the number of requests, total amount of computing resources, and availabilities of the network components. We set the number of VNF types provided by our NFVI to $|F| = 5$. We assumed an NFVI with 3 clusters ($|C| = 3$) and 3 access points ($|P| = 3$). Each request has a random demand $d_r \in [1, 10]$, while each server has a random capacity $q_s \in [75, 125]$. The availabilities of all the components of our NFVI are selected between $\{0.9995, 0.9999, 0.99995, 0.99999\}$ as in [40, 29, 41].

We generated 30 instances for each combination of:

- number of requests $|R| = \{50, 100, 200, 300, 400, 500\}$;
- number of access points for each request $|P_r| \in \{1, 2, 3\}$.

The number of servers depends on the number of requests, the total amount of the demands, and the random capacities: we generated a set of servers such that their capacities are enough to serve all the demands $Q = \sum_{s \in S} q_s \geq \sum_{r \in R} d_r$. Note that such condition guarantees the feasibility only when splitting requests is allowed. Servers are randomly distributed among all the clusters, in such a way that for each pair of clusters c and c' we have $|S_c - S_{c'}| \leq 1$. Under these conditions we obtained instances with around 3 servers when $|R| = 50$ and 28 servers when $|R| = 500$.

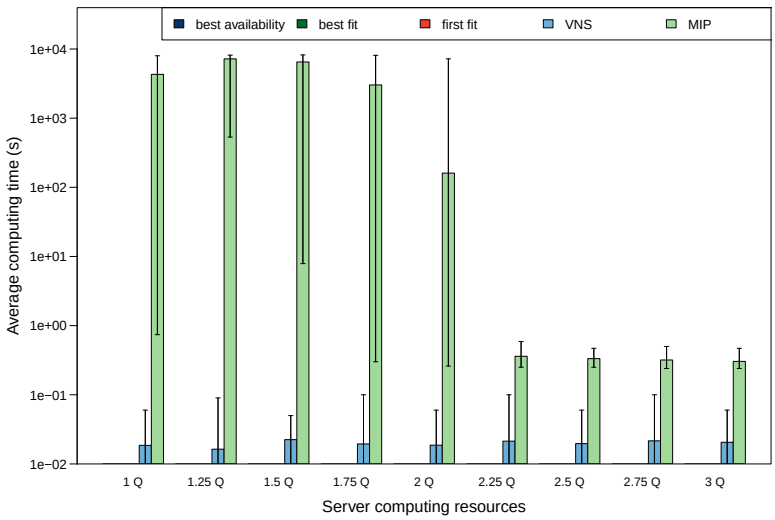
7.2. Comparison on small instances

We first evaluate the quality of our VNS heuristic against the solutions obtained by the MIP solver and the greedy heuristics. We set a time limit of two hours to the MIP solver, which was enough to obtain optimal or near optimal solution for all instances. In order to study how the NFVI behaves on different levels of congestion, we let its computing resources to grow from an initial value of $Q = \sum_{s \in S} q_s$, to $3 \cdot Q$, with a step of $0.25 \cdot Q$. Our algorithms assigned all requests without splits.

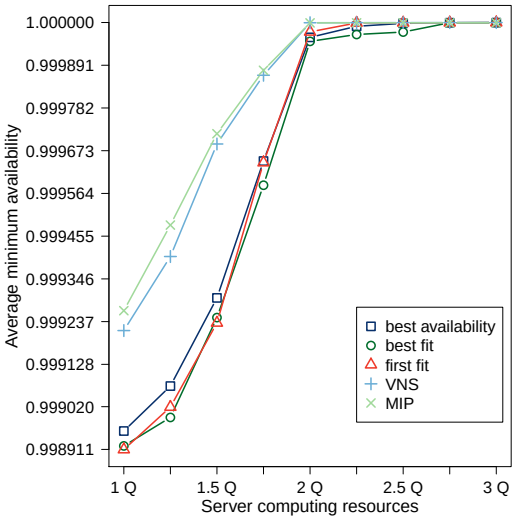
In Fig. 12(a) we show the average computing time of the algorithms: while the MIP hits several times the time limit, computing times are negligible for all the heuristics, and VNS can be considered as a good alternative for online orchestration when the set of the requests is small. The optimization problem seems harder when the amount of computing resources is scarce: in fact, the average computing time of the MIP is closer to the time limit when the overall capacity is less than $2 \cdot Q$. Instead, with higher quantities of resources the MIP always finds the optimal solution within the time limit.

In Fig. 12(b) we show that the results of our VNS heuristic are close to the MIP ones, while there is a significant gap between the latter and the greedy heuristic ones. In fact, both the MIP and the VNS succeed in finding solutions with an availability of three nines even with scarce resources. Eventually all the algorithms reach an high level of availability when computing resources are doubled.

In Fig. 13 we show the variation of the availability when the number of access points for each request increases: in Fig. 13(a), Fig. 13(b), and Fig. 13(c) we report the average availability when requests can be routed to the NFVI using 1, 2, and 3 access points, respectively. The path protection obtained by using more than one access point substantially increases the level of the availability. However, having more than 2 access points does not provide additional benefits.

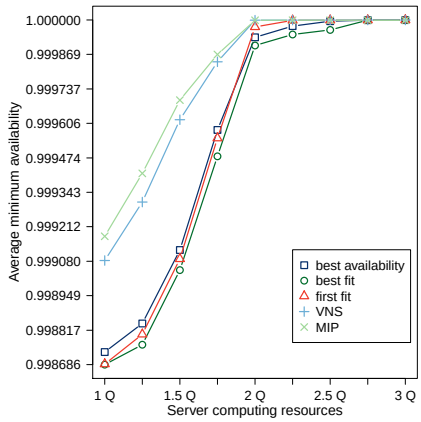


(a) Average computing time (vertical axis in log. scale).

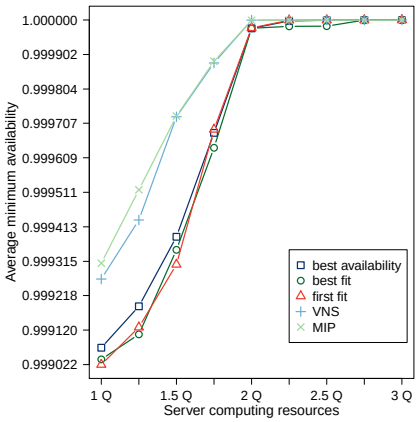


(b) Average minimum availability.

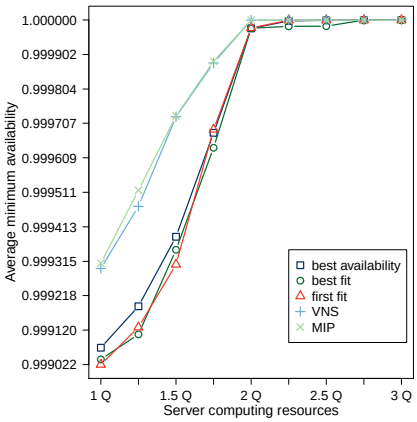
Figure 12: Results for instances with 50 requests.



(a) 1 access point for request.



(b) 2 access points for request.



(c) 3 access points for request.

Figure 13: Average minimum availabilities for instances with 50 requests and different number of access points for each request.

7.3. Scaling up the number of requests

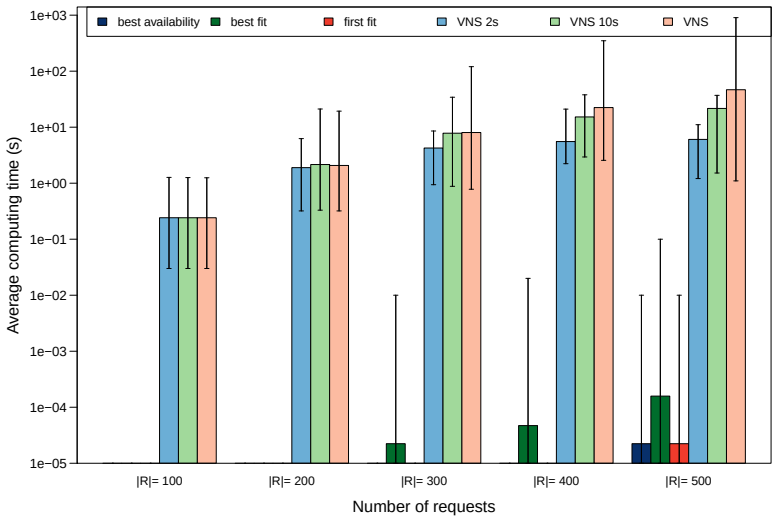
In a second round of experiments, we evaluated how our VNS algorithm behaves when scaling up the number of requests. However, when the number of servers increases its is not possible to use the MIP. Therefore, in the following analysis we compare the results of our VNS algorithm to the greedy ones only. In addition, since our VNS algorithm has not polynomial time complexity, we include in the comparison the results obtained by setting a time limit of 10s at the exploration of each starting solution.

We can first observe in Fig. 14(a) that the computing time of the VNS algorithm grows exponentially when the number of requests increases. Indeed setting a time limit reduces the overall computing time, which is always less than a minute.

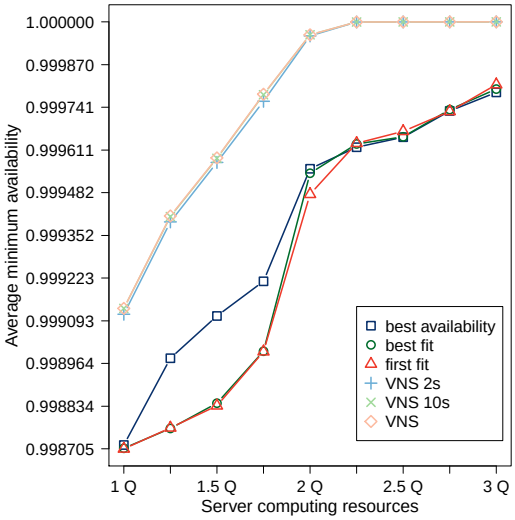
In Fig. 14(b) we show that on average there is always a substantial gap between the results obtained by our VNS algorithms and the greedy heuristics. We can also observe that on average the VNS time limit does not penalize substantially the results. Therefore our VNS algorithm with time limit can reduce computing times with minimal loss in availability.

In Figures 15 and 16, we show the results on instances having from 100 to 500 requests. We observe that the greedy heuristics progressively loose in quality of the solutions, and the gap from the VNS algorithms increases with both the number of the requests and the computing resources.

Finally in Fig. 17 we show only the results concerning the VNS algorithm without time limit and how it behaves when both the number of requests and the overall capacity increase. We can observe that the quality of the solutions provided by our algorithm is not affected by the increasing of the size of the instances. On average, our VNS algorithm always provides placements with an availability of three nines even when resources are scarce, and it reaches four nines availability when the capacity is doubled.



(a) Average computing time (vertical axis in log. scale).



(b) Average minimum availability.

Figure 14: Results for instances with up to 500 requests.

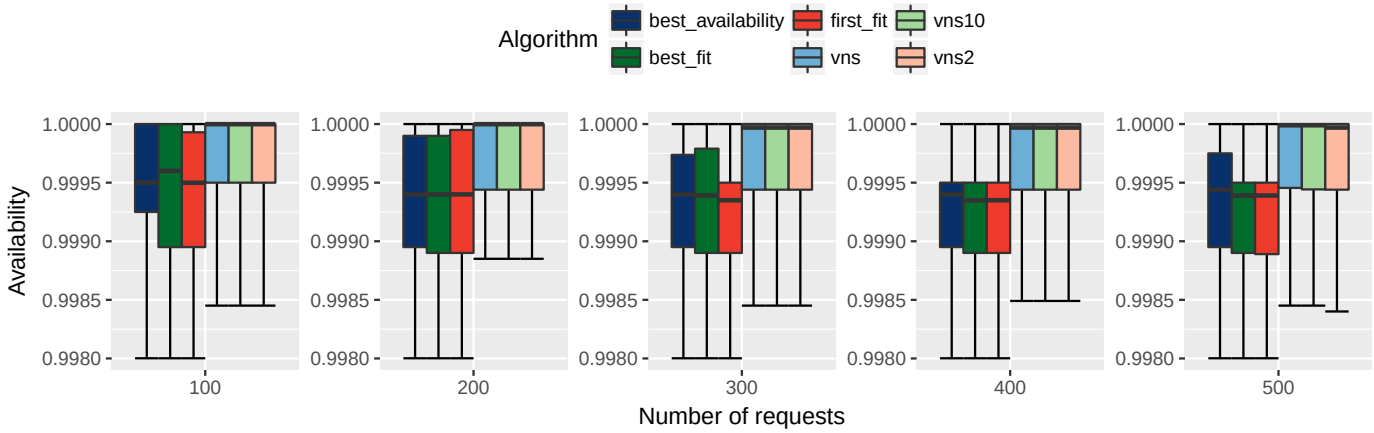


Figure 15: Distribution of availabilities for instances with up to 500 requests.

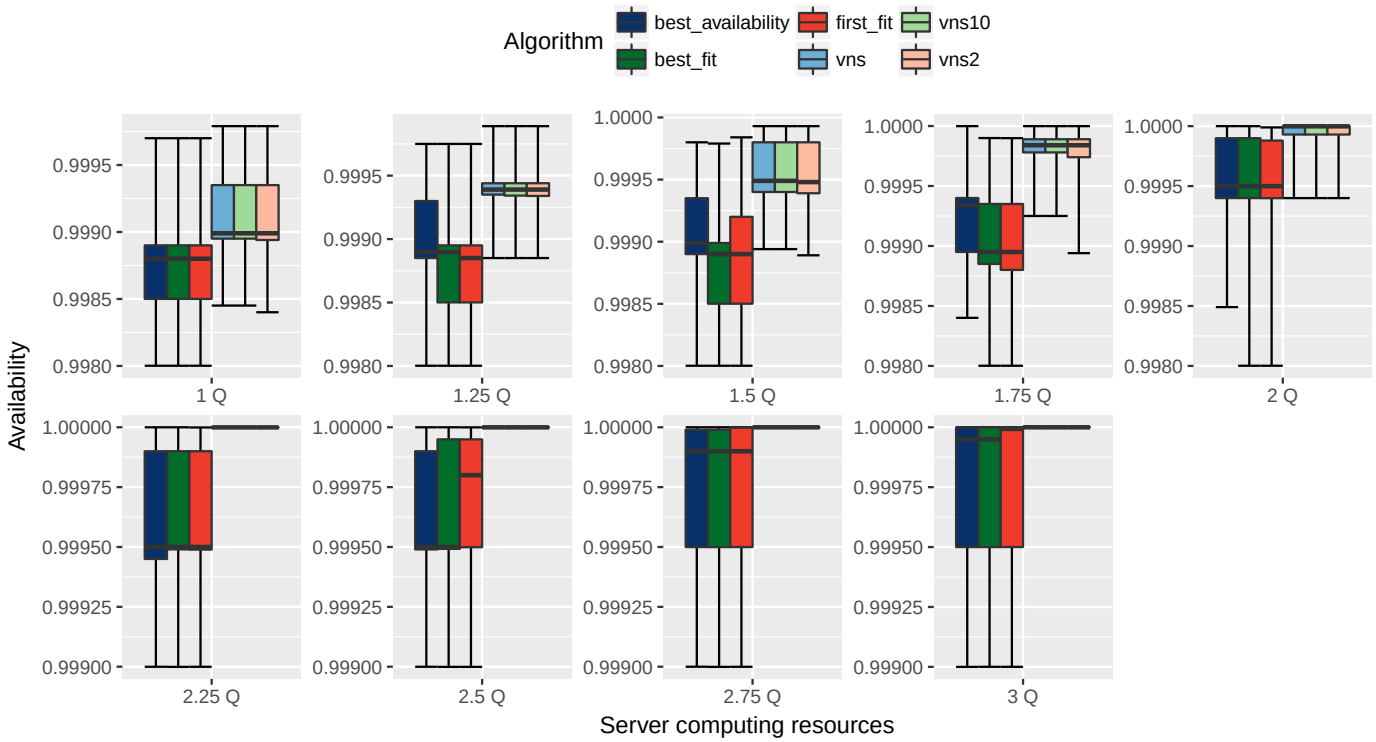


Figure 16: Distribution of availabilities for instances with up to 500 requests.

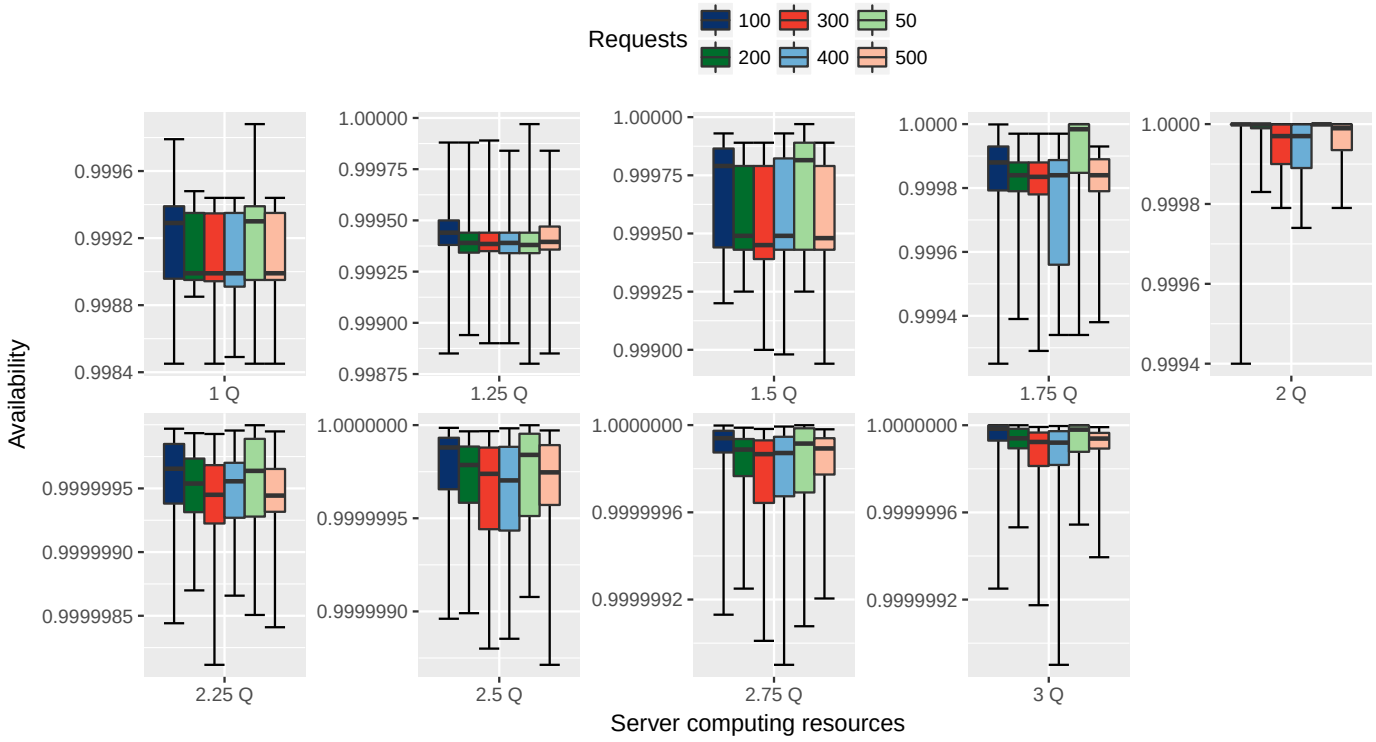


Figure 17: Distribution of availabilities for instances with up to 500 requests for VNS algorithm only.

7.4. Shared vs dedicated protection

Given the results of both greedy and VNS algorithms, we applied the sharing protection algorithm in Algorithm 3 to obtain placements with shared protection, and then we computed the upper bound to their availability. In Fig. 18 we show the potential availability that can be reached with shared protection: results confirm that when shared protection is enabled, VNS algorithm still provides better solutions than the greedy algorithms. However, shared protection allows also greedy algorithms to provide, on average, solutions with at least an availability of 99.9%.

Also, in Fig. 19 we show the failure probability reduction, i.e., an availability gain: the ratio between the increase in availability of a solution using shared protection and the probability of failure of the same solution but without shared protection. In other words, it answers the question "how much can we reduce the probability of a failure using shared protection?", and we can observe that greedy algorithms are the ones that benefit the most from a shared protection scenario, up to 45% failure probability reduction.

7.5. Flat vs hierarchical topologies

In order to test the flexibility of our algorithms, we compared the availability reached by our VNS algorithm with different topologies and settings. Besides the flat topology, we simulated a 2-layer topology like the one depicted in Fig. 11, with 4 access clusters and 2 core clusters. Each access cluster has exactly one server, while core clusters have potentially many servers. The difference between the access cluster servers and the core cluster ones is that the latter have higher availabilities and twice as many computing resources than the former. The number of servers still depends on the overall demand, with up to 18 servers for instances with 500 requests.

For what concerns link availabilities for the 2-layer topology case, we simulated two different settings: in the first one, that we call *Hierarchical*, each link availability is randomly selected between the values $\{0.9995, 0.9999, 0.99995, 0.99999\}$. In the second setting instead, we distinguish between access clusters to simulate a case with fixed access clusters (e.g., attached or close to a classical fixed base station) and a case with deployable access clusters (e.g., in a vehicle as envisioned in forthcoming LTE-based or 5G public-safety networks): the first ones are close to core clusters but far from the access points, and therefore their links connecting to core clusters have an availability set to 0.99999, while the ones connecting to the access points have an availability set to 0.9995. Instead, deployable clusters behave in the opposite way: the availability is set to 0.9995 for links connecting to core clusters and to 0.9995 for the ones connecting to the access points. We call this second setting *Hierarchical asymmetric*.

We transform both hierarchical topologies to a flat one using the generalization proposed in Section 6, and compared the results to observe how our VNS algorithm behaves.

In Fig. 20 we can observe that on average, the availabilities provided by both flat and hierarchical topologies are similar, although when the number of requests increases the hierarchical topology offers on average a higher lever of availability.

In Fig. 21 we still show the availability reached by the three topologies when the server capacity increases: still their behaviours are similar, however when there is a surplus of computing resources, the flat topology outperforms the hierarchical ones.

Also, in both simulations the hierarchical topologies behave almost the same, showing that topologies with deployable stations can offer an equivalently high level of availability as fixed-station alternatives.

7.6. Network capacity

In our last run of simulation, we tested how the network capacity influences the availability of the placement obtained by our algorithms.

We assume to have a link between each pair of servers, each link having limited capacity. We also assume that to synchronize a master and each of its slaves, the capacity of the links connecting the corresponding servers is consumed.

We estimate the network capacity and the network consumption to be proportional to values q_s and d_r, respectively. In fact, such values represent the computing resources (CPU, RAM, ...) available on servers and required by client requests. Thus it is reasonable to assume that, for example, saving the state of a VNF using a higher quantity of RAM will consist of a higher usage of the network.

We first set a link capacity equal to $\sum_{s \in S} q_s / |S|$, i.e., the average servers' bit-rate processing resource capacity, for each link, and then we run our algorithms reducing such capacity to 90%, 75%, 50%, 25%, and 10%.

In Fig. 22 we show the average availability with different link capacities, and we can observe that all algorithms are affected by the reduction of the link capacity, but the VNS algorithm is the only one that guarantees an average availability of three nines in all scenarios.

8. Integration in OpenStack

In the frame of the FED4PMR French collaborative project public demonstrator on future public-safety networks – making use of edge computing to virtualize application servers and core network functions – we built a demonstration testbed composed of multiple OpenStack[2] clusters, each with one or two compute nodes. The OpenStack nodes are equipped with a Tacker NFV orchestrator module[3] allowing us to exploit the placement algorithms to deploy VNFs on the compute nodes.

We produced a video tutorial published in [6], where we also release the VNS algorithm and the online algorithms executable, integrated in a Graphical User Interface (GUI) used to interact with the NFVI orchestration platform. We provide in the following a complementary description of the integration and corresponding operational steps.

The OpenStack clusters are configured to be accessible from the outside (by default only local users can deploy VMs on the

[2] OpenStack opensource project: `https://www.openstack.org`.

[3] Tacker orchestrator: `https://wiki.openstack.org/wiki/Tacker`.

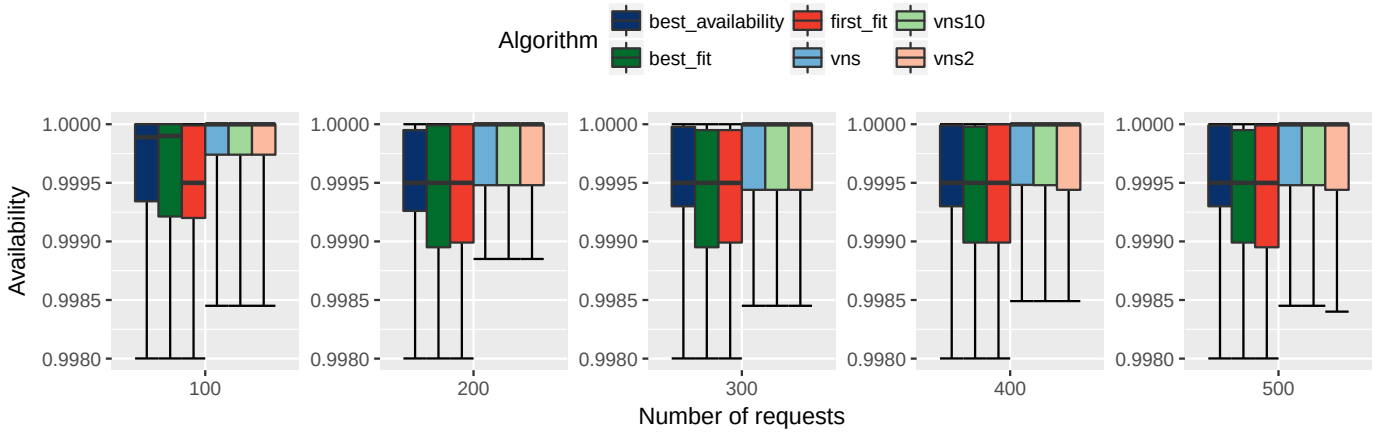


Figure 18: Upper bound to the availability obtained by our heuristics when shared protection is enabled.

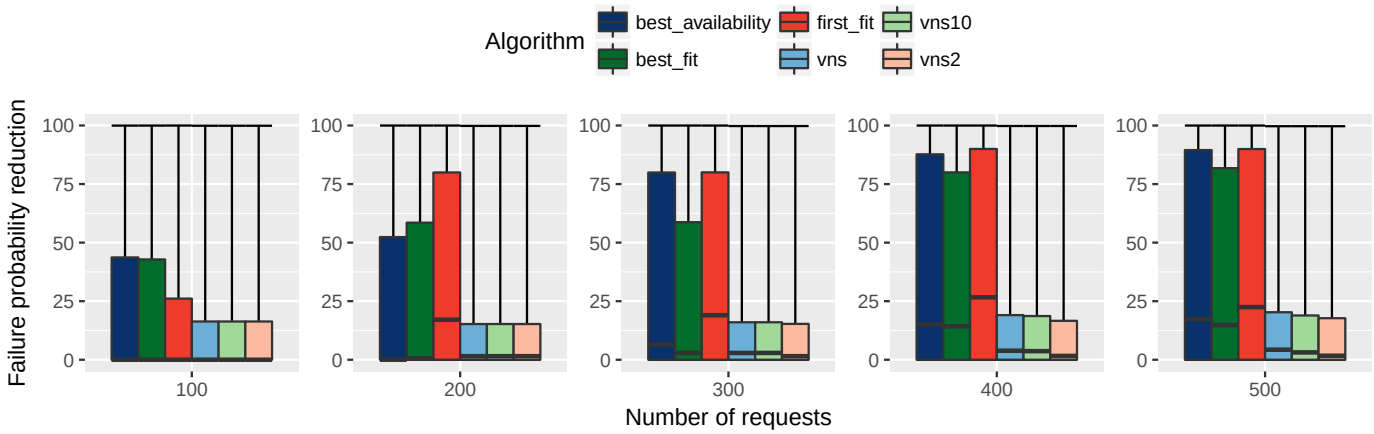


Figure 19: Potential failure probability reduction when shared protection is enabled (in percentage).

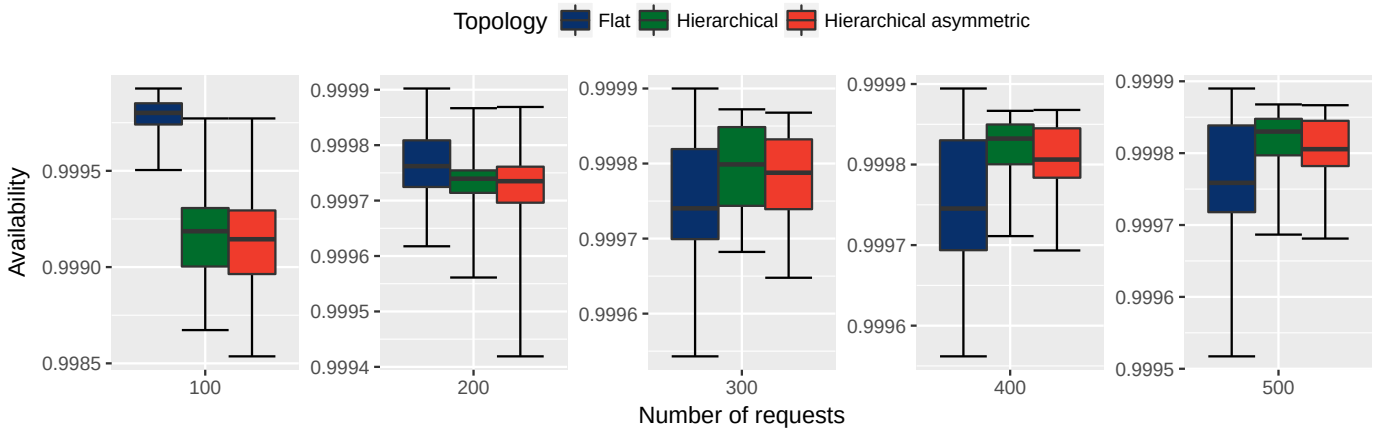


Figure 20: Availability of flat and hierarchical topologies when the number of requests increases.

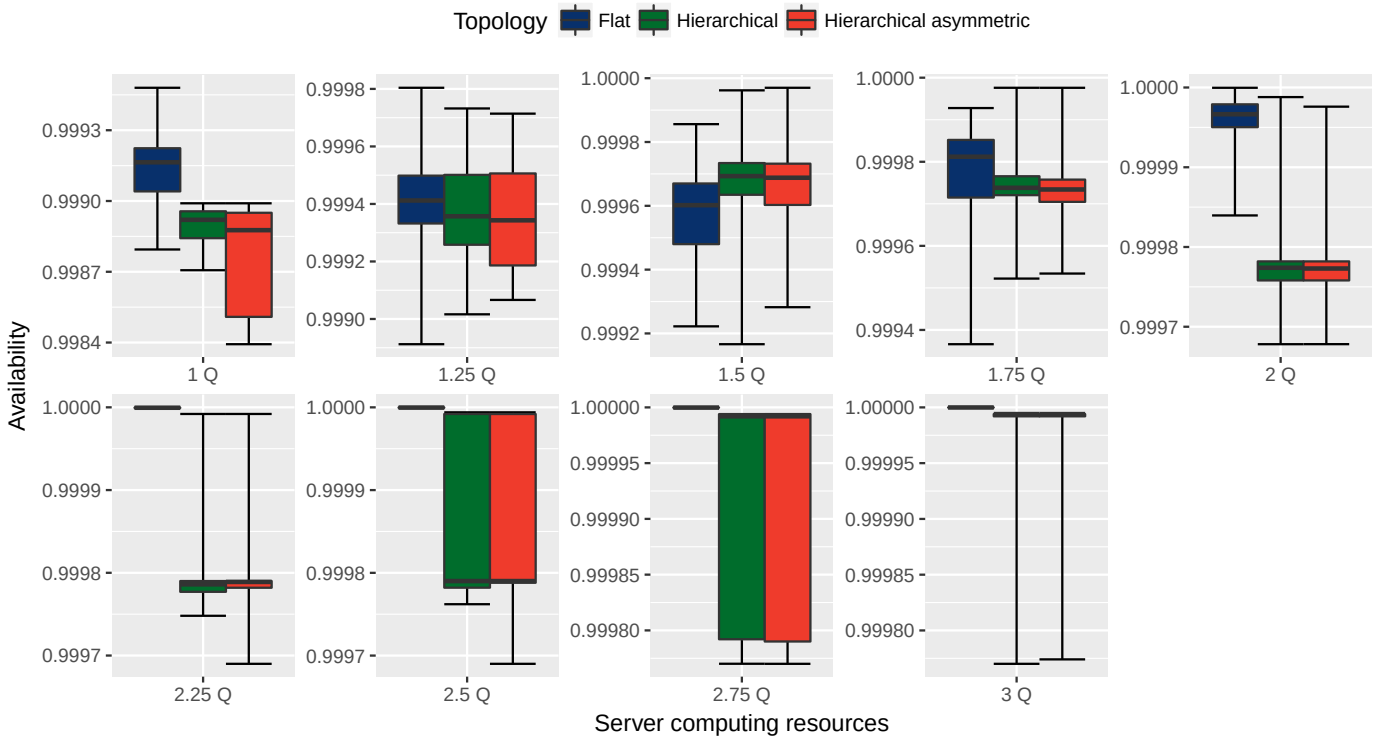


Figure 21: Availability of flat and hierarchical topologies when the server capacity increases.

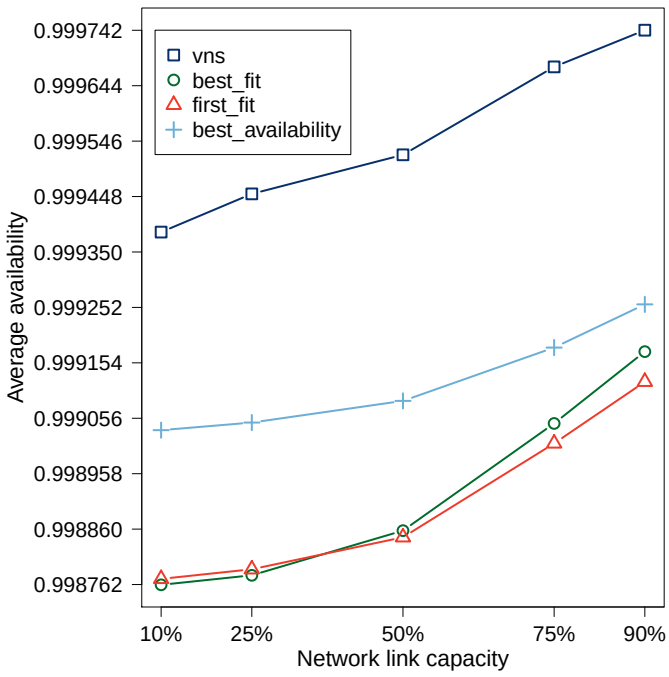


Figure 22: Average availability when the network link capacity decreases.

nodes). Tacker has multiple VNF Descriptors (VNFD) in its catalog and the OpenStack clusters are registered as Virtual Infrastructure Managers (VIMs).

The VNF placement requests are gathered from the operator in the NFVI system by using the NFV Orchestrator CLI; this registers a list of VNFs wished to be deployed, as well as two key indicators: 1) the availability target needed to be achieved and 2) the penalty occurred if this target is not reached. After adding one or more request, the operator will commit the request. This has two main effects: first, the NFV Orchestrator will gather up to date information on the nodes (e.g., available resources, availability) from the OpenStack clusters; moreover, Tacker is not yet able to provide this information from the VIMs, so the NFV Orchestrator has to get them directly at the source. Then, the NFV Orchestrator will send this information as well as the placement requests to the placement server (using an HTTP POST in YAML format). Finally, it will start a REST server to wait for the response.

The GUI of the placement server can display the topology of the clusters based on the information provided by the NFV Orchestrator API. There, the operator selects which algorithm best suits their needs. When ready, the placement server pushes the response to the NFV Orchestrator API (via an HTTP POST in YAML).

The NFV Orchestrator parses the response which contains the assignment for each VNF, the specific server (or servers if the algorithm decided that a master and one or more slave had to be launched) of a cluster where it should be deployed. This is passed on to Tacker which will transmit the order to deploy

the VNF to the correct OpenStack cluster.

9. Conclusion

In this paper we defined an availability-driven NFV orchestration problem and related resolution approaches. We proposed VNF protection strategies, ranging from dedicated VNF protection to shared heterogeneous VNF protection, and provided quantitative models based on a probabilistic approach to offer estimations of the availability of an NFV Infrastructure.

We modeled the optimization problem arising within this context, proving that it is $\mathcal{NP}$-hard and providing an integer linear formulation. We designed and evaluated heuristics based on both greedy and Variable Neighborhood approaches. By extensive simulations we proved that our VNS algorithm solves instances with a realistic size, and it finds nearly optimal solutions while being orders of magnitude faster than a MIP.

We highlighted the substantial gap between the availability obtained using classic greedy policies, and the one obtained with a more advanced VNS algorithm, when the NFV infrastructure is under computing congestion. We showed that our VNS approach allows to obtain high level of availability even when the computing time is limited, proving that even few iterations of local search substantially improve the solution quality. We also showed that simpler greedy heuristics can benefit from our shared protection scheme, although the VNS algorithm always improves their results. Finally, we highlighted the flexibility of our VNS algorithm, proving that it works with potentially any topology, if the latter is provided with a methodology to estimate its availability.

Acknowledgments

This article is based upon work from COST Action CA15127 ("Resilient communication services protecting end-user applications from disaster-based failures - RECODIS") supported by COST (European Cooperation in Science and Technology). This work was funded by the Reflexion (contract no. ANR-14-CE28-0019) and the FED4PMR projects. We thank K. Phemius from Thales Communication & Security for his effort in the integration and testing of our framework in OpenStack, and M. Tornatore and P. Fouilhoux for their insightful comments.

References

[1] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, F. Huici, ClickOS and the art of network function virtualization, in: USENIX NSDI 2014.

[2] J. Mo, B. Khasnabish, NFV Reliability using COTS hardware.

[3] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, R. Boutaba, Network function virtualization: State-of-the-art and research challenges, IEEE Communications Surveys Tutorials 18 (1) (2016) 236–262.

[4] Delivering High Availability in Carrier Grade NFV Infrastructures, VMWare Technical White Paper.
URL `https://www.vmware.com/files/pdf/techpaper/vmware-vcloud-nfv-high-availability.pdf`

[5] ETSI, ETSI GS NFV-REL 003 V1.1.1 Network Functions Virtualisation (NFV); reliability; report on models and features for end-to-end reliability, Tech. rep.

[6] HA-NFV simulator software [online].
URL `http://ha-nfv.roc.cnam.fr`

[7] M. Casazza, P. Fouilhoux, M. Bouet, S. Secci, Securing virtual network function placement with high availability guarantees, in: Proc. of IFIP Networking 2017, 2017, pp. 1–9.

[8] B. Han, V. Gopalakrishnan, G. Kathirvel, A. Shaikh, On the Resiliency of Virtual Network Functions, IEEE Communications Magazine 55 (7) (2017) 152–157.

[9] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, A. Warfield, Remus: High availability via asynchronous virtual machine replication, in: USENIX NSDI 2008.

[10] J. F. Riera, J. Batallé, J. Bonnet, M. Días, M. McGrath, G. Petralia, F. Liberati, A. Giuseppi, A. Pietrabissa, A. Ceselli, A. Petrini, M. Trubian, P. Papadimitrou, D. Dietrich, A. Ramos, J. Melián, G. Xilouris, A. Kourtis, T. Kourtis, E. K. Markakis, TeNOR: Steps towards an orchestration platform for multi-PoP NFV deployment, in: IEEE NetSoft 2016.

[11] B. Jennings, R. Stadler, Resource management in clouds: Survey and research challenges, J. Network and Systems Management 23 (3) (2015) 567–619.

[12] X. Meng, V. Pappas, L. Zhang, Improving the scalability of data center networks with traffic-aware virtual machine placement, in: IEEE INFOCOM 2010.

[13] M. Alicherry, T. V. Lakshman, Network aware resource allocation in distributed clouds, in: IEEE INFOCOM 2012.

[14] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, E. Silvera, A stable network-aware vm placement for cloud systems, in: IEEE/ACM CCGrid 2012.

[15] A. Basta, W. Kellerer, M. Hoffmann, H. J. Morper, K. Hoffmann, Applying NFV and SDN to LTE mobile core gateways, the functions placement problem, in: Proc. 4th Workshop on All Things Cellular: Operations, Applications, and Challenges, 2014.

[16] R. Cohen, L. Lewin-Eytan, J. S. Naor, D. Raz, Near optimal placement of virtual network functions, in: IEEE INFOCOM 2015.

[17] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, L. P. Gaspary, Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions, in: IEEE/IFIP IM 2015.

[18] B. Addis, D. Belabed, M. Bouet, S. Secci, Virtual network functions placement and routing optimization, in: IEEE CloudNet 2015.

[19] F. Carpio, S. Dhahri, A. Jukan, Vnf placement with replication for load balancing in nfv networks, in: 2017 IEEE International Conference on Communications (ICC), 2017, pp. 1–6.

[20] M. Abu-Lebdeh, D. Naboulsi, R. Glitho, C. W. Tchouati, Nfv orchestrator placement for geo-distributed systems, in: 2017 IEEE 16th International Symposium on Network Computing and Applications (NCA), 2017, pp. 1–5.

[21] X. Song, X. Zhang, S. Yu, S. Jiao, Z. Xu, Resource-efficient virtual network function placement in operator networks, in: GLOBECOM 2017 - 2017 IEEE Global Communications Conference, 2017, pp. 1–7.

[22] D. Li, P. Hong, K. Xue, J. Pei, Virtual network function placement considering resource optimization and sfc requests in cloud datacenter, IEEE Transactions on Parallel and Distributed Systems (2018) 1–1.

[23] E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, D. H. Lorenz, Guaranteeing High Availability Goals for Virtual Machine Placement, in: ICDCS 2011.

[24] A. Israel, D. Raz, Cost aware fault recovery in clouds, in: IEEE/IFIP IM 2013.

[25] Y. Zhu, Y. Liang, Q. Zhang, X. Wang, P. Palacharla, M. Sekiya, Reliable resource allocation for optically interconnected distributed clouds, in: IEEE ICC 2014.

[26] R. S. Couto, S. Secci, M. E. M. Campista, L. H. M. Costa, Server placement with shared backups for disaster-resilient clouds, Computer Networks 93, Part 3 (2015) 423–434.

[27] H. A. Alameddine, S. Ayoubi, C. Assi, Protection plan design for cloud tenants with bandwidth guarantees, in: DRCN 2016.

[28] A. Hmaity, M. Savi, F. Musumeci, M. Tornatore, A. Pattavina, Protection strategies for virtual network functions placement and service chains provisioning, Networks 70 (4) (2017) 373–387.

[29] S. Yang, P. Wieder, R. Yahyapour, Reliable virtual machine placement in distributed clouds, in: RNDM 2016.

[30] A. Hmaity, M. Savi, F. Musumeci, M. Tornatore, A. Pattavina, Virtual network function placement for resilient service chain provisioning, in: 2016

8th International Workshop on Resilient Networks Design and Modeling (RNDM), 2016, pp. 245–252.

[31] A. Hmaity, M. Savi, F. Musumeci, M. Tornatore, A. Pattavina, Protection strategies for virtual network functions placement and service chains provisioning, Networks 70 (4) 373–387.

[32] P. Vizarreta, M. Condoluci, C. M. Machuca, T. Mahmoodi, W. Kellerer, Qos-driven function placement reducing expenditures in nfv deployments, in: 2017 IEEE International Conference on Communications (ICC), 2017, pp. 1–7.

[33] L. Yala, P. A. Frangoudis, G. Lucarelli, A. Ksentini, Balancing between cost and availability for cdnaas resource placement, in: GLOBECOM 2017 - 2017 IEEE Global Communications Conference, 2017, pp. 1–7.

[34] N. Menakerman, R. Rom, Bin packing with item fragmentation, in: Algorithms and Data Structures: 7th Int. Workshop, 2001.

[35] A. Birolini, Reliability and Availability of Repairable Systems, 2004, pp. 160–258.

[36] IEC, Analysis techniques for dependability – Reliability block diagram method.

[37] P. Silva, C. Perez, F. Desprez, Efficient heuristics for placing large-scale distributed applications on multiple clouds, in: IEEE/ACM CCGrid 2016.

[38] S. Ktari, S. Secci, D. Lavaux, Bayesian diagnosis and reliability analysis of private mobile radio networks, in: 2017 IEEE Symposium on Computers and Communications (ISCC), 2017, pp. 1245–1250.

[39] IBM, CPLEX user's manual - version 12 release 6.

[40] P. Gill, N. Jain, N. Nagappan, Understanding network failures in data centers: Measurement, analysis, and implications.

[41] Q. Zhang, M. F. Zhani, M. Jabri, R. Boutaba, Venice: Reliable virtual data center embedding in clouds, in: IEEE INFOCOM 2014.